

A Deep Learning Approach for Forecasting Cost Estimate at Completion (EAC)
in Construction Projects

by

Denisse Magdalena Diaz Merino

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Construction Engineering and Management

Department of Civil and Environmental Engineering
University of Alberta

© Denisse Magdalena Diaz Merino, 2024

Abstract

Inaccurate cost forecasting is a significant issue that can lead to potential budget overruns, cash flow problems, poor stakeholder relationships, and financial losses for construction execution companies. To improve cost forecasting accuracy, this research proposes a deep-learning framework structured into five phases, starting with exploring the literature review and finishing with the practical application of the developed model. This framework leverages historical data from completed projects and deep-learning algorithms to enhance the cost estimate at completion accuracy.

First, this study explores the literature on previous approaches for cost forecasting and examines factors that influence cost forecasting. Second, it analyses current practices in the industry, gathers historical data, and establishes a data acquisition model at the level of individual work packages. Third, it develops a computerized model, including data preprocessing, designing, and building forecasting models based on deep learning algorithms. It also develops a graphical user interface (GUI) to store generated data and deploy the deep learning model. Fourth, model application and verification are performed using the dataset to select the optimal forecasting algorithm and compare results with traditional earned value methods. Finally, the GUI is applied to deploy the cost forecasting model at the work package level.

The study results demonstrate that the Gated Recurrent Unit (GRU) algorithm significantly outperforms traditional cost forecasting methods. The GRU model achieved Mean Absolute Percentage Errors (MAPE) of 7.38%, 6.14%, and 3.97% for the concrete, backfill, and piping work packages, respectively. In contrast, the Earned Value Management (EVM) method yielded MAPE values of 11.32%, 13.42%, and 13.3% for the same work packages. This study demonstrates the effectiveness of the deep learning model in accurately predicting cost forecasting for ongoing

construction projects. By integrating deep learning algorithms with a comprehensive analysis of historical cost data, the proposed framework offers a methodological foundation for future innovation in cost forecasting analytics.

Preface

This thesis is an original work by Denisse Magdalena Diaz Merino. No part of this thesis has been previously published.

Acknowledgements

First and foremost, I would like to thank God for all his blessings throughout this academic journey.

I am profoundly thankful to my supervisor, Dr. Ahmed Hammad, for his unwavering support, guidance, and encouragement. Dr. Hammad has been an exceptional mentor, always providing kind and constructive feedback to improve my research.

I am also grateful to my examining committee members, Dr. Leila Hashemian, Dr. Yasser Mohamed, and Dr. Ali Imanpour, for their invaluable insights and for reviewing my dissertation. Their feedback has been crucial in refining my research.

A special thank you to my husband, Cristhian Laura, whose support has been invaluable. We spent countless hours discussing and debating my topic, motivating me to learn new tools and techniques to enhance my work. I am profoundly grateful to my family; their love and enthusiasm have helped me complete this journey.

Dedication

I dedicate this research to my two greatest loves, my children Miranda and my baby on the way. I hope to show them that nothing is impossible when you truly desire to achieve something. As inspired by Thomas Edison's words, "Success is 1% inspiration and 99% persistence."

Table of Contents

Abstract	ii
Preface	iv
Acknowledgements	v
Dedication	vi
List of Tables	x
List of Figures	xi
Chapter 1: Introduction	1
1.1 Background	1
1.2 Problem Statement	3
1.3 Research Objectives	4
1.4 Research Methodology	5
1.5 Thesis Organization	7
Chapter 2: Literature Review	8
2.1 Introduction	8
2.2 Project Cost Management in Construction	8
2.3 Cost Forecasting Techniques	9
2.3.1 Earned Value Method	10
2.3.2 The Regression-Based Approach	15
2.3.3 Bayesian Statistics	17
2.4 Machine Learning	20
2.4.1 Conceptual Machine Learning Terms	20
2.4.2 Types of Machine Learning Algorithms	21
2.4.3 Machine Learning Algorithms Used for Cost Forecasting	24

2.5	Emergence of Deep Learning	26
2.6	Assessing the Performance of Machine Learning Algorithms	30
2.7	Machine Learning Application for Cost Forecasting	32
2.8	Summary and Research Gaps.....	37
Chapter 3:	Development of Conceptual Model for Cost Forecasting	39
3.1	Introduction	39
3.2	Current Cost Forecasting Practices	41
3.3	Key Factors Influencing EAC	43
3.4	Proposed Data Acquisition Model	48
3.4.1	Entity Relationship Diagram (ERD)	49
3.5	Data Collection Process	52
3.6	Data Description.....	53
3.7	Limitations in the Acquired Dataset.....	57
Chapter 4:	Development of Computational Model for Cost Forecasting	59
4.1	Introduction	59
4.2	Data Preprocessing.....	59
4.2.1	Grouping of Work Package Dataset	60
4.2.2	Dealing with Missing Values in Time Series	61
4.2.3	Outlier Detection.....	62
4.2.4	Correlation Analysis and Feature Selection	62
4.2.5	Time Series Data for Supervised Learning Problem	64
4.2.6	Max-Min Normalization and Data Splitting	65
4.3	Developing a Deep Learning-Based Cost Forecasting Model	67
4.3.1	Deep Learning for Time Series Forecasting	68

4.3.2	Cost Forecasting Model Design	72
4.4	Graphical User Interface for Cost Forecasting Model	76
4.4.1	Designing GUI Application with Tkinter	77
4.5	Integrating SQL into GUI	91
4.6	Project Forecasting Section	92
Chapter 5:	Application, Verification and Validation	94
5.1	Introduction	94
5.2	Selecting the Optimal Forecasting Model.	94
5.3	Applying the Forecasting Model to the Testing Dataset.....	98
5.4	Comparative Analysis of Forecasting Models and EVM Methods.....	107
5.5	Sensitivity Analysis	114
Chapter 6:	Conclusion.....	117
6.1	Research Summary.....	117
6.2	Research Contributions	120
6.2.1	Academic Contributions	120
6.2.2	Industrial Contributions:.....	120
6.3	Limitations	121
6.4	Future Work and Recommendations	122
References	124
Appendix A:	Python Scrip for Deep Learning Model.....	135
Appendix B:	Python Code for GUI Creation, SQLite, and Model Deployment.....	140

List of Tables

Table 2.1 EVM Terminology.....	11
Table 2.2 EAC Equations Summary	14
Table 2.3 Performance Measure Equations	31
Table 2.4 Summary of Machine Learning Approach	36
Table 3.1 Factors Influencing Project Cost Forecasting	44
Table 3.2 Preliminary Inputs and Output Available	47
Table 3.3 Initial Dataset Overview	54
Table 3.4 Actual Cost at Completion Bounds.....	54
Table 3.5 Earned Value Bounds	55
Table 3.6 Duration Bounds in Weeks.....	56
Table 4.1 Forecasting Model Features Variables.....	64
Table 5.1 Model Results Comparison ETC Results.....	95
Table 5.2 Model Results Comparison EAC Results	95
Table 5.3 Results of ACC vs. Deep Learning Models.....	97
Table 5.4 Results of Concrete-GRU on Testing Data.....	100
Table 5.5 Results of Piping-GRU on Testing Data	104
Table 5.6 Results of Backfill-GRU on Testing Data	106
Table 5.7 Comparative Analysis of EAC Predictions for Concrete WP	109
Table 5.8 Comparative Analysis of EAC Predictions for Piping (HDPE) WP.....	111
Table 5.9 Comparative Analysis of EAC Predictions for Backfill WP.....	113

List of Figures

Figure 1.1 Phases and Stages of the Project	2
Figure 1.2 Research Methodology	6
Figure 2.1 EVM components (Project Management Institute, 2019).....	10
Figure 2.2 Gompertz Growth Model (Ead, 2020)	16
Figure 2.3 Machine Learning Algorithm Types	22
Figure 2.4 ANN architecture (Tyagi & Abraham, 2022)	25
Figure 2.5 AI, Machine Learning, and Deep Learning Overview	27
Figure 2.6 LSTM cell (Fan et al., 2020).....	28
Figure 2.7 GRU Network (Yamak et al., 2019)	30
Figure 3.1 Conceptual Model for Cost Forecasting	39
Figure 3.2 DAM and Forecasting Model Development Methodology	41
Figure 3.3 Breakdown Structure Schema	50
Figure 3.4 Entity Relationship Diagram	51
Figure 3.5 Actual Cost to Date _Concrete dataset	55
Figure 3.6 Earned Value Cumulate _Fill dataset	55
Figure 4.1 Forecasting Model as a Workflow	60
Figure 4.2 Missing Values Process	61
Figure 4.3 Outlier Detection Process	62
Figure 4.4 Spearman Correlation Analysis	63
Figure 4.5 Time Series Transformation	65
Figure 4.6 Data Split Diagram.....	67

Figure 4.7 RNN architecture	69
Figure 4.8 GRU Neural Network Setup	73
Figure 4.9 Code snippet for the Class ProjectCostManagementApp	79
Figure 4.10 Segment Python Code for Creating Navigation Menu	80
Figure 4.11 Methods include in the Project Setup Frame Design.....	81
Figure 4.12 Project Setup Frame.....	82
Figure 4.13 Error Message in Project Setup Frame	82
Figure 4.14 Work Package Setup Frame	83
Figure 4.15 Method Structure for Project Tracking in Tkinter GUI Application	85
Figure 4.16 Project Update Frame	86
Figure 4.17 Current Baseline Window	86
Figure 4.18 Work Package Update Frame	87
Figure 4.19 Work Package Current Baseline Window	88
Figure 4.20 Weekly Progress per Work Package Frame	89
Figure 4.21 Actual Cost Details Window	90
Figure 4.22 SQL Database_Weekly Progress Table	90
Figure 4.23 Database Structure.....	91
Figure 4.24 Methods Developed for Cost Forecasting	92
Figure 4.25 Project Forecasting Frame	93
Figure 5.1 ACC vs Predicted EAC (Simple RNN, LSTM and GRU)	96
Figure 5.2 Actual vs. Predicted ETC for the Concrete Work Package	99
Figure 5.3 ACC vs. Predicted EAC for the Concrete Work Package	99

Figure 5.4 Cost Forecasting for the Concrete Work Package - Period 26	101
Figure 5.5 Cost Forecasting for the Concrete Work Package - Period 50	101
Figure 5.6 Actual vs. Predicted ETC for the Piping Work Package	103
Figure 5.7 ACC vs. Predicted EAC for the Piping Work Package.....	103
Figure 5.8 Cost Forecasting for the Piping Work Package - Period 45	103
Figure 5.9 Actual vs. Predicted ETC for the Backfill Work Package	105
Figure 5.10 ACC vs. Predicted EAC for the Backfill Work Package.....	105
Figure 5.11 Cost Forecasting for the Fill Work Package - Period 26	107
Figure 5.12 Comparative Analysis of EAC Predictions for Concrete WP	108
Figure 5.13 Comparative Analysis of EAC Predictions for Piping WP.....	110
Figure 5.15 Results - Sensitivity Analysis for Concrete Work Package	115
Figure 5.16 Results - Sensitivity Analysis for Piping (HDPE) Work Package.....	116
Figure 5.17 Results - Sensitivity Analysis for Backfill Work Package.....	116

Chapter 1: Introduction

1.1 Background

The construction sector is vital to nations' economic progress as it significantly contributes to the Gross Domestic Product (GDP), generates employment, and stimulates economic growth. In Canada, the construction industry is essential in developing and maintaining infrastructure and supporting the nation's growth. According to Statistics Canada's report in 2021, the construction industry contributed \$23.6 billion (8.0%) to Alberta's GDP. Additionally, in 2023, Alberta allocated \$260.4 million in capital towards the construction industry (Statistics Canada, 2024).

Construction projects are inherently complex and involve multiple phases throughout their lifecycle, including Pre-Engineering, Engineering, Procurement, Construction, Commissioning and Start-up, as defined by the Construction Industry Institute (CII). Each phase presents unique challenges and requirements, but this study explicitly emphasizes the execution phase. This phase is critical, as most of the project's budget is allocated here. Therefore, effective cost control is necessary to prevent budget overruns, which can significantly impact the project's finances.

Within the execution phase, controlling and monitoring is a pivotal stage that helps the project manager to identify potential deviations early, preventing delays and avoiding cost overruns (Atout, 2019). An ineffective performance at this stage significantly contributes to project failures (Nassar, 2005). As elucidated in Figure 1.1, a key outcome of cost control is the cost estimate at completion (EAC). EAC involves predicting the total project cost based on current data and trends. Accurate EAC allows construction project managers to detect deviations and make informed decisions throughout the project execution.

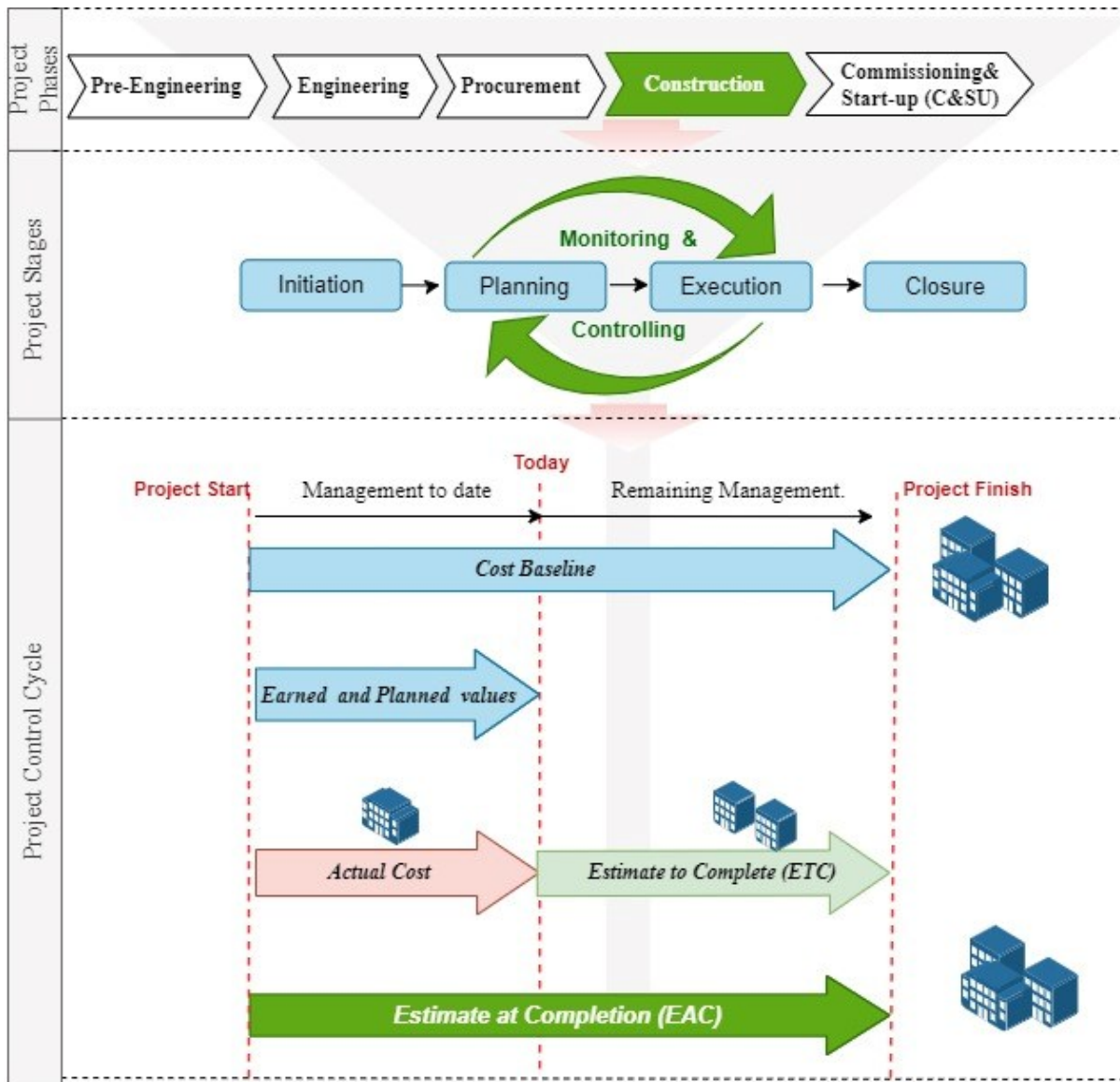


Figure 1.1 Phases and Stages of the Project

Traditional cost forecasting methods are often challenged by the dynamic nature of construction projects, where material cost escalation, labour productivity, unexpected weather conditions or unforeseen delays can complicate these forecasts, leading to potential budget overruns. Consequently, these challenges underscore the need for more sophisticated, adaptable methods to handle current construction projects' complexities. In this regard, deep learning, a machine learning subfield, has shown promising results in various sectors, such as finance and healthcare, for its

ability to analyze and make predictions from complex and voluminous data. Deep learning models are based on architectures like neural networks to uncover complex patterns in data, making them powerful tools for forecasting analytics in data-rich sectors.

In construction management, machine learning applications have primarily been investigated to predict resource allocation, ensure safety, and estimate costs and duration during the planning stage. Despite these advancements, deep learning integration into cost forecasting remains unexplored. This study aims to develop and validate a deep learning framework to enhance cost forecasting accuracy during the execution phase of construction projects, potentially setting a benchmark for future predictive analytics in project cost management.

1.2 Problem Statement

Forecasting the final cost in construction projects is often inaccurate due to several inherent difficulties and limitations in the current methods. One popular approach for cost estimated at completion in the construction sector is earned value management (EVM) (Fleming, 2016). This method uses indexed formulas to calculate the cost estimate at completion. Recent evidence indicates that the EVM method's cost prediction needs to be more accurate because it assumes that no further shifts in performance or risk will occur in the remaining work (Cheng & Hoang, 2014; Du et al., 2016; Kim, 2015). Additionally, EVM formulas do not consider factors such as weather conditions (Dastgheib et al., 2022), material price escalation, or labour productivity (Lema & Price, 1996), essential for capturing the uncertainties inherent in lengthy and complex construction projects. Another methodology is the bottom-up approach, which involves estimating the costs of remaining individual tasks or work packages and then aggregating them to determine the total project cost at completion. Although this method might be more accurate, it is time-consuming, data-intensive, complex for large projects, and carries the risk of overestimation.

Recent technological advancements have introduced machine learning methods as alternative approaches for EAC forecasting, aiming to address the limitations of traditional methodologies. Among these are Artificial Neural Networks (ANN) (Pewdum et al., 2009), Support Vector Machines (SVM) (Cheng et al., 2010, 2012, 2013; Cheng & Hoang, 2014), and Adaptive Neuro-Fuzzy Inference Systems (ANFIS) (Dastgheib et al., 2022). Despite this advance in machine learning techniques, their application in real-world construction projects is still limited. Also, according to recent studies, machine-learning techniques such as ANN, SVM, and ANFIS present limited performances when compared with deep-learning techniques (Kareem Kamoona & Budayan, 2019).

Deep learning is a promising option for forecasting time series because it can model complex, nonlinear temporal data (Hopfe et al., 2024). Several studies have proposed deep learning approaches for time series forecasting, demonstrating improvements in accuracy. However, its application is still minimal in the construction industry, especially for cost estimates at completion. Thus, this study aims to create a forecasting model using a deep learning architecture to improve the accuracy of predicting the final cost.

1.3 Research Objectives

This research aims to develop an integrated framework that utilizes deep learning algorithms to provide accurate cost-forecasting outcomes at the work package level. To achieve this, the study will analyze the various factors that impact the final construction cost, evaluate the performance of three deep learning algorithms using time series datasets, and develop a reliable forecasting model that the construction industry can use through a provided User Interface to enhance cost control at the work package level.

1.4 Research Methodology

This study was conducted in five phases.

Phase 1

- Investigating the literature review regarding traditional approaches and alternative methods for cost forecasting in construction projects.
- Examining the factors influencing the cost estimate at completion significantly and assessing their applicability to the defined problem.

Phase 2

- Exploring the current practices in the construction industry for calculating cost estimate at completion of construction projects.
- Proposing a data acquisition model to gather data for the forecasting model at the work package level.
- Historical data collection from completed projects.
- Data description.

Phase 3

- Developing data preprocessing to tailor the data to the specific architecture of deep learning models.
- Designing and configuring forecasting models based on deep learning algorithms, including Long-Short-Term Memory, Simple Recurrent Networks, and Gate Recurrent Units.
- Engineering a graphical user interface for storing data generated during the project execution and deploying the deep learning model.

Phase 4

- Performing the models using the dataset according to the proposed computerized model.
- Selecting the optimal forecasting model based on their performance metrics.
- Deploying the selected model at the work package level for unseen data and comparing the result with traditional earned value methods.
- Verifying the proposed model through a sensitivity analysis.
- Saving the cost forecasting model.
- Applying the graphical user interface to predict the cost estimate at completion.

Figure 1.2 illustrates the research methodology and details for each phase.

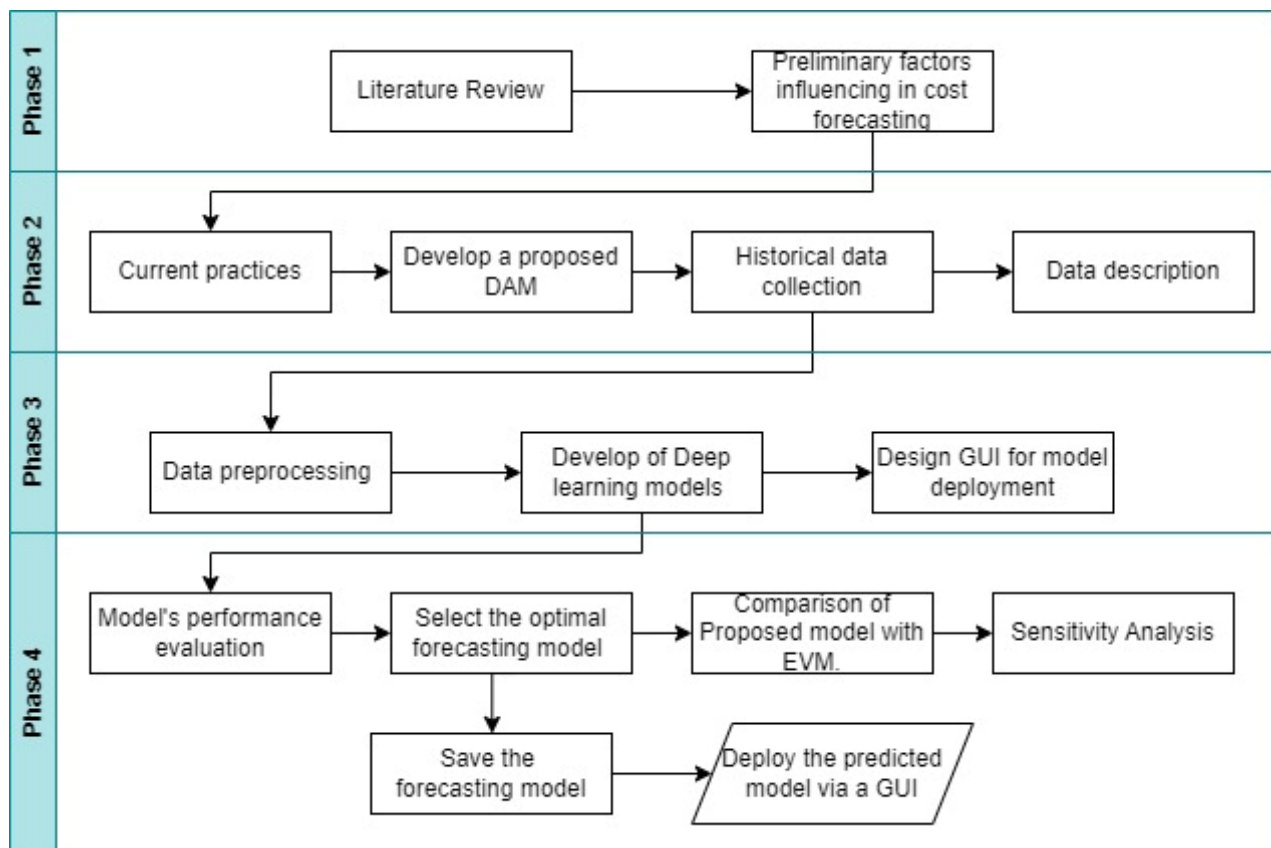


Figure 1.2 Research Methodology

1.5 Thesis Organization

This thesis comprises six chapters, beginning with an introduction. The subsequent chapters' contents are summarized as follows:

- Chapter 2: Literature Review—This chapter describes construction project cost management, including cost forecasting techniques. Additionally, it explains the fundamental principles of machine learning, algorithms, and prior academic research in machine learning in the context of predicting the cost estimate at completion.
- Chapter 3: Development of Conceptual Model for Forecasting Cost Estimate at Completion—This chapter presents the proposed research methodology, including a detailed explanation of the data collection process and the proposed data acquisition model.
- Chapter 4: Development of Computational Model for Forecasting EAC with Graphical User Interface – This chapter covers the development of forecasting models utilizing deep learning algorithms. It also includes the development of the graphical user interface for entering inputs, saving inputs, and deploying the proposed model.
- Chapter 5: Application and Verification – This chapter implements the developed model in a case study, compares the proposed model against traditional approaches, and verifies the model through a sensitivity analysis.
- Chapter 6: Conclusion—This chapter includes a summary of the work completed under this thesis, the research conclusion, its limitations, and recommendations for future research.

Chapter 2: Literature Review

2.1 Introduction

This research's main objective is to develop a predictive model for accurately forecasting the cost estimate at completion. This model is built based on a deep learning algorithm to predict the final cost of ongoing construction projects. Given the research problem's complex nature, this Chapter covers three areas. To develop the model effectively, it is imperative to analyze critical areas comprehensively: 1) Project cost management in construction; 2) Cost forecasting techniques, such as earned value management and other methods proposed in the literature; 3) Fundamental principles of machine learning, deep learning, algorithms, and prior academic research within the context of machine learning to predict the cost estimate at completion.

2.2 Project Cost Management in Construction

Construction project cost management involves estimating, budgeting, monitoring, controlling, and managing the day-to-day costs of a project (Project Management Institute, 2016). Cost estimating entails the development of an approximation of the monetary resources required for the successful completion of the project. Cost budgeting aggregates the estimated costs of activities or construction work packages and is established once an estimate is approved. The main benefit of this process is determining the cost baseline, which can later be used to monitor and control project performance. Monitoring and controlling project costs entails updating and tracking costs, managing any changes made to the cost baseline, and providing a forecast for all remaining costs (Project Management Institute, 2016). Forecasts and estimates are both critical for planning future project activities. Estimates are used to establish the Budget at Completion (BAC) for performance baselines.

On the other hand, forecasts predict the cost to complete, or Estimate to Complete (ETC), for works that are in progress or have not started (Amos, 2012). Accurate cost forecasting becomes challenging when factoring in unexpected events such as material cost escalation, material delays, scope variation, poor productivity, alterations to the project execution strategy, and poor subcontractor performance. Most current systems like EVM do not incorporate user judgmental inputs; judgmental inputs they rely on quantitative variables (Nassar, Nadim Kamil, 2004). The application of this system could lead to inaccurate forecast results.

Predicting the final cost of ongoing projects holds significance within the construction industry's project control procedures. The profitability of construction projects depends significantly on the construction organization's ability to predict the cost at completion well in advance and take the necessary corrective actions if some cost overrun occurs. One way to effectively enhance the accuracy of cost forecasts is by creating more dependable prediction tools that can integrate the inherent variability in construction projects into their calculations. Most project managers rely on index-based methods based on experts' judgment and linear trending approaches.

This Chapter will examine various forecasting methods and evaluate their advantages and disadvantages.

2.3 Cost Forecasting Techniques

Cost forecasting is a crucial aspect of construction projects, which has led to numerous research efforts in developing methods to predict the final cost at project completion. These methods focus on calculating the cost estimate at completion (EAC), which is considered the most reliable estimate of the total cost at the end of the project. Two main categories of cost forecasting methods are developed: 1. Methods based on earned value management and 2. regression-based approach.

2.3.1 Earned Value Method

For years, EVM has been the most widely used technique to manage a project's time and cost performance and predict the final project duration and cost. EVM has been a foundational project management tool in architecture, engineering, and construction since its inception in the late 1970s (He et al., 2017). Currently, EVM is recognized as a method that integrates project cost, schedule, and scope to evaluate actual status versus its baseline and to estimate the cost and duration at completion (Project Management Institute, 2019).

The EVM method has a crucial cost metric called EAC. EAC represents the estimates of the project's total cost and results from the summation of the actual costs incurred for work completed and the cost estimate to complete, as illustrated in Figure 2.1.

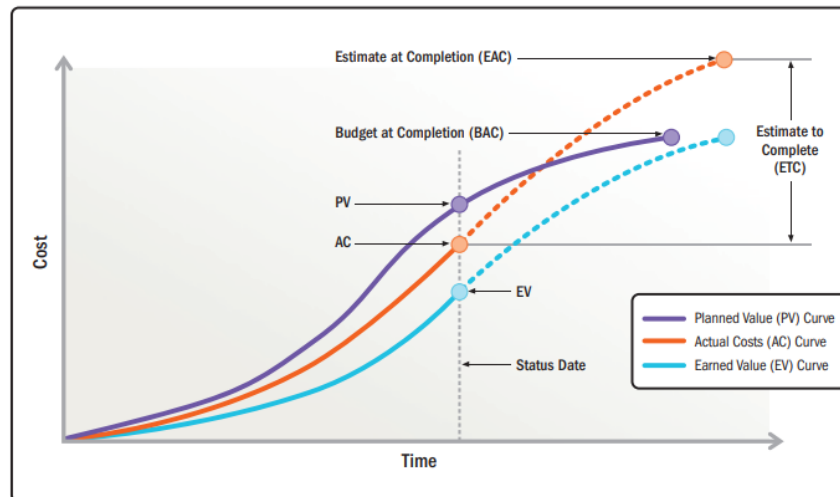


Figure 2.1 EVM components (Project Management Institute, 2019)

The following are important terms used in cost project management. Table 2.1 describes the principal EVM terminology.

Table 2.1 EVM Terminology

Abbreviation	Name	Definition
PV	Planned Value	The budget allocated and approved for the scheduled work.
EV	Earned Value	The amount of the work performed in terms of the authorized budget.
AC	Actual Cost	The realized cost incurred for the work performed.
BAC	Budget at Completion	The sum of all budgets created for the work to be developed
ETC	Estimate to Complete	The estimated cost to conclude all the remaining project work.
EAC	Estimate at Completion	The expected cost of completing all work is expressed as the sum of the AC to date and the ETC.

The EVM tool provides several extensions for EAC calculation. These extensions are represented as index-based formulas, which use cost and schedule performance index to calculate the EAC. Following the prescribed guidelines established by the Association for the Advancement of Cost Engineering (AACE), the equations for computing the EAC and their associated drawbacks under specific scenarios are as follows (Amos, 2012).

- a) This equation assumes that the remaining work will be accomplished precisely at the total approved budget value without observing the past performance index. It is valid when the cost performance index (CPI) is close to 1.0. Otherwise, this method could yield an unreal EAC calculation.

$$EAC = AC + \frac{(BAC-EV)}{1} \quad \text{Equation (2.1)}$$

- b) This equation assumes that the cumulative cost performance index (CPI_{cum}) efficiency remains constant throughout the project. Consequently, any incremental change in the CPI value over time will have less impact on the cumulative CPI. Notably, the formulation does not incorporate a scheduled performance index. This limitation can result in potentially inaccurate assessments when addressing schedule delays. Usually, two distinct alternatives may arise in a project schedule delay: allocating increased resources and effort to recover the delay or providing additional resources for an extended duration. Either of these alternatives will entail extra costs.

$$EAC = AC + \frac{(BAC-EV)}{CPI} \quad \text{Equation (2.2)}$$

- c) This method presupposes that the forecasted final cost is a function of both the cost and schedule index. This equation recommends that the project manager evaluate the schedule's critical path to identify the nature of the SPI. Some projects may still have an $SPI > 1.0$ when noncritical work is completed more than planned and activities on the critical path have delays. This equation is not recommended after 80% of the work effort is accomplished because the SPI value will be moved to 1.0 when the project is completed.

$$EAC = AC + \frac{BAC-EV}{CPI*SPI} \quad \text{Equation (2.3)}$$

- d) This approach is also influenced by the combination of six months' CPI (CPI_{6mth}) and the cumulative schedule index SPI_{cum} . Because the CPI index has a shorter duration and only a six-month average, this equation works best once the project gets into the steep incline on the S-curve rather than at the start of the project.

$$EAC = AC + \frac{BAC - EV}{CPI_{6mth} * SPI_{cum}} \quad \text{Equation (2.4)}$$

- e) Unlike the previous equations, this method assigns a weighted value to the CPI and SPI. The sum of this weighted value must equal 1.0. Weighted values require expert opinion from the project team. The weighting is usually heavier for the cost performance index because it has been demonstrated to be a more reliable indicator of final cost.

$$EAC = AC + \frac{BAC - EV}{(A * CPI_{cum} + B * SPI_{cum})} \quad \text{Equation (2.5)}$$

Where:

- AC: Actual Cost.
- EV: Earned Value.
- BAC: Budget at Completion.
- CPI: Cost Performance Index, $CPI = EV/AC$.
- SPI: Schedule Performance Index, $SPI = EV/PV$.

The underlying assumptions and prediction equations of the five estimate-at-completion (EAC) methods are summarized in Table 2.2.

Table 2.2 EAC Equations Summary

Method	Assumption	EAC Equation
EAC ₁	The remaining work will be completed within the approved budget.	$AC + \frac{(BAC - EV)}{1}$
EAC ₂	Future performance will be the same as the cumulative CPI _{cum} .	$\frac{BAC}{CPI}$
EAC ₃	Both the CPI and SPI will influence future performance.	$AC + \frac{BAC - EV}{CPI * SPI}$
EAC ₄	Future performance will be equal to the average of the recent six months' CPI _{6mth} and the SPI _{cum} .	$AC + \frac{BAC - EV}{CPI_{6mth} * SPI_{cum}}$
EAC ₅	Future performance will be equal to a weighted average of the CPI _{cum} and the SPI _{cum}	$AC + \frac{BAC - EV}{(A * CPI_{cum} + B * SPI_{cum})}$

Project managers typically rely on EVM for EAC calculation. Nevertheless, traditional EVM techniques have limitations, such as employing past information and cost performance index to calculate the remaining budget (Barraza et al., 2004; Cheng et al., 2012; Howes, 2000; Kareem Kamoon & Budayan, 2019). The EVM method assumes that cost performance remains stable throughout the project; however, cost performance tends to change over time in most projects (He et al., 2017). A study by Due et al. illustrates that for most projects, the CPI falls erratically when the project is around 25% complete; accordingly, the cost estimation will also change dramatically when about 25% of the work is completed (Du et al., 2016). Hence, the assumption that past performance guarantees future performance may not hold.

In this regard, certain academics have developed alternative methodologies to enhance the effectiveness of earned value-based techniques. Specific approaches have been formulated among these alternatives with a foundation in regression models.

2.3.2 The Regression-Based Approach

Researchers have increasingly embraced regression-based approaches to address the shortcomings of the conventional earned value-based method. Regression analysis calculated the EAC by regressing a dependent and independent variable (Nystrom, 1995). Regression analysis includes several variations, including linear and nonlinear statistics between the inputs and outputs relationship. Some of these equations are shown below (Hammad, 2009):

- Linear regression	$Y = \beta_0 + \beta_1 X$	Equation (2.6)
---------------------	---------------------------	----------------

- Quadratic regression	$Y = \beta_0 + \beta_1 X + \beta_2 X^2$	Equation (2.7)
------------------------	---	----------------

- Quadratic regression	$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$	Equation (2.8)
------------------------	---	----------------

- Exponential	$Y = \beta_0 + \beta_1 \text{Exp}^{(x)}$	Equation (2.9)
---------------	--	----------------

Where:

$\beta_0, \beta_1, \beta_2$ and β_3 , are the coefficients of regression.

The following section will briefly review regression-based approaches to forecasting the cost estimate at completion.

Narbaev and De Marco (2014) introduced a Gompertz growth model (GGM) via nonlinear regression curve fitting that employs S curve fitting, offering a pragmatic approach for predicting cost EAC. The S-curve, commonly used in project management, serves as a graphical representation of cumulative work progress over time. Its characteristic shape represents a slower progression at the project's start and finish. It reflects a consistent pace while displaying a more rapid advancement in the middle phase, indicating a steeper trajectory.

The Gompertz growth model articulates phenomena within data exhibiting a growth trajectory. It is widely employed for curve fitting and predictions and is a part of the sigmoidal model family. The GGM growth rate pattern aligns well with typical project cycles where the initial stages of the project see slow growth, followed by a rapid phase and a slow completion phase, as shown in Figure 2.2 (Ead, 2020)

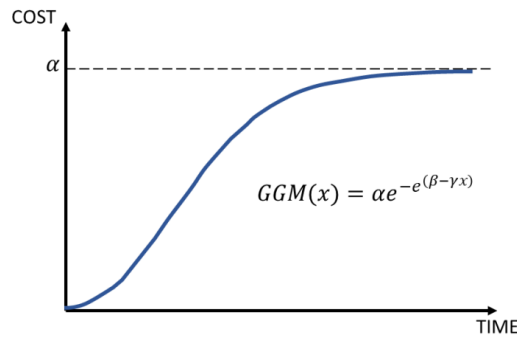


Figure 2.2 Gompertz Growth Model (Ead, 2020)

The Narbaev and De Marco's study is formulated based on three main targets:

1. To create a novel formula by modifying the index based (EAC) formula with four distinct growth models (Logistic, Gompertz, Bass, and Weibull).
2. To assess the effectiveness of the newly developed method through its application on nine past projects.
3. To determine the most effective growth model by considering statistical validity tests and comparing the accuracy of EAC estimations.

The Gompertz model-based cost EAC formula demonstrated superior fitting when comparing the four growth models based on their EAC error. It produced more accurate final cost forecasting than those obtained through the index-based method and the three other models (Narbaev & De Marco, 2014). In theory, the proposed method enhances EAC calculation by integrating EV metrics

with regression-based analysis, offering a practical approach that considers schedule impacts to improve cost forecasting accuracy (Araba et al., 2021).

Ottaviani and Marco (2022) unveiled a new approach. A multiple linear regression model performed this approach to forecast the cost of ongoing projects. The study was conducted on EVM data from twenty-nine projects and used inputs such as CPI, actual cost, work schedule, work performed, and periods. The model development process consists of three steps:

1. A generalized linear model selection procedure will be used to select the inputs included in the model.
2. Correlation analysis is used to select inputs to understand their relationship.
3. Multiple linear regression analysis to find the best-fit model by considering the correlations among the inputs and avoiding underfitting or overfitting issues.

The model's performance was then compared to the index based EAC method regarding variance and accuracy. The new EAC formulation's results show better accuracy and lower variance than the standard EAC calculation (Ottaviani & Marco, 2022).

2.3.3 Bayesian Statistics

Bayesian is a method of statistical inference in which Bayes' theorem is applied to deduce the parameters of a probability distribution and update these parameters based on new data available (Bartlett & Keogh, 2018). The general Bayes' theory can be written as follows (Kim, 2015).

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \quad \text{Equation (2.10)}$$

Where:

$P(\theta)$ is a prior distribution of a set of parameters θ

$P(D|\theta)$ is the conditional probability that a particular outcome D would be observed, given θ .

$P(D)$ is the marginal distribution of the outcome D .

$P(\theta|D)$ is the posterior distribution of θ given D .

The prior distribution is the most sensitive step because it includes expert opinions about project trends, future risks and opportunities, project patterns and impacts of corrective actions, corresponding to the main contribution given by experts (Caron et al., 2013)

Kim & Reinschmidt (2011), this study introduced a probabilistic cost forecasting method that combines the inside estimate (bottom-up), the outside forecast (top-down) and actual performance data during execution. The authors argue for an adaptive approach using Bayesian inference and Bayesian model averaging techniques, which incorporates actual performance data generated during project execution to update forecasts dynamically. Finally, the authors present qualitative examples of a hypothetical project to demonstrate the validity of the proposed method as a viable tool for effective project cost forecasting (Kim & Reinschmidt, 2011).

Caron et al. (2013) proposed a model to calculate the EAC based on integrating quantitative data and qualitative information in terms of past performance data records and expert knowledge. This study developed a Bayesian model within the EVM framework based on two main assumptions: independence of the indices CPI and SPI and a log-normal distribution. The proposed model includes three phases. First, data analysis and logarithmic transformation of the indices' values. Second, transforming experts' opinions into a prior distribution. Third, calculating the future

distribution of CPI_f and $SPI(t)_f$, where f represents the future value. The results highlight that the Bayesian model improved cost forecasting accuracy compared to traditional EVM methods.

Kim B (2015) presented a second-moment Bayesian model to enhance project cost forecasting by considering cost risk assessment and actual performance data. The model provides a probabilistic range of possible project costs at various confidence levels. The proposed method offers algebraic formulas for probabilistic forecasting without requiring extensive additional data collection or complex statistical analyses. The author illustrates the model's practical application and forecast accuracy through simulation experiments and numerical examples (Kim, 2015).

Caron et al. (2016) proposed a Bayesian model for forecasting the performance of large projects in the Oil & Gas industry. Their study explores different knowledge sources, such as experts' opinions and data from similar past projects and integrates them using the Bayesian approach. The study also highlights the application of the model to some real-life projects and concludes that the Bayesian model outperforms EVM, exhibiting significantly lower MEAN, approximately one-tenth (Caron et al., 2016).

Having explored the most common approaches presented in the literature, such as EVM, regression models and Bayesian statistics for calculating EAC, it is crucial to pivot towards contemporary advances that enhance predictive accuracy and efficiency. The following section will explore machine learning theory, which leverages computational algorithms to learn from data and make informed decisions. This shift not only represents the evolution of project forecasting methods but also sets the stage for a discussion on how machine learning techniques can be applied to optimize the calculation of EAC.

2.4 Machine Learning

The construction industry can use Machine Learning, a subfield of AI, to create automated technologies and make the construction process smarter (Y. Xu et al., 2021). Machine learning techniques involve developing algorithms and models for computer systems to learn from patterns in data without relying on explicit programming (Kareem Kamoona & Budayan, 2019; Rebala et al., 2019). In construction projects, machine learning techniques offer a data-driven approach to addressing complex challenges such as cost estimations. When correctly used, these techniques can significantly enhance the accuracy of cost predictions and decision-making by leveraging historical data, patterns, and relationships within construction processes.

ML algorithms can analyze large-scale historical data from construction projects to identify patterns and make more accurate cost predictions (Y. Xu et al., 2021). ML models can consider quantitative factors such as cost, physical progress, and duration and qualitative factors such as weather conditions and project complexity, among other factors (Feylizadeh et al., 2012). ML also offers the potential for real-time cost monitoring and adjustment, allowing for more effective project control and decision-making (Kareem Kamoona & Budayan, 2019). Despite the promising results of machine learning in construction, challenges remain: 1) the need for a significant amount of data and 2) some existing machine learning algorithms are not high enough for practical applications. (Xu et al., 2021). The following section provides an introductory framework for understanding the fundamental concepts, roles, and potential challenges of machine learning in cost forecasting.

2.4.1 Conceptual Machine Learning Terms

To facilitate understanding of the concept of ML, the following are some critical conceptual terms.

- **Dataset:** A dataset is a collection of data that follows a specific structure or schema. In a typical dataset, each column represents a feature or attribute, while each row represents an individual member or example of the dataset (Awad & Khanna, 2015). In classical statistics, independent variables are inputs, and dependent variables are outputs (Hastie et al., 2009). The output is the class, target value, or label.
- **Model:** The concept of a model is the core representation of the problem for trying to solve. Each model can be adjusted to the specific requirements of an application. They are building a suitable model, whether a linear regression, decision tree, or deep neural network, significantly impacts the machine-learning system accuracy.
- **Algorithms:** Algorithms drive how models learn from data. Understanding different algorithms and their suitability for tasks is essential.
- **Training, validation, and testing split:** In many practical applications, a dataset is split into three subsets: training, validation, and testing (Shalev-Shwartz & Ben-David, 2014). The first subset is used to train the algorithm, and the second subset is reserved for selecting the best model. Once the optimal model is selected, the test subset is employed to evaluate the performance of the resulting predictor.

2.4.2 Types of Machine Learning Algorithms

Machine Learning is a vast domain. Consequently, the field of ML has been divided into several types dealing with different learning tasks. Three main types of machine learning algorithms depend on the problem to be solved: supervised, unsupervised, and reinforcement learning (Prince, 2023). Figure 2.3 shows a taxonomy of machine learning with exemplary applications listed under each type.

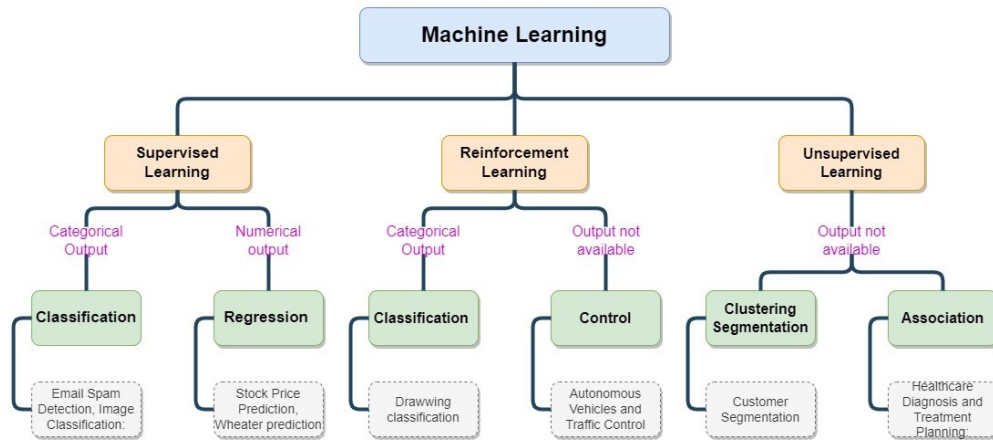


Figure 2.3 Machine Learning Algorithm Types

a) Supervised Learning:

This form of learning is a prevalent type of machine learning where the algorithm is provided with a large dataset of data points labelled with actual answers (Rebala et al., 2019). The algorithm analyzes the key characteristics of each data point in the dataset to identify patterns and make predictions. When presented with a new data point, the algorithm uses these patterns to predict the outcome accurately. Supervised learning is divided into two subcategories: Regression and classification problems. Regression is used to predict numerical values, while classification is used to classify one or more labels.

b) Unsupervised Learning:

The algorithms present data with no explicit labels or predefined categories in unsupervised learning. In other words, the algorithm does not have a specific target to achieve. Instead, the algorithms focus on discovering patterns, structures, or relationships within the data. Unsupervised learning techniques include:

- **Clustering:** In clustering, also called data segmentation, the algorithms group data points based on similarity into clusters. Standard methods include K-Means clustering and hierarchical clustering. Clustering is used for tasks like customer segmentation and anomaly detection.
- **Association Rule Learning:** This technique focuses on discovering exciting relationships or associations between variables in a dataset.

Unsupervised learning is not widely used in construction due to limitations in extracting information from unlabelled data compared to labelled data (Y. Xu et al., 2021).

c) Reinforcement learning:

Although not directly relevant to the techniques employed in this research, reinforcement learning (RL) is briefly outlined here to ensure a comprehensive coverage of machine learning types. RL is a type of ML where a system interacts with an environment and learns to make sequential decisions (Mohsen, 2021). The system's main objective (agent) is to maximize the cumulative reward over time by taking actions that result in favourable outcomes. It is achieved through a trial-and-error process where the agent learns from the consequences of its activities and receives feedback in the form of rewards or penalties. RL is used in various fields such as robotics, game playing, autonomous vehicles, and recommendation systems where decision-making over time is critical.

This chapter developed a basic understanding of the core machine-learning concepts discussed in the previous sections. In the subsequent section, a concise theoretical overview of the most widely applied algorithms within the realm of cost forecasting in construction projects will be presented.

2.4.3 Machine Learning Algorithms Used for Cost Forecasting

In regression learning, the objective is to establish a connection between a set of input variables (predictors) and a continuous output variable (target) to predict outcomes for new input variables. The literature encompasses a range of regression algorithms employed to estimate final cost forecasting surrounding basic approaches like linear regression (LR), support vector machines (SVMs), artificial neural networks (ANNs), as well as more sophisticated techniques such as deep neural networks (DNNs) and long short-term memory (LSTM). The subsequent section briefly overviews these methods to understand their application in the context of construction management.

a) Artificial Neural Network (ANN)

An artificial neural network (ANN) is an inspired algorithm which tries to mimic the behaviour of the human brain's neural network (Tyagi & Abraham, 2022, p. 3). This mathematical model is based on connections between units or nodes known as artificial neurons (Mirtaheri & Shahbazian, 2022). Each connection from one artificial neuron to another establishes the flow of information. Artificial neurons and connections have a weight that adjusts as the system learns. The strength of the information in each connection can be intensified or diminished due to the weight (Tyagi & Abraham, 2022). An ANN is typically organized in layers, as shown in Figure 2.4.

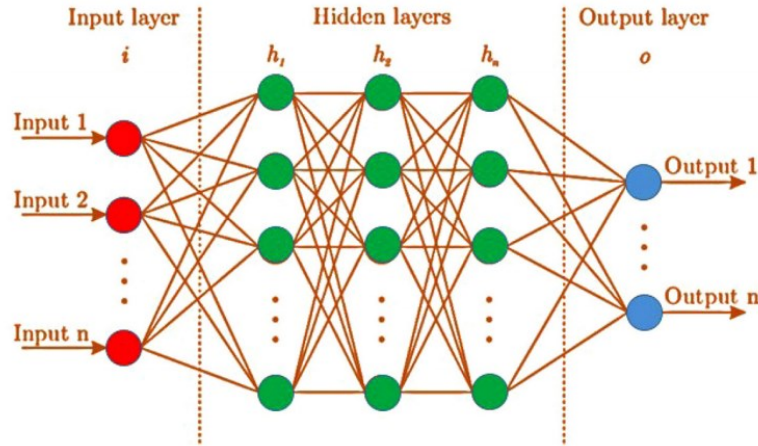


Figure 2.4 ANN architecture (Tyagi & Abraham, 2022)

Since the early 1990s, ANN has been investigated as an innovative management tool in construction management (Moselhi et al., 1991). Some applications of ANN in construction management include predicting labour productivity (Heravi & Eslamdoost, 2015), safe work behaviour (Patel & Jha, 2015), estimating schedule to completion (Cheng et al., 2019), and cost estimates at completion (Pewdum et al., 2009) in construction projects.

b) Support Vector Machine (SVM)

SVM is a supervised learning algorithm for classification and regression problems, which is founded on statistical learning theory developed by Vapnik (Vapnik, 2000). SVM regression aims to find a hyperplane that accurately fits data points, with the margin around the hyperplane measuring the model's accuracy (Mirtaheiri, 2022). SVM algorithm is beneficial for complex or nonlinear data relationships because it can use kernel functions, such as polynomial, radial basis function or sigmoid, to map the data into a higher-dimensional space and to capture intricate patterns. SVM is also robust against outliers and can control the trade-off between

model complexity and error, making it a valuable tool in various regression applications in construction, including control project forecasting (Wauters & Vanhoucke, 2014).

Despite the previous advantages mentioned for this algorithm, SVM for regression analysis has a significant drawback in its sensitivity to hyperparameters and the kernel function (Mirtaheiri, 2022). Selecting the right values for these hyperparameters can be challenging and time-consuming. If not chosen correctly, they can lead to suboptimal model performance, overfitting or underfitting. Moreover, SVM regression can be computationally expensive, particularly with large datasets, which may limit its usefulness in settings with vast amounts of data.

2.5 Emergence of Deep Learning

Deep learning is a subset of machine learning that can analyze time series data (Prince, 2023). It performs better in handling big data and complex nonlinear relationships, as evidenced by recent applications such as image classification (Jiang & Zhang, 2020), object detection (Zhao et al., 2019), and time series prediction (Brownlee, 2018; Jiang & Zhang, 2019). Deep Learning embraces several neural networks, including recurrent neural networks (RNNs). RNNs are specially built to handle time series data and can maintain the memory of previous inputs through a hidden state (Tyagi & Abraham, 2022). There are three main types of RNNs, which are categorized based on their architectural features: Simple Recurrent Neural Network (Simple RNN), Long short-term Memory (LSTM) and Gated Recurrent Unit (GRU).

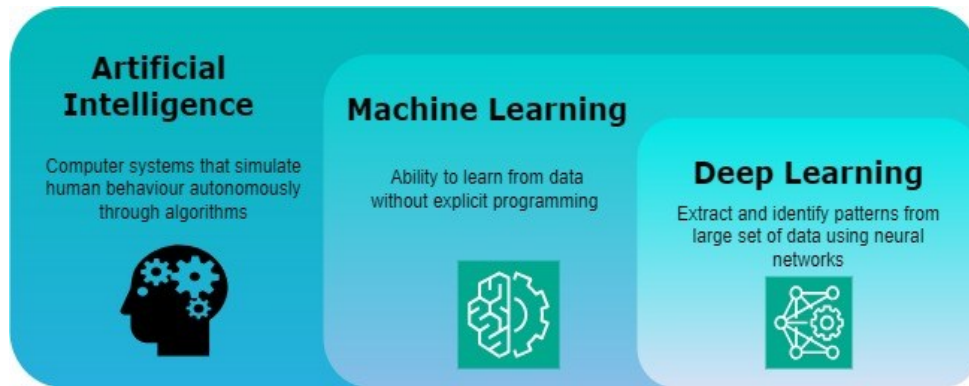


Figure 2.5 AI, Machine Learning, and Deep Learning Overview

a) Simple Recurrent Neural Network

Simple Recurrent neural networks (Simple RNN) effectively model time series data (LeCun et al., 2015). Simple RNNs make decisions influenced by previous decisions at each time step, meaning that the outcome at the time step 't-1' impacts the decision-making process at the current time step, 't' (Desell et al., 2020). However, Simple RNNs can struggle with long-term dependencies, leading to gradients that either explode or vanish (Hossen et al., 2018; Tian et al., 2018). To address this issue, a specialized type of RNN architecture known as LSTM was introduced to improve the performance of the simple RNNs.

b) Long short-term Memory (LSTM)

LSTM is a particular type of RNN; LSTM was introduced by Hochreiter and Schmidhuber (Hochreiter & Schmidhuber, 1997) and was refined and popularized by Gers, Schmidhuber, and Cummins (2000) and Graves and Schmidhuber (2005). LSTM networks are architecture designed to learn long-term dependencies and can avoid vanishing gradients using memory cells (J. Xu et al., 2024). The LSTM architecture replaces the typical hidden

layers with LSTM cells. These cells consist of multiple gates that regulate the input flow. These multiple gates comprise an input gate, forget gate, and output gate. Figure 2.6 illustrates the architecture of Long Short-Term Memory (LSTM) cells.

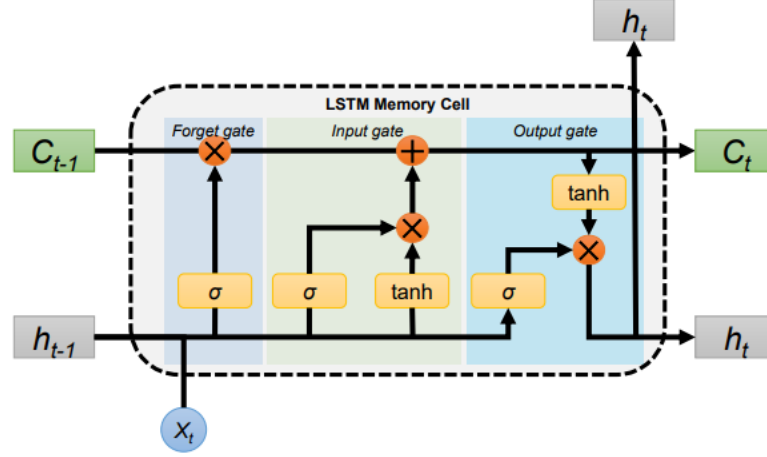


Figure 2.6 LSTM cell (Fan et al., 2020)

The key to LSTM is the cell state (C_t), which enables the information to flow unchanged. Three gates regulate the cell state to let information flow optionally. The first gate, called the *forget gate layer*, decides the elements of the cell state vector (C_{t-1}) will be ignored.

$$f_t = \sigma(W_f h_{t-1} + W_f h_t) \quad \text{Equation (2.11)}$$

Afterwards, the *input gate* determines which value needs to be updated.

$$i_t = \sigma(W_i h_{t-1} + W_i h_t) \quad \text{Equation (2.12)}$$

Following the *intermediate cell state* is calculated by the current input x_t and the last hidden state h_{t-1} :

$$\tilde{C} = \tanh(W_c h_{t-1} + W_c h_t) \quad \text{Equation (2.13)}$$

After that, the old **cell state** C_{t-1} can update into the new cell state C_t by element-wise multiplication:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad \text{Equation (2.14)}$$

Finally, the **output gate** decides which to be output by a sigmoid layer:

$$o_t = \sigma (W_o h_{t-1} + W_o h_t) \quad \text{Equation (2.15)}$$

The new hidden state h_t is then computed by employing equations 2.12 and 2.13.

$$h_t = O_t * \tanh(C_t) \quad \text{Equation (2.16)}$$

Where:

x_t : Input vector at time t, h_t : output vector at time t, C_t : cell state vector, f_t : forget gate vector, i_t : input gate vector, O_t : output gate vector.

c) **Gated Recurrent Unit (GRU)**

Gated recurrent Unit (GRU) can be considered a variation of LSTM that aims to overcome the vanishing gradient limitation in Simple RNN. GRU proposed by Cho et al (Cho et al., 2014) enables each unit within the network to adjust and capture dependencies dynamically across varying time scales (Chung et al., 2014). This algorithm utilizes two gates, one less than LSTM, so it is slightly simpler architecture than LSTM (Tyagi & Abraham, 2022).

- Reset gate (r) establishes how much past information can be forgotten (Yamak et al., 2019).
- Update gate (Z) decides how much of the past information (previous hidden state) needs to be passed along to the next step.

$$r = \sigma(W_r h_{t-1} + U_r x_t) \quad \text{Equation (4.15)}$$

$$Z = \sigma(W_z h_{t-1} + U_z x_t) \quad \text{Equation (4.16)}$$

$$c = \tanh(W_c(h_{t-1} * r) + U_c x_t) \quad \text{Equation (4.17)}$$

$$h_t = (1 - Z) * h_{t-1} + z * c \quad \text{Equation (4.18)}$$

Where σ denotes the sigmoid function, W 's are the parameters matrix, x_t is the input at time t , h_{t-1} represent the hidden state from the previous time step. c the candidate's hidden state determines what to collect from the current memory content. h_t is the final hidden state for the current time state.

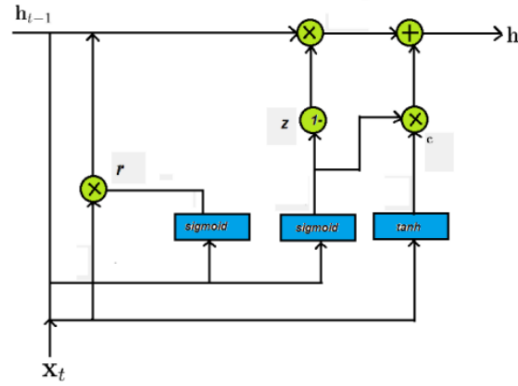


Figure 2.7 GRU Network (Yamak et al., 2019)

2.6 Assessing the Performance of Machine Learning Algorithms

To compare different machine learning models, a systematic approach evaluates their performance and ability to analyze new data effectively. Several commonly employed metrics exist to measure the accuracy of numerical predictions. There are some standard metrics used for evaluating machine learning regression algorithms, including Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Mean Percentage Error (MPE), Mean Square Error (MSE) and Root

Mean Square Error (RMSE) (Botchkarev, 2019). Table 2.3 displays the equations for these performance measures.

Table 2.3 Performance Measure Equations

Performance Measure	Expression
Mean Absolute Error (MAE)	$MAE = \frac{1}{n} \sum_{i=1}^n a_i - p_i $
Mean Absolute Percentage Error (MAPE)	$MAPE = \frac{1}{n} \sum_{i=1}^n \left \frac{(a_i - p_i)}{a_i} \right $
Mean Percentage Error (MPE)	$MPE = \frac{1}{n} \sum_{i=1}^n \frac{(a_i - p_i)}{a_i}$
Mean Square Error (MSE)	$MSE = \frac{1}{n} \sum_{i=1}^n (a_i - p_i)^2$
Root Mean Square Error (RMSE)	$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (a_i - p_i)^2}$

Where: a_i is the actual observed target value of example i , p_i is the predicted target value of example i , and n is the total number of examples in the dataset.

When selecting performance measures for a machine learning algorithm, it is crucial to consider the problem and its application carefully. The performance metric chosen should align with the model's goals. Some metrics, such as MAPE, are more relevant in specific business applications as they provide insights into the percentage accuracy. Choosing interpretable metrics easily is essential, depending on the users. Metrics such as MAE and MAPE are relatively easy to interpret, whereas others, such as RMSE, may need to be more intuitive to non-technical stakeholders.

2.7 Machine Learning Application for Cost Forecasting

In the context of Cost EAC, it is essential to highlight that limited studies have been conducted. This section provides an overview of the existing studies that have primarily focused on developing machine learning models, with a particular emphasis on their use for forecasting project costs within the project execution phase.

Pewdum et al. (2009) created an Artificial Neural Network (ANN) model to predict the cost and duration of highway construction projects during the execution phase in Thailand. The EAC was calculated based on eight factors: traffic volume, topography, weather conditions, evaluation date, contract duration, construction budget, as-planned completion percentage, and actual completion percentage. On the other hand, the duration was calculated using factors such as work starting date, evaluation date, contract duration, rate of as-planned completion, and percentage of actual completion. The study findings demonstrate a notable contrast in forecasting accuracy, as indicated by the Mean Absolute Percentage Error (MAPE) values. The MAPE for the ANN model was 2.6%, whereas the MAPE for the earned value method was significantly higher at 28.37%. This divergence underscores the effectiveness of the proposed ANN model in accurately predicting the EAC for highway construction projects in Thailand (Pewdum et al., 2009).

While the results showcased the predictive capabilities of the Artificial Neural Network (ANN) model, it is essential to acknowledge some limitations associated with the ANN algorithm. One disadvantage of using an Artificial Neural Network (ANN) is its inherent black-box nature, so it cannot explain the underlying processes (Kareem Kamoona & Budayan, 2019; Kudirat et al., 2023). ANN's black box modelling predicts output variables based on their relationships with the input variables (Vu & Do, 2021). Consequently, when employing ANN modelling, it becomes

essential to have a sufficiently large dataset to minimize the risk of overfitting in predictions (Aggarwal, 2018).

One of the primary disadvantages of ANNs for forecasting is their lack of transparency, often described as a 'black box' characteristic (Kudirat et al., 2023). The model has a complex architecture with multiple hidden layers and parameters, making it difficult to interpret how the predictions were made. Additionally, ANNs are not inherently designed to handle temporal dependencies of time series data (Fang et al., 2017).

Cheng et al. (2010) presented a Support Vector Machine Inference Model (ESIM) to estimate the final project cost during the construction execution. The proposed model combines two machine learning approaches: The fast messy genetic algorithm (fmGA) and the Support Vector Machine (SVM). The ESIM model considered seven factors: CPI, SPI, subcontractor billed index, owner billed index, change orders, climate, and market pricing effects. Data from two testing case studies were modelling and comparing the predicted values of EAC with actual values, and average errors were 7.02% and 1.68%, respectively (Cheng et al., 2010).

Cheng and Hoang (2014), in a subsequent study, proposed a model to predict the construction project cost through the combination of Least Squares Support Vector Machine (LS-SVM), machine learning-based interval estimation (MLIE), and differential evolution (DE). A difference from the previous proposal is that this model provides interval results with lower and upper prediction limits. The study analyzed data from 13 reinforced concrete building projects that were carried out in Taiwan. The researchers considered ten input variables: construction progress %, actual cost, planned cost, CPI, SPI, subcontractor billed index, owner billed index, change order index, construction price fluctuations, and climax effect index. The study result indicated that the proposed model achieved the following values in the testing data: MAPE=3.74, MAE = 0.03, and

RMSE = 0.04 (Cheng & Hoang, 2014). Additionally, the performance metrics of this model were compared with the ANN model, and the EAC-LSPIM performed the best result.

Feylizadeh et al. (2012) proposed a different approach, presenting a Fuzzy Neural Network (FNN) model to estimate the final project cost. The model included both qualitative and quantitative factors. For quantitative factors, the authors considered actual cost, budget at completion, earned value, planned value, cost performance index, and schedule performance index. The authors also considered qualitative inputs such as weather conditions, employer cash status, and the degree of experience of project staff. The study compared the model's results with the traditional EAC forecasting approach and found that the proposed model outperformed (Feylizadeh et al., 2012). However, the study did not provide the values of performance prediction indicators.

Dastgheib et al. (2022), this study proposed a model which employs an adaptive neuro-fuzzy inference system (ANFIS) to forecast the EAC of construction projects. This model included earned value, planned value, actual cost, contractor payments as inputs, and EAC as an output. The EV, AC and PV datasets were simulated considering the normal distribution. This study compared the ANFIS model with three types of neural networks: multi-layer perceptron, radial basis function, and generalized regression neural networks. The results confirmed that the ANFIS model performed well with RMSE = 0.0181 and MSE= 0.0003 (Dastgheib et al., 2022).

Despite the potential benefits of using Adaptive Neuro-Fuzzy Inference Systems (ANFIS) in construction projects, certain limitations must be considered. One of the biggest challenges of this study was obtaining comprehensive and appropriate project data from completed past projects to train the ANFIS model effectively. Therefore, the authors created synthetic data to evaluate the proposed model's performance.

Kamoona and Budayan (2019), this study proposed a novel model named deep neural networks (DNN) to calculate the EAC. The first phase of this study compares the DNN model with the support vector regression model (SVR). The results confirmed the predictability of the DNN over the SVR models. The second phase involves implementing a hybrid model that combines a genetic algorithm (GA) and brute force (BF) with a deep neural network to allocate correlated attributes and build a predictive model accurately. The authors' findings suggest that the GA-DNN model is a robust intelligence model for calculating EAC. The inputs incorporated in this model were cost variance, schedule variance and SPI. The performance metrics for the testing data were $RMSE = 0.0566$, $R^2 = 0.91$, and $MAE = 0.4446$, indicating that the model's predictions were accurate and reliable.

İnan et al. (2022) developed a predictive model based on a deep learning algorithm (LSTM) to forecast actual costs at completion. While the authors mention using a seven-dimensional feature vector, including CPI and SPI, details about including other input variables are lacking. Furthermore, the study's analysis compares the outcomes derived from the LSTM model against those stemming from an index-based method. The findings showed that, in 75.33% of the examined projects, the Mean Absolute Percentage Error (MAPE) metric generated by the LSTM model exhibits superior performance compared to the index-based method (İnan et al., 2022). The study proposed by İnan et al. makes significant progress in applying deep learning models to cost forecasting for Estimate at Completion (EAC). While their approach effectively forecasts EAC at the project level using variables derived from traditional Earned Value Management (EVM) inputs, it also highlights opportunities for further enhancement. Specifically, there is potential for more granular analysis and the inclusion of new input variables to improve EAC forecasting. Table 2.4 provides a summary of these studies with a brief description.

Table 2.4 Summary of Machine Learning Approach

Modelling Technique	Inputs	Description	Reference
ANN	Traffic volume, topography, weather conditions, evaluation date, contract duration, construction budget, planned completion %, and actual completion %.	An ANN model is used to forecast the duration and EAC of highway projects in Thailand. The model employs data from fifty-one highway construction projects.	Pewdum et al. (2009)
SVM and fmGA	Duration, AC, PV, CPI, SPI, subcontractor management contract payment, change order, contract payment, construction price fluctuation and weather conditions.	A hybrid model for predicting EAC. The model employed data from thirteen building projects executed in Taiwan.	Cheng et al. (2010)
LS-SVM, MLIE, and DE	Duration, AC, PV, CPI, SPI, subcontractor management contract payment, change order, contract payment, construction price fluctuation and weather conditions.	The study provides a hybrid model for predicting EAC and interval results with lower and upper prediction limits. The model employed data from thirteen building projects executed in Taiwan.	Cheng and Hoang (2014)
FNN	AC, BAC, EV, PV CPI, SPI, employer cash status, weather conditions, and the experience of project staff.	The FNN model considers qualitative and quantitative factors. This study needs to mention the origin of the data employed.	Feylizadeh et al. (2012)
GA-DNN	CV, SV, CPI	A deep neural network model for predicting EAC. This model employed fifteen construction projects executed in Iraq.	Kamoon and Budayan (2019)
ANFIS	EV, PV, AC, contractor payments	The ANFIS model forecasts the EAC, considering the risk for qualitative variables. The inputs were randomly generated using a normal distribution.	Dastgheib et al. (2022)
LSTM	Seven-dimensional feature vector, including CPI and SPI	The LSTM model for predicting EAC, the study used data from forty-one projects.	İnan et al. (2022)

2.8 Summary and Research Gaps

Several previous research studies attempted to forecast the EAC of construction projects by considering diverse cost factors and using various modelling techniques. The EVM technique has been the industry's prevailing cost prediction technique for decades. Despite its popularity, several findings suggest that the traditional EVM technique can be misleading due to their assumption that cost performance remains consistently stable and uniform throughout the project (Barraza et al., 2004; Du et al., 2016; Howes, 2000; Narbaev & De Marco, 2014). Moreover, EAC formulas used in the EVM method cannot consider qualitative factors such as project types, weather conditions, and contractor payments (Dastgheib et al., 2022). These limitations can affect the EAC accuracy calculated by EVM methodology.

A regression-based approach has been proposed to address the challenge posed by the EVM. For instance, Narbaev and De Marco (2014) proposed a method based on an earned schedule factor modified by GGM via a nonlinear regression analysis (Narbaev & De Marco, 2014). Later, Caron et al. (2016) developed a Bayesian approach capable of including experts' opinions, data from past projects and the current performance (Caron et al., 2016). In 2022, Ottaviani and De Marco presented a new EAC formulation based on multiple linear regression analysis (Ottaviani & Marco, 2022). While there have been improvements in EAC computation using the regression-model approach, adopting advanced modern artificial intelligence models is crucial to advance cost forecasting in system development. In the quest for more comprehensive cost forecasting solutions, the integration of machine learning emerges as the next logical step, building upon the insights gained from the EVM method and the regression-based approach.

Leveraging the capabilities of ML promises to address the remaining challenges in a dynamic and data-driven manner. Although there have been several studies since 2009 on EAC prediction using

machine learning models, the studies developed still need to be improved and require more attention, especially in applying deep learning algorithms to solve time series forecasting problems. For instance, two studies only tested deep learning algorithms in cost forecasting of ongoing projects. This limitation is notable, considering the potential of deep learning models such as Simple RNN, GRU and LSTM networks, which can capture intricate patterns in time series data. The underutilization of these techniques hinders the development of more accurate cost forecasts in the construction industry.

The current machine learning models exhibit a common limitation: their focus on project-level modelling. While these models offer valuable insights into overall project cost forecasting, they lack granularity at the work package level. This limitation hinders identifying potential deviations and cost overruns at the lowest level of the work breakdown structure. Work package-level modelling becomes essential for identifying specific areas of concern that a project-level approach could mask.

Considering the described limitations in the cost forecasting for ongoing projects, the proposed work follows: 1) Selecting relevant inputs impacting cost forecasting in construction projects; 2) Choosing algorithms capable of capturing nuanced patterns within time series data like deep learning techniques, and 3) Building a deep learning model able to enhance the accuracy of cost forecasting at the work package level

Chapter 3: Development of Conceptual Model for Cost Forecasting

3.1 Introduction

This study aims to create a cost forecasting model to help project managers calculate EAC at the work package level. To create the forecasting model, the input factors, the process for developing the forecasting model, and the output target were defined, as illustrated in Figure 3.1. By focusing on construction projects in the execution phase and introducing an innovative method for real-time forecasting, this study proposes a comprehensive framework for accurately predicting EAC. The input stage encompasses critical factors necessary for the model, identified by examining industry practices and literature review. The primary process consists of four essential steps: developing a data acquisition model, data preprocessing, model building and configuration, and evaluating and selecting the optimal forecasting model. The output target represents the value calculated by the forecasting model, which subsequently facilitates the determination of the EAC.

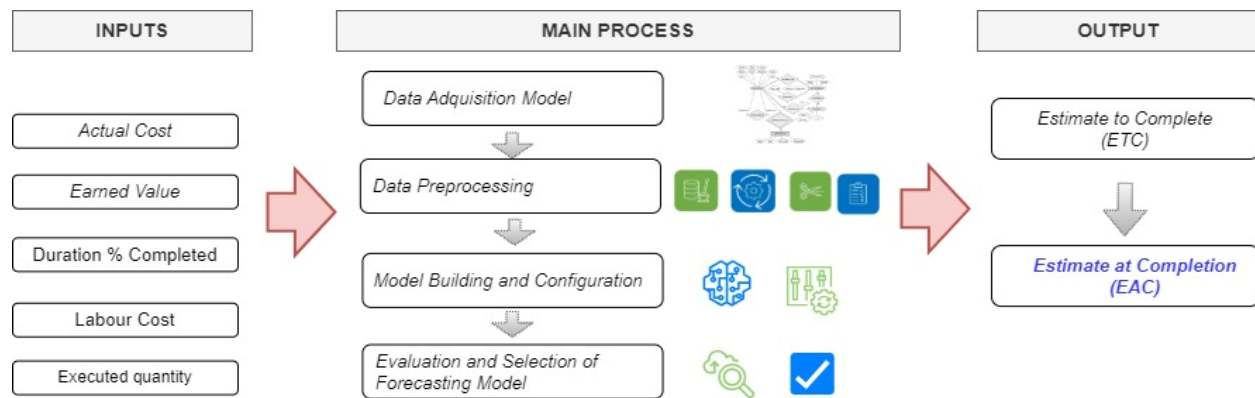


Figure 3.1 Conceptual Model for Cost Forecasting

Defining the input factors involves examining industry practices and conducting a literature review. It comprises analyzing the current practices employed by construction companies in cost forecasting calculation and reviewing academic contributions that propose new methodologies for cost forecasting. This comprehensive review also encompassed the factors integrated within these

methodologies, ensuring a thorough understanding of the existing and emerging approaches in the field.

This study concentrates on construction projects in the execution phase from the contractor's perspective. It suggests an innovative method for real-time forecasting of direct costs. The research outlines the primary process for developing a data acquisition system and model development with a graphical user interface (GUI). This process is based on a framework comprising five main steps: data acquisition model, data preprocessing, model building and configuration, evaluation and selection of the optimal forecasting model, and GUI development, as shown in Figure 3.2.

This chapter centres on several key aspects.

1. Investigating forecasting methods employed by industry practitioners.
2. Identifying the project factors significantly impacting cost EAC prediction, drawn from prior research and discussions with experienced project managers.
3. Establishing a data acquisition model to streamline data collection for project managers.
4. Gathering historical project data for use in developing forecasting models.
5. Describing the preliminary dataset's factors

The following chapter outlines the detailed processes for model development with a GUI.

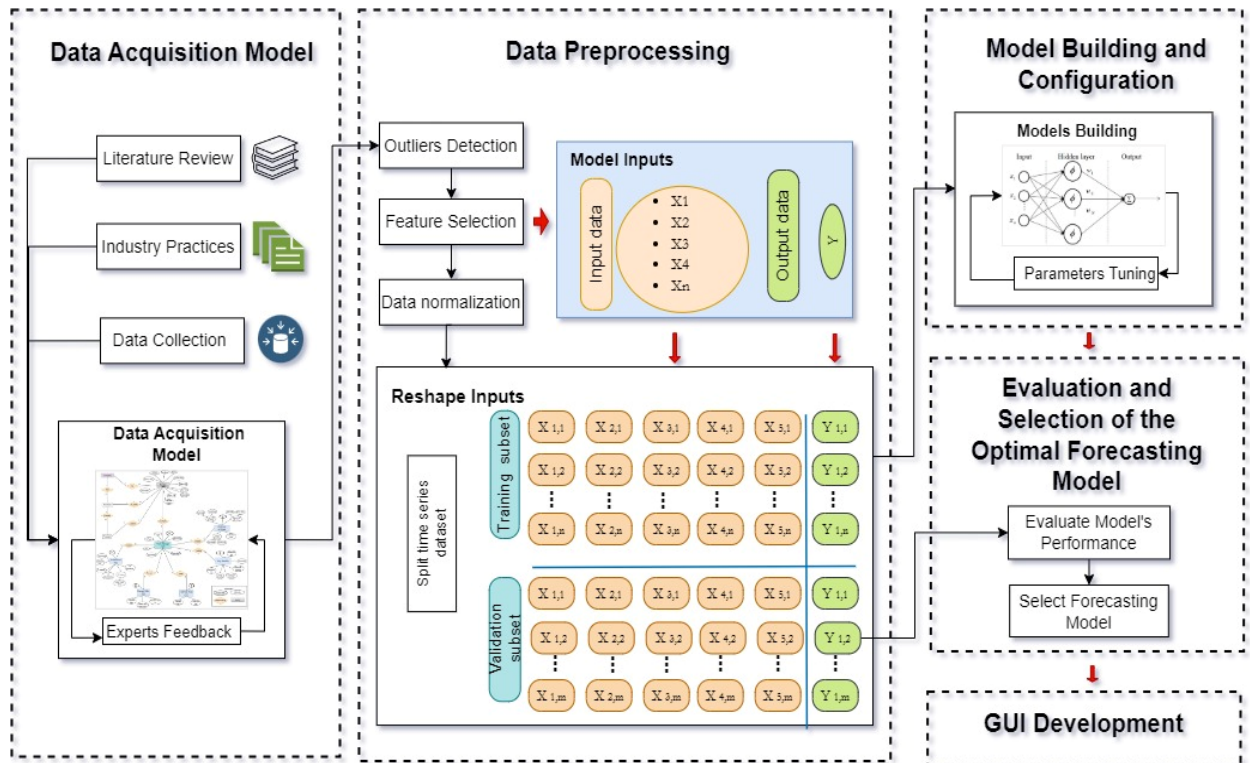


Figure 3.2 DAM and Forecasting Model Development Methodology

3.2 Current Cost Forecasting Practices

From the contractor's perspective, EAC forecasting in construction projects involves several approaches and methods. Typically, project managers choose between two strategies to calculate EAC: Bottom-up and EVM methodology.

The bottom-up technique for cost forecasting during construction execution is a detailed and systematic approach aggregating the estimated costs of elements or work packages to determine the remaining project cost. One of the primary advantages of the bottom-up approach is its accuracy; focusing on the specifics of each task offers reasonable forecasting of total remaining project costs, making it particularly useful for complex projects where each component's cost can significantly impact the overall budget.

However, the bottom-up technique is not without its disadvantages. Typically, the most significant drawback is the time-consuming nature of this technique, as it requires a detailed understanding of the project and much effort to break down and estimate every aspect of the remaining work. This level of detail can also lead to analysis paralysis, where an overabundance of information hinders decision-making. As a result, this method is laborious for periodically or monthly developing forecasts. Additionally, the accuracy of a bottom-up approach is dependent on the quality of the collected data, including actual rate cost, current productivity, remaining quantities, change orders, and price material escalations; inaccuracies in forecasting individual activities or work packages can compound, leading to significant inconsistencies in the overall project cost forecast.

Earned Value Management is one of the industry's most widely known techniques for cost prediction (He et al., 2017). During the project execution, the EVM method measures the project's performance periodically, forecasts the EAC and analyzes variances in schedule and budget (Fleming, 2016). When it comes to predicting costs, EVM uses the cost performance index (CPI) to determine cumulative cost performance by dividing EV by AC, and then this index is used to forecast EAC (Lipke et al., 2009). Several EVM-based cost forecasting formulas are available, as discussed in Chapter 2 of this study. Each formula has its own set of assumptions and is used at the discretion of project managers, depending on the specific project requirements. Although EAC formulas from the EVM method are popular, they do not account for certain factors that impact project cost performance. These factors include unexpected weather conditions, price fluctuation, exchange rate (Baloi & Price, 2003), and variability of labour productivity (Lema & Price, 1996).

3.3 Key Factors Influencing EAC

Identifying critical factors influencing the calculation of cost EAC is based on a thorough review of existing literature and expert opinion from practitioners.

As mentioned previously, there are limited investigations about prediction cost forecasting of ongoing projects employing artificial intelligence techniques. Nevertheless, it is worth noting that many studies have primarily concentrated on developing models for estimating project costs during the planning stage. These models consider factors including project type, total floor area, soil condition, presence of underground floors, seismic zone and technical complexity (Cheng & Roy, 2010). During the early stages of construction projects, construction companies emphasize budget planning while overlooking other aspects, such as cost fluctuations, information updates, and comprehensive cost management. In contrast, the main goal of this study is to create a forecasting model framework that can forecast the cost estimate at completion, capturing the data generated during the execution stage. To achieve this goal, this study comprehensively reviews pertinent literature on the key inputs to be incorporated into an EAC forecasting model.

Researchers have identified several factors that influence cost forecasting models. Table 3.1 presents the factors used in the past literature, including SPI, CPI, Cost Variance (CV), Schedule Variance (SV), subcontractor billing, contract payment, change orders index, construction cost index, AC, EV, PV, weather conditions, percentage work planned and performed, and project staff experience.

Table 3.1 Factors Influencing Project Cost Forecasting

Factor number	Influence Factors	Literature												
		1	2	3	4	5	6	7	8	9	10	11	12	13
1	SPI	✓	✓	✓		✓			✓	✓	✓	✓	✓	
2	CPI	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	
3	CV		✓											
4	SV		✓											
5	Subcontractor billed index		✓			✓				✓	✓	✓	✓	
6	Contract payment		✓	✓	✓	✓				✓	✓	✓	✓	
7	Change Order Index		✓			✓				✓	✓	✓	✓	
8	Construction Price Fluctuation		✓			✓				✓	✓	✓	✓	✓
9	Weather condition		✓	✓		✓	✓			✓	✓	✓	✓	✓
10	AC			✓	✓	✓				✓	✓	✓	✓	
11	Budget at Completion			✓			✓							
12	EV			✓	✓					✓	✓	✓	✓	
13	PV			✓	✓	✓				✓	✓	✓	✓	
14	Experience of project staff			✓										✓
15	Construction duration					✓	✓			✓	✓	✓	✓	
16	Topography						✓							
17	% of work performed						✓	✓						
18	Increase in labour Cost													✓
19	Lack of experience in the location													✓
20	Inaccurate quantity take-off													✓
21	Traffic volume						✓							
22	Evaluating date						✓							
23	% of work planned						✓							

Sources for cost estimate factors

1. (Wauters & Vanhoucke, 2014)
2. (Kareem Kamoona & Budayan, 2019)
3. (Feylizadeh et al., 2012)
4. (Dastgheib et al., 2022)
5. (Cheng et al., 2010)
6. (Pewdum et al., 2009)

7. (Ottaviani & Marco, 2022)
8. (İnan et al., 2022)
9. (Cheng & Hoang, 2014)
10. (Cheng et al., 2012)
11. (Cheng & Roy, 2010)
12. (Cheng et al., 2013)
13. (Kaming et al., 1997)

Several factors are crucial when forecasting construction project costs from a practitioner's perspective. Firstly, it is essential to capture the actual cost rate, which includes the actual cost by unit measure. This indicator allows for more accurate forecasting and budget adjustments. Similarly, actual productivity, a unit measured per hour or day, is essential. Another factor is unexpected weather conditions, which can cause significant delays, damage materials, or necessitate additional labour, affecting both the timeline and cost. Another key consideration is material price escalation. As prices for critical materials fluctuate, the project's overall cost can increase unexpectedly. Currency exchange rate fluctuations could affect the final cost of imported materials and equipment, which is crucial for those depending on foreign supplies or projects budgeted in different currencies. Inflation rates gradually reduce the purchasing power of a budget, impacting cost forecasting. Lastly, the evaluation date for grasping the metric according to the season acknowledges that certain times of the year can affect both the availability and price of labour and materials and the feasibility of construction work, making timing a crucial element in cost forecasting. The collaborating practitioner has emphasized that these factors contribute to the intricate nature of cost forecasting in construction projects.

Previous studies and expert opinions have identified some critical factors impacting cost forecasting. However, some of them are unavailable for the current research. In this study, the

available factors were considered based on those identified through a literature review and expert opinion to create a preliminary list of factors influencing cost forecasting.

Regarding the output, the Estimate at Completion computation is contingent upon predicting the Estimate to Complete (ETC), given the progressive availability of actual costs throughout the project. This study's approach involves forecasting the ETC at each period and adding the cumulative actual cost to derive the EAC. The following equation calculates the EAC based on the predicted ETC.

$$EAC_{predicted} = AC_{cumulate} + ETC_{predicted} \quad \text{Equation 3.1}$$

Where:

- $EAC_{predicted}$: Estimate at Completion predicted.
- $AC_{cumulate}$: Cumulative Actual Cost
- $ETC_{predicted}$: Estimate to Complete predicted.

Table 3.2 presents a preliminary selection of input variables and an output to further analyze in the subsequent chapter. While the CPI is presented within the data collected, it has been omitted from this preliminary list because it is derived from two inputs under consideration: AC and EV. However, in the next chapter, CPI input will be incorporated into the Pearson correlation analysis, part of the feature selection process, to validate this decision.

Table 3.2 Preliminary Inputs and Output Available

Feature code	Feature Name	Description	Category
x_1	AC	The total cost of a specific work package from the start to the current period.	Input
x_2	EV	Total earned value for a specific work package up to the current period.	Input
x_3	% Duration	The percentage of time passed about the current schedule baseline for a specific work package.	Input
x_4	Labour Cost	The total labour cost of a specific work package from the start to the current period.	Input
x_5	Executed quantity	Work that has been completed on a project relative to the total work required.	Input
x_6	WP Type	Categorization of a specific set of tasks within the project	Input
y_1	ETC	The projection represents the anticipated cost required to complete the remaining work.	Output

It is essential to mention that the data for this study was collected weekly at the work package level. Previous studies on forecasting EAC using a machine learning model have focused on the project level, so their models did not include work package type inputs. However, this research proposes a new approach to predicting EAC at the work package level. To achieve this, the model consists of a work package type, categorizing the work into concrete, piping, and fill. Utilizing the work packages approach in managing construction projects is expected to improve cost forecasting analysis by giving alerts about potential cost overruns at this level, consequently helping the project manager identify and promptly implement targeted corrective actions. Additionally, the work packages approach will likely decrease the chances of experiencing schedule delays and cost overruns (Hammad, 2009).

3.4 Proposed Data Acquisition Model

Collecting and maintaining quality data is crucial for the deep learning model's success. A Data Acquisition Model (DAM) ensures that data is gathered correctly and consistently, increasing the chance of a more accurate model. Ongoing construction projects are dynamic, with costs, progress and performance changing frequently. A data acquisition model allows for real-time data collection, providing up-to-date information for evaluating cost performance and forecasting. Data acquisition models can be built to collect data at a granular level, such as a work package. This level of detail is convenient for addressing and managing potential cost variations. Also, DAM provides transparency in data collection, ensuring that all stakeholders can access reliable data and promote better decision-making. Therefore, obtaining good-quality data is crucial for the organization's success.

Construction companies have different methods of collecting data from onsite projects. Some rely on manual data collection by filling out forms onsite and entering the data into an Excel spreadsheet (El-Omari & Moselhi, 2011). However, this practice is prone to human error, such as missing data or misinterpretation. It is also time-consuming and may cause delays in data reporting. On the other hand, some companies use specialized systems, such as SAP and BST Global, that are more efficient for collecting data. Nevertheless, this software may not cover all the cost-tracking data necessary to calculate and accurately predict costs.

Hence, this study aims to introduce a data acquisition model that establishes a tracking system capable of gathering high-quality data from onsite projects, particularly data needed for cost forecasting. The conceptual design of the database offers an overview of the data requirement. The development was carried out using an Entity Relationship Diagram (ERD).

3.4.1 Entity Relationship Diagram (ERD)

The entities-relationship diagram (ERD) methodology, developed by Chen in 1976, is used to map and formulate data into a project database (P. P.-S. Chen, 1976). The ERD includes entities, relationships, and attributes. Entities are essential objects that exist physically or conceptually.

Relationships are connections between two entities and are used in designing the database. In a binary relationship, the association includes one-to-one (1:1), one-to-many (1:m), and many-to-many (m: n) relationships. Attributes refer to the qualities or features that describe entities; attributes are the data fields containing entity information. The ERD diagram acts as a guide for developers to model data entities and relationships without conflicts.

Developing an ERD is pivotal in establishing a comprehensive data collection framework designed to enhance cost forecasting at the work package level within the context of construction projects. Figure 3.3 illustrates that the ERD encapsulates the relationships and entities crucial for systematically capturing, organizing, and forecasting costs associated with different work package types. The ERD framework is a valuable tool for contractors, facilitating the collection of essential project data for cost forecasting.

The proposed ERD presents a hierarchical structure for project management. At the highest level is the 'Portfolio,' a strategic collection of projects and programs to achieve overall business objectives (Project Management Institute, 2018). The second level comprises programs that are clusters of interconnected projects; their success depends on the completion of each project in collaboration. In the third level, each construction project is represented by the 'Project' entity. The last level is the work package, which is the level of evaluation in the proposed predictive model. Figure 3.3 illustrates the Breakdown Structure Schema.

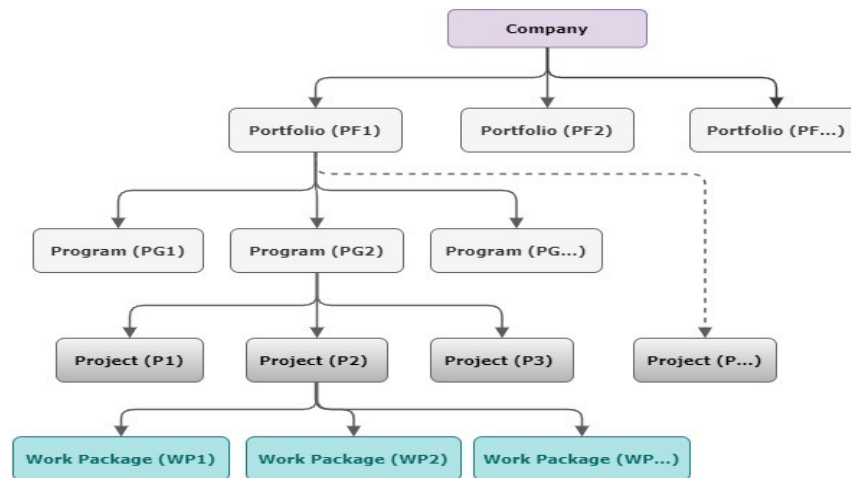


Figure 3.3 Breakdown Structure Schema

The ERD proposed represents a data acquisition model for managing construction projects to forecast EAC at the Work Package level. This ERD is comprised of several entities, each of which is integral to the database's overarching structure and is interconnected through various relationships. At the top of the ERD is the "Project" entity, which holds attributes such as 'ID,' 'Name,' 'Project Manager/Controller,' 'Owner Company,' 'Location,' 'Project Type,' 'Contract Number,' 'Contract Type,' 'Contract Amount,' 'Contract Duration,' 'Original Budget at Completion (BAC),' and 'Schedule Baseline.' It is a central node that connects to the 'Company,' 'Portfolio,' 'Program,' 'Project Update,' and 'Work Package (WP).'

The central entity in the ERD is the 'Work Package,' which includes attributes such as 'Name,' 'ID,' 'WP Type,' 'Key Quantity,' 'Description,' and 'Unit of Measure,' allowing for granular tracking and management of project tasks. The WP entity relates to 'Weekly Progress per WP,' which captures the weekly progress on work packages. This entity includes 'Earned Value (EV),' 'Planned Value (PV),' 'Actual Cost (AC) for Equipment, Labour, Material, Subcontractor,' 'Quantity,' and 'Comments,' providing critical data points for performance measurement and management. The

'WP Original Baseline' and 'WP Update' entities establish a plan or reference point against which the performance and progress of specific work packages can be measured.

The ERD employs one-to-many ('1', 'm') and many-to-one ('m', '1') relationships. For instance, a single project may contain multiple work packages, each with multiple updates and progress reports.

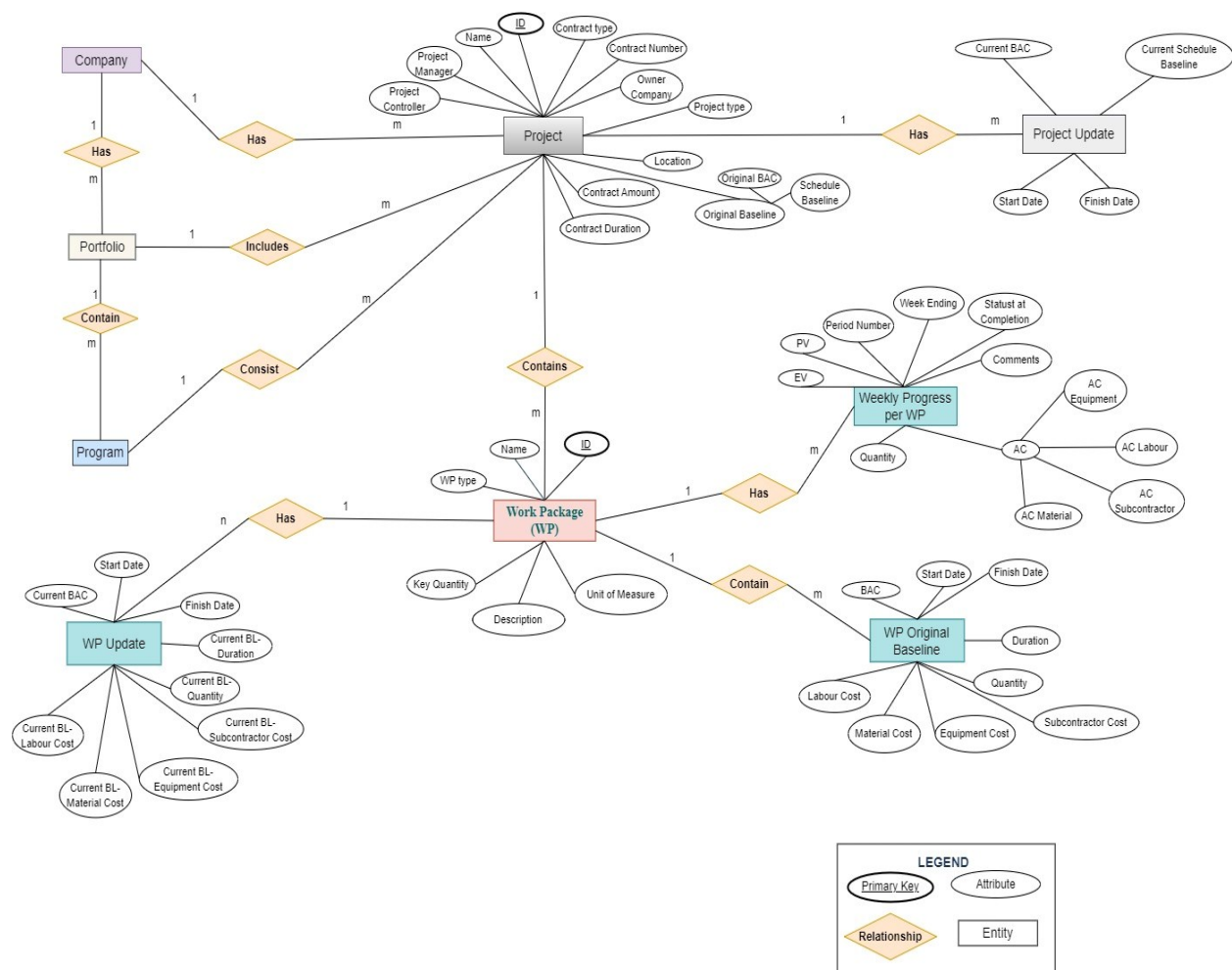


Figure 3.4 Entity Relationship Diagram

3.5 Data Collection Process

The historical data cases in this study consist of five mining projects developed between June 2014 and June 2016 by one construction company in Peru. The project comprises part of the expansion of the mining unit, including the following main work package: earthworks, concrete and steel structures, mechanical and piping installation, guides and anchor blocks, electrical installation, and instrumentation. This data is based on weekly cost-related information for each project.

In this study, historical data is systematically acquired through a range of reports supplied by the contractor. These reports encompass diverse information, including the final project report, weekly production report, weekly earned value management reports, equipment control data, and labour hours data, among other general documentation. Most historical data is available in the time-series format and at the work package level, contributing to the robustness of the data collection process for the study.

The industry partner collaborating in this study has followed a comprehensive tracking system encompassing a variety of spreadsheets tailored for data collection. Most data collected for this study was obtained from two reports provided by the contractor:

- The Earned Value Management Report: This report furnishes weekly insights at the work package level, featuring data points such as work package descriptions, earned value (\$), actual cost (\$), budget at completion (\$), CPI (Cost Performance Index) for both the current period and cumulatively, and the percentage of physical completion.
- Cost Control reports: This report compares the periodically incurred actual costs with the budgeted costs. The actual cost is divided into four categories: labour, material, equipment, and subcontractors. It is important to note that the labour cost had periodic information,

whereas the material, equipment, and subcontractor costs had the total amount at the end of the project.

- The Weekly Production Report: This report focuses on monitoring labour and production, detailing actual hours expended weekly, actual quantities executed and planned quantities based on critical units for each work package.

While the contractor provided valuable information to build the cost forecasting data set, there are specific attributes that the contractor did not provide. These missing elements include the planned value per week at the work package level, the SPI (Schedule Performance Index), material price escalation, information related to the impact of unexpected weather conditions, and detailed information concerning contract payments, including delays in payments or discrepancies between payments and actual work executed onsite.

3.6 Data Description

The dataset collected for this study is briefly explained in the following section. An example of the initial dataset overview is provided in Table 3.3.

- Actual Costs to Date:

This input consists of ongoing costs as the project progresses. This data is available at the work package level with weekly frequency for each construction project. This information encompasses the actual direct cost, including materials, equipment, subcontractors, and labour costs, excluding indirect costs. Direct costs are specific and tangible to the project, including materials, labour, equipment, and subcontractors (Amos, 2012). Indirect costs are not directly accountable or tangible to the project, such as business taxes, home office overhead, or transportation fleet distributed cost, but are necessary for the business to remain solvent (Amos,

2012). Table 3.5 displays the lower and upper bounds (in monetary units) for the actual cost at completion associated with each work package type.

Table 3.3 Initial Dataset Overview

Project	Work Package Type	Unit	Start Date	Finish Date	Period	Duration (weeks)	BAC (\$)	AC to date (\$)	EV to date (\$)	Key quantity to date (unit)	Actual labour Cost (\$)
Project 01	Concreto	m3	4-Dec-14	10-Dec-14	1.00	59.00	1,736,919.69	5,428.58	6,195.61	10.11	3,097.00
Project 01	Concreto	m3	11-Dec-14	17-Dec-14	2.00	59.00	1,736,919.69	36,090.56	9,259.89	16.71	18,316.00
Project 01	Concreto	m3	18-Dec-14	24-Dec-14	3.00	59.00	1,736,919.69	82,552.47	24,772.37	39.43	26,163.00
Project 01	Concreto	m3	25-Dec-14	31-Dec-14	4.00	59.00	1,736,919.69	122,488.09	25,688.44	40.92	22,990.00
Project 01	Concreto	m3	1-Jan-15	7-Jan-15	5.00	59.00	1,736,919.69	171,070.01	35,613.77	57.12	27,721.00
Project 01	Concreto	m3	8-Jan-15	14-Jan-15	6.00	59.00	1,736,919.69	201,308.59	44,280.76	71.27	17,214.00
Project 01	Concreto	m3	15-Jan-15	21-Jan-15	7.00	59.00	1,736,919.69	248,464.94	44,280.76	71.27	27,056.00
Project 01	Concreto	m3	22-Jan-15	28-Jan-15	8.00	59.00	1,736,919.69	283,760.79	44,806.38	72.32	20,406.00
Project 01	Concreto	m3	29-Jan-15	4-Feb-15	9.00	59.00	1,736,919.69	316,208.04	73,241.88	107.42	19,000.00
Project 01	Concreto	m3	5-Feb-15	11-Feb-15	10.00	59.00	1,736,919.69	346,337.49	83,511.86	136.02	17,594.00
Project 01	Concreto	m3	12-Feb-15	18-Feb-15	11.00	59.00	1,736,919.69	370,514.54	89,213.78	164.52	14,554.00
Project 01	Concreto	m3	19-Feb-15	25-Feb-15	12.00	59.00	1,736,919.69	389,311.97	141,537.63	290.59	9,785.00
Project 01	Concreto	m3	26-Feb-15	4-Mar-15	13.00	59.00	1,736,919.69	412,070.03	153,024.42	318.61	12,787.00
Project 01	Concreto	m3	5-Mar-15	11-Mar-15	14.00	59.00	1,736,919.69	439,730.30	176,075.72	354.08	15,124.00
Project 01	Concreto	m3	12-Mar-15	18-Mar-15	15.00	59.00	1,736,919.69	464,587.84	177,185.66	355.90	14,193.00
Project 01	Concreto	m3	19-Mar-15	25-Mar-15	16.00	59.00	1,736,919.69	497,896.62	179,954.49	360.45	18,981.00
Project 01	Concreto	m3	26-Mar-15	1-Apr-15	17.00	59.00	1,736,919.69	512,854.33	192,521.23	372.64	10,279.00
Project 01	Concreto	m3	2-Apr-15	8-Apr-15	18.00	59.00	1,736,919.69	527,309.30	192,521.23	372.64	8,322.00
Project 01	Concreto	m3	9-Apr-15	15-Apr-15	19.00	59.00	1,736,919.69	553,573.64	260,594.80	468.84	17,822.00
Project 01	Concreto	m3	16-Apr-15	22-Apr-15	20.00	59.00	1,736,919.69	615,981.91	369,389.09	614.46	41,743.00
Project 01	Concreto	m3	23-Apr-15	29-Apr-15	21.00	59.00	1,736,919.69	697,353.91	430,018.59	694.30	50,825.00
Project 01	Concreto	m3	30-Apr-15	6-May-15	22.00	59.00	1,736,919.69	777,293.15	466,662.63	751.08	45,980.00
Project 01	Concreto	m3	7-May-15	13-May-15	23.00	59.00	1,736,919.69	820,301.69	512,003.64	820.78	24,282.00
Project 01	Concreto	m3	14-May-15	20-May-15	24.00	59.00	1,736,919.69	893,795.80	588,981.87	938.80	39,634.00
Project 01	Concreto	m3	21-May-15	27-May-15	25.00	59.00	1,736,919.69	984,077.11	608,352.06	963.64	52,744.00
Project 01	Concreto	m3	28-May-15	3-Jun-15	26.00	59.00	1,736,919.69	1,057,379.74	664,222.32	1,049.72	41,249.00
Project 01	Concreto	m3	4-Jun-15	10-Jun-15	27.00	59.00	1,736,919.69	1,187,979.03	717,408.64	1,137.72	73,397.00
Project 01	Concreto	m3	11-Jun-15	17-Jun-15	28.00	59.00	1,736,919.69	1,341,861.54	861,976.52	1,437.40	85,025.00
Project 01	Concreto	m3	18-Jun-15	24-Jun-15	29.00	59.00	1,736,919.69	1,498,786.81	942,130.16	1,561.28	86,868.00
Project 01	Concreto	m3	25-Jun-15	1-Jul-15	30.00	59.00	1,736,919.69	1,656,872.07	969,806.57	1,605.82	90,269.00
Project 01	Concreto	m3	2-Jul-15	8-Jul-15	31.00	59.00	1,736,919.69	1,694,438.16	1,028,493.34	1,773.59	50,027.00
Project 01	Concreto	m3	9-Jul-15	15-Jul-15	32.00	59.00	1,736,919.69	1,736,849.86	1,108,636.48	1,983.10	55,708.00
Project 01	Concreto	m3	16-Jul-15	22-Jul-15	33.00	59.00	1,736,919.69	1,842,858.17	1,159,535.90	2,060.98	65,455.00
...

Table 3.4 Actual Cost at Completion Bounds

Work Package Type	Number of Work Packages	Number of Records	Actual Cost at Completion(\$)	
			Lower Bound	Upper Bound
Concrete	8	439	1,422,346.05	4,172,899.84
Piping (HDPE)	9	388	1,214,274.77	7,292,779.17
Backfill	6	328	1,474,004.06	3,683,582.26

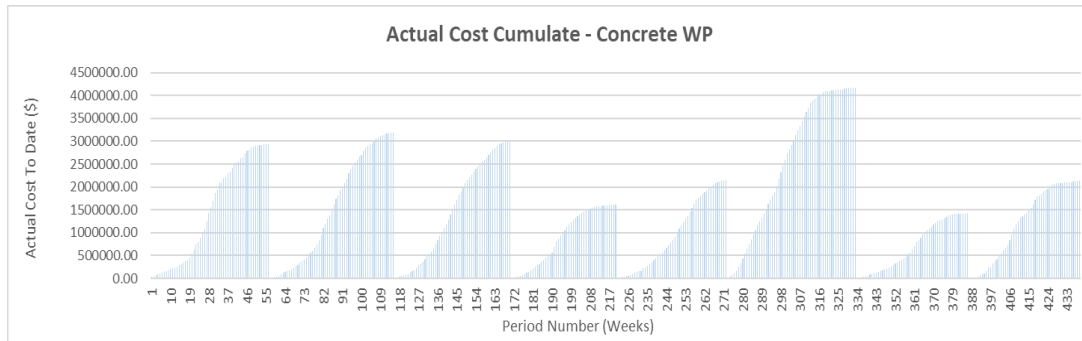


Figure 3.5 Actual Cost to Date _Concrete dataset

- Earned Value to Date:

This attribute represents the budget for the work package accomplished at a specific time. EV is calculated by multiplying the percentage completed by the BAC. Table 3.5 displays the lower and upper bounds (in monetary units) for the earned value associated with each work package type.

Table 3.5 Earned Value Bounds

Work Package Type	Number of Work Packages	Number of Records	Earned Value (\$)	
			Lower Bound	Upper Bound
Concrete	8	439	1,058,914.93	3,321,901.78
Piping (HDPE)	9	388	912,213.67	8,228,545.37
Backfill	6	328	930,789.85	2,886,114.86

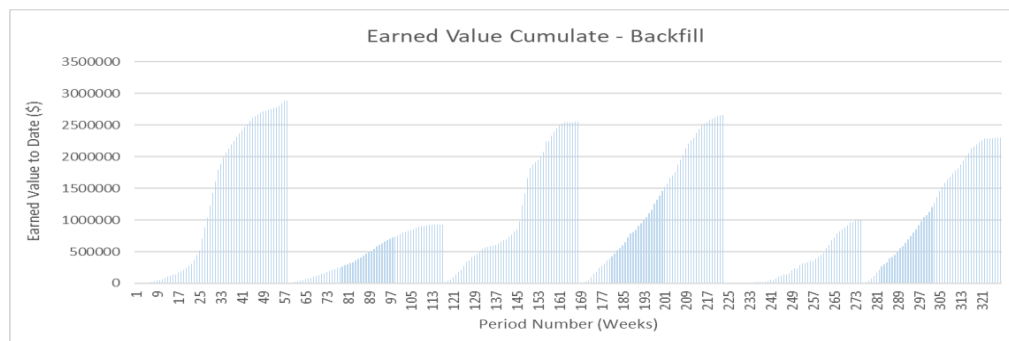


Figure 3.6 Earned Value Cumulate _Fill dataset

- **Duration % Completed:**

This input refers to the percentage of the total project duration that has been completed at a given point in time. This was obtained by considering each work package's actual start and finish dates and the current baseline duration. Table 3.6 shows the lower and upper bound (measures in weeks) for the duration input associated with each work package type.

Table 3.6 Duration Bounds in Weeks

Work Package Type	Number of Work Packages	Number of Records	Duration (Weeks)	
			Lower Bound	Upper Bound
Concrete	8	439	50.00	61.00
Piping (HDPE)	9	388	38.00	51.00
Backfill	6	328	51.00	59.00

- **Actual Labour Cost:**

Labour costs are a significant portion of the actual cost, representing the total payments made to field workers who carry out the work package task. This input helps the model understand trends and patterns in labour spending throughout the project.

- **Work Package Type:**

The work package type attribute represents the category of the work executed. The projects collected include nine main work packages: Excavation, fill, rockfill, concrete (including concrete placement, formwork, and steel reinforcement), piping, assembly of light and medium structures, assembly of heavy structures, electrical cables, and instrumentation. However, subcontractors executed some work package types, and the weekly complete information is unavailable. Thus, this study only considers work packages that contain all completed cost data. These are fill, concrete, and piping (HDPE).

- **Executed quantity:**

The executed quantity comprises the quantity completed (for each work package) at a given time; it is expressed according to the key unit defined for each work package.

3.7 Limitations in the Acquired Dataset

The data acquired process encountered a set of constraints and challenges, primarily stemming from the unavailability or confidentiality of specific data on the contractor's side. At the onset of the data collection process, a preliminary meeting was held with a Canadian company to introduce the research investigation, inquire about their current methodologies and practices in the cost forecasting task and request information to build the proposed forecasting model. The data provided by this Canadian entity was notably well organized. However, it only included some crucial attributes for developing the forecasting model. A principal constraint imposed by the Canadian company was the stringent confidentiality surrounding cost-related data. Given that the proposed model's main objective is predicting the project's final cost, the absence of access to cost-related data posed a significant limitation.

In response to this limitation, an alternate option was explored, leading to a connection with a Peruvian company. This company possessed data from construction projects completed around eight years ago, offering an alternative source of information. This strategic connection was established to acquire a comprehensive dataset to build the predicted model.

Although the dataset provided by the Peruvian company included some essential attributes required for the model, it notably omitted other significant attributes. Specifically, the dataset lacked data regarding the planned value, SPI, indexes for construction price fluctuations, information about extreme weather conditions, change orders, and monthly project progress

payments made by the owner to the contractor. This study constructed the current model using the available information despite the unavailable data. However, we hope that future investigations will consider incorporating these additional attributes to enable the development and evaluation of an improved model version.

Chapter 4: Development of Computational Model for Cost Forecasting

4.1 Introduction

This research aims to develop a deep learning framework to predict the final cost in construction projects by determining influential factors in cost forecasting, using historical data, and employing a deep learning algorithm. Once the forecasting model is developed, the study will create a graphical user interface to deploy the forecasting model. The resulting model will help project managers make better-informed decisions and predict the final cost in real time during the construction project execution. Chapter 4 proposes a framework for developing the model and engineering a graphical user interface for data collection and model deployment. This framework involves preprocessing the collected data, building and configuring three models, evaluating the models using performance metrics, selecting the optimal forecasting model, and developing GUI for collecting data and directly applying the model. In the subsequent sections, a comprehensive elaboration of the main steps, including their respective sub-steps, will be presented in detail. This detailed exposition aims to thoroughly understand the sequential processes for developing a forecasting model based on a deep learning algorithm. A pictorial depiction of the workflow for a forecasting model is given in Figure 4.1.

4.2 Data Preprocessing

As per standard machine learning practices, a systematic preprocessing protocol is essential to refine raw data for optimal utilization in model training and validation endeavours. Data preprocessing can significantly impact the predictive capability of a model, potentially determining its success or failure (Kuhn & Johnson, 2013). The preprocessing steps aim to remove categorical data, address missing values through deletion or imputation, and organize each time

observation into a single vector row (Barrera-Animas et al., 2022). The preprocessing procedure developed in this study is explained in the following sections.

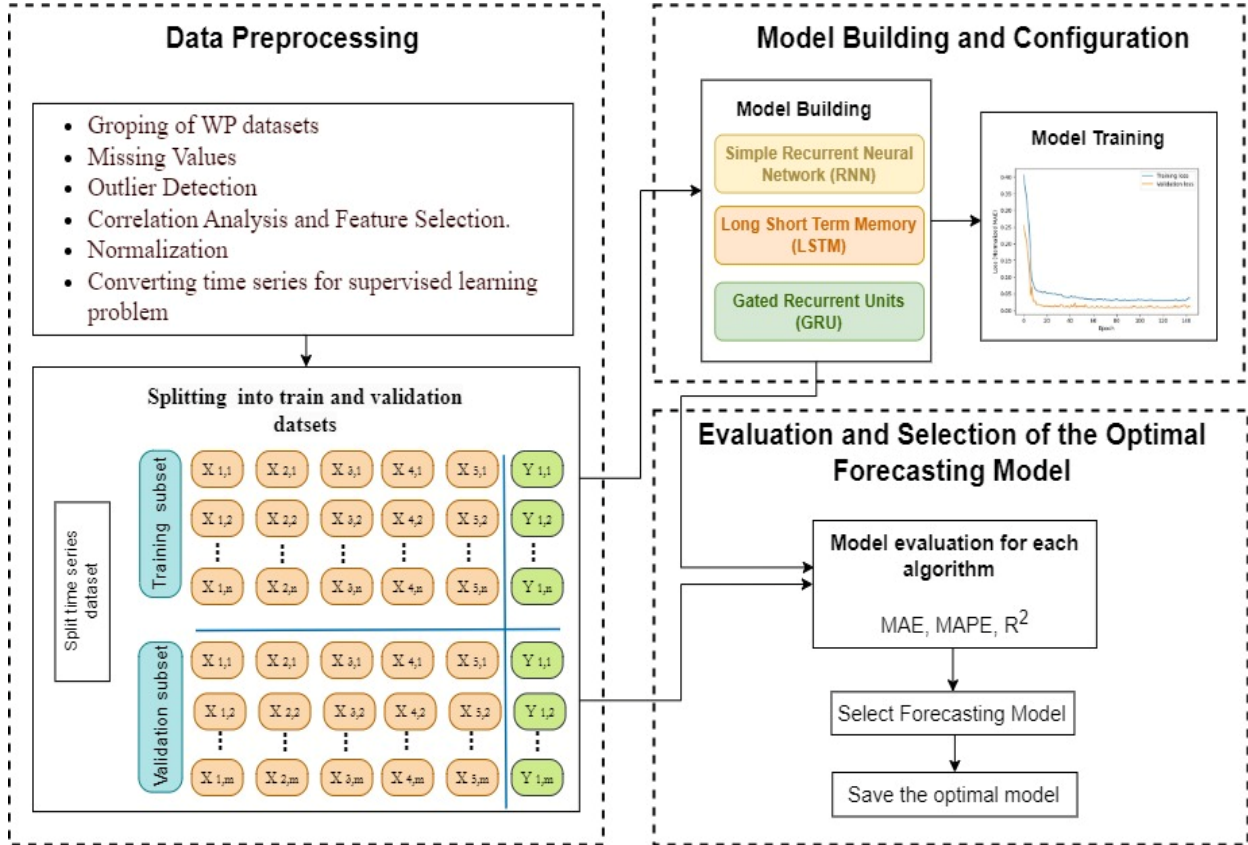


Figure 4.1 Forecasting Model as a Workflow

4.2.1 Grouping of Work Package Dataset

In the first step, this study considers separating the dataset based on work package types. The collected data reveals three distinct work package types: concrete, piping (HDPE) and backfill. This procedural segmentation safeguards the singularities inherent in cost forecasting for each work package throughout the training process. The complete dataset for each work package type comprises weekly recordings, including actual cost, earned value, executed quantity, actual labour cost, and duration (start and finish date). Moreover, the dataset provides project-level records, including name, location, start date, finish date, contract amount and actual estimate at completion.

After dividing the data for each work package category, the "Work Package Type" input was removed. The date inputs, including start and finish date, were also removed because they duplicated information in the duration attribute.

4.2.2 Dealing with Missing Values in Time Series

Missing values are common in data due to data entry errors or incomplete data collection. However, these missing values can negatively impact the accuracy of forecasting models. Regarding time series data, handling missing values is different because the data knows what preceded the missing value. In this study, we used the "Replace Missing Values (Series)" operator from Rapid Miner Study software to address this issue. This software replaces missing numerical values with the average of the neighbouring values in the series, as shown in Equation 4.1. Figure 4.2 illustrates replacing missing values in time series data using Rapid Miner Studio software.

$$x'_t = \frac{x_{t-1} + x_{t+1}}{2} \quad \text{Equation 4.1}$$

Where:

- x'_t : Represent the padding value.
- x_{t-1} and x_{t+1} : Denote the values of the previous time and the next time.

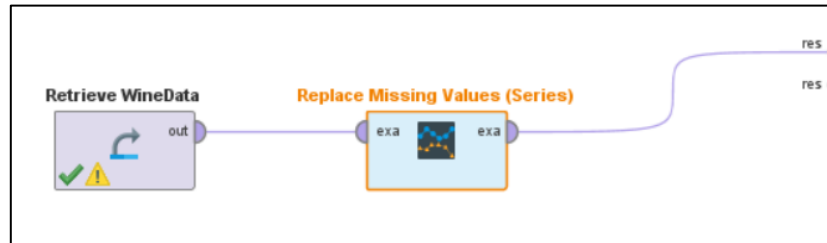


Figure 4.2 Missing Values Process

4.2.3 Outlier Detection

Outliers are defined as data point observations that are significantly far from the mainstream of the data (Kuhn & Johnson, 2013). They can be caused by several factors, such as data entry errors, measurement errors, or genuinely unusual events. Identifying and handling outliers is essential in data preprocessing because they can significantly impact the predicted model results. The Local Outlier Factor operator in RapidMiner identifies values far from their local neighbourhoods, making it a valuable method for detecting local outliers. Figure 4.3 displays the outlier detection process in RapidMiner Studio 9.10.013, including a step-by-step description.

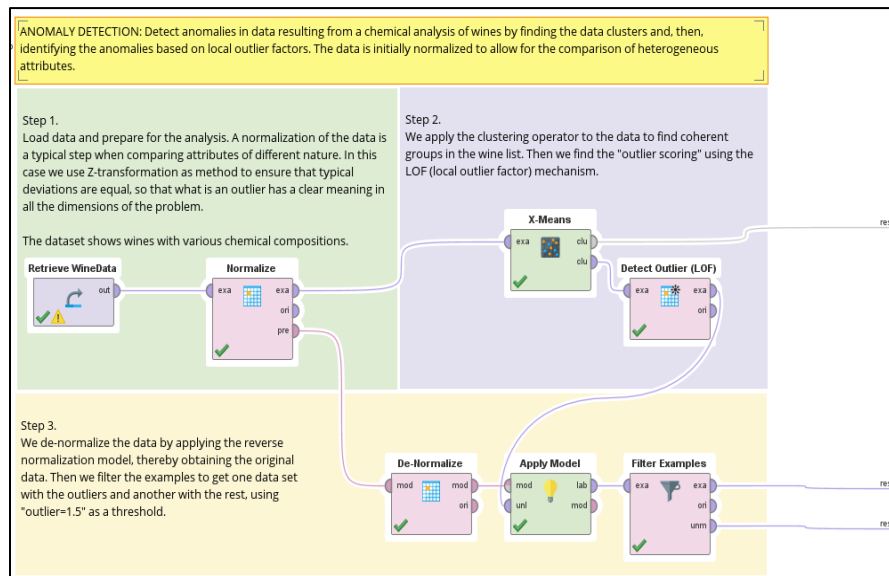


Figure 4.3 Outlier Detection Process

4.2.4 Correlation Analysis and Feature Selection

The study analyzed the Pearson correlation coefficient (Liu et al., 2020) between six preliminary input parameters and a single output value (ETC) to comprehend the relationship between input parameters and output. This was done to evaluate their linear relationship. The Pearson Correlation Coefficient is a statistical method to measure the linear relationship between two inputs, X and Y.

The resulting value ranges from +1 to -1 (Liu et al., 2020). A value of +1 implies an absolute positive correlation, meaning that as variable X increases, variable Y also increases. On the other hand, a value of -1 implies a perfect negative correlation, meaning that as variable X increases, variable Y decreases. A value equal to 0 indicates no linear correlation. Figure 4.4 illustrates the correlation matrix analysis.

The study has identified varying degrees of correlation between the six inputs and the ETC. The CPI reveals very low correlations with all the variables, ranging from - 0.03 to 0.33, indicating almost no linear relationship between CPI and the other variables in the set. However, inputs such as AC cumulate, % Duration and Labour Cost cumulate exhibit significant correlations with the ETC output. Finally, the AC cumulate, and EV cumulate display a high correlation, like the AC cumulate and the labour cost cumulate. Despite this high correlation, it has been decided to retain the AC cumulate variable due to the limited number of inputs, with each input providing valuable information. Table 4.1 presents the final list of features included in the cost forecasting model.

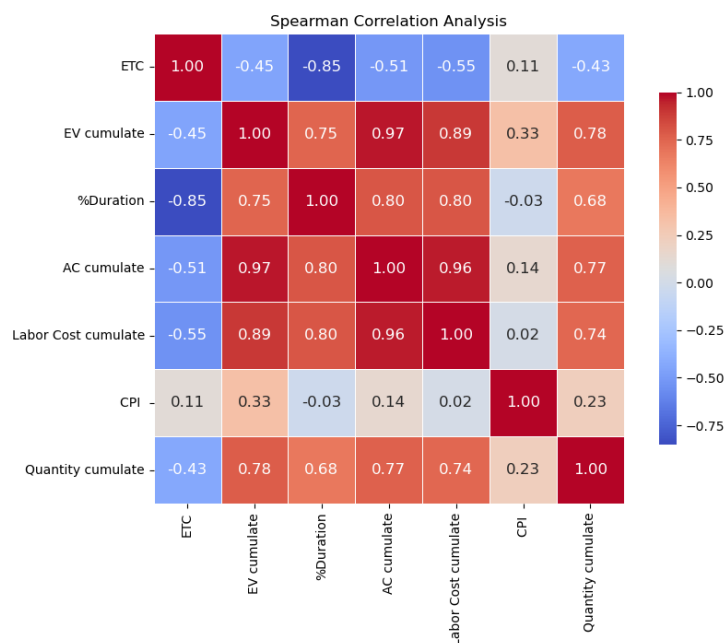


Figure 4.4 Spearman Correlation Analysis

Table 4.1 Forecasting Model Features Variables

Feature code	Feature Name	Index	Category
x_1	Actual Cost	$AC_{cumulate}$	Input
x_2	Earned Value	$EV_{cumulate}$	Input
x_3	% Duration	$\frac{Duration_{to\ day}}{Current\ Baseline\ duration}$	Input
x_4	Labour Cost	$Labour\ cost_{cumulate}$	Input
x_5	Executed quantity	$.Executed\ quantity_{cumulate}$	Input
y_1	Estimate to Complete	ETC	Output

4.2.5 Time Series Data for Supervised Learning Problem

Time series refers to observations of different values corresponding to specific periods. Each observation contains a time and an observation element. The observation element may correspond to a single variable, a univariate time series, or multiple variables, also called a multivariate time series (Wen & Li, 2023). The dataset collected for this study corresponds to a multivariate time series encompassing various inputs for each observation. This dataset structure is particularly suitable for applying regression-based deep learning algorithms, where the objective is to predict continuous outcomes based on the temporal relationship and intricate patterns inherent in the collected multivariate inputs.

The dataset needs to undergo specific preparations to prepare time series data for supervised deep learning algorithms, transforming it into input and output components. In this research, the dataset consists of five input time series, one output time series, and a time window of length 3. The proposed model uses three weeks of data to predict the estimate to complete (ETC) target of the following week. The input time series includes actual cost, earned value, duration percentage completed, labour cost, and quantities. The output time series corresponds to the ETC. Since the

Where:

- Figure 4.5 displays data packing five inputs, one output from three preceding steps, and one target.



Dataset normalization is a common approach performed in Machine Learning models when their features have different magnitudes of values between them (Barrera-Animas et al., 2022). This

study employs the min-max normalization function from Python's Scikit learn library, which involves linearly transforming the values of each parameter to map them into a standardized range.

Equation 4.3 for this calculation is as follows:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad \text{Equation 4.3}$$

Where:

- x denotes a value in the sequence of original variables.
- x_{max} and x_{min} Denote the maximum and minimum values in variables, respectively.

When working with time series data for a regression problem using deep learning algorithms, it is important not to split the data randomly. This is because time series data inherently possess temporal dependencies and sequential patterns. Randomly splitting the data would disrupt these temporal relationships, leading to several issues. In the following step, the dataset is split into three subsets, keeping the data sequence structure to evaluate the learning performance objectively.

- The first subset, the training set, comprises around 75% of the records and is used to develop predictive models.
- The second subset, the validation set, consists of about 15% of the dataset and is used to evaluate the model's performance and fine-tune its parameters.
- The last subset, the testing set, the study case set or unseen data, comprises around 10% of the data set and assesses the model's future performance.

Figure 4.6 represents the division of a dataset into three splits: The training, validation, and testing set, which are used during the deep learning model development process.

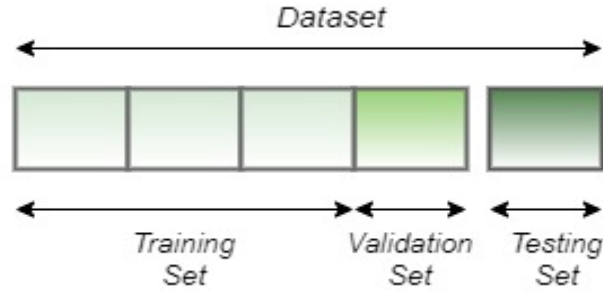


Figure 4.6 Data Split Diagram

Considering the dataset composed of finalized projects, the division into subsets for training, validation and testing was carried out with the intention of including a completed project in each evaluation category. Consequently, the percentages delineated below are approximations.

The dataset's preprocessing stage has been completed until this section. This ensures the data is clean, normalized, and suitable for analytical model processing. This study will build a predictive model using the refined data to establish a forecasting framework in the upcoming section.

4.3 Developing a Deep Learning-Based Cost Forecasting Model

This section describes the experiments to train three deep-learning models for time series forecasting. This research evaluates the effectiveness of three network architectures for predicting the ETC for the subsequent period. The comparative analysis focuses on Simple RNN, LSTM and GRU, assessing their respective capabilities in accurate ETC forecasting.

Deep learning methods such as RNN, LSTM, and GRU have been proven to help learn temporal dependence structures for time series forecasting challenges. The selected model pretends to predict the target variable at one future time step, denoted as y_{t+1} , based on historical time series data $\{x_1, x_2, \dots, x_t\}$ that ends at time t . This data includes a vector of m input parameters, represented as x_t , observed at time t .

This study applies a length sliding time window of size three to make the input to the model uniform in length. Mathematically, the function relationship learned by the deep learning model can be expressed as shown in Equation 4.4.

$$\hat{y}_{t+1} = f(x_t, x_{t-1}, x_{t-2}, y_t, y_{t-1}, y_{t-2}) \quad \text{Equation 4.4}$$

Where \hat{y}_{t+1} is the target variable predicted for time $t+1$, and y_t, y_{t-1}, y_{t-2} are the target values observed in their given time and x_t, x_{t-1}, x_{t-2} are the vector of m observed input parameters.

4.3.1 Deep Learning for Time Series Forecasting

This section provides a mathematical description of the deep learning algorithms used in this research: Simple Recurrent Neural Network, Long Short-Term Memory, and Gated Recurrent Units.

- **Simple Recurrent Neural Network (RNN):**

A Simple Recurrent Neural Network (RNN) is an artificial neural network type designed to work for data involving sequences (Tyagi & Abraham, 2022). RNN is specifically designed to identify patterns within time series data, such as text, audio, video, stock market prices or forecasting tasks. RNN has the concept of "memory," which benefits the collection of the states or information of previous inputs to generate the following sequence output (Hossen et al., 2018). As shown in equation 4.6, the output \hat{y}_t is determined by the internal state, S_t , which depends on the current input x_t and the previous state S_{t-1} . The Simple RNN formula for calculating the current state of RNN can be expressed as follows (Wu et al., 2020):

$$S_t = f(\omega_x x_t + \omega_s S_{t-1} + b_s) \quad \text{Equation 4.5}$$

$$\hat{y}_t = f(\omega_y S_t + b_y) \quad \text{Equation 4.6}$$

Where:

- S_t means hidden state (or activation) at time step t .
- $x_t \in \mathbb{R}^m$ denotes the input vector of m inputs at time t .
- $\omega_x \in \mathbb{R}^{m \times (m+n)}$ and $\omega_y \in \mathbb{R}^{m \times n}$ learned parameters, n is the number of neurons in the RNN layers.
- b_s and b_y are bias vectors for the internal state and output, which helps to define the model.

\hat{y}_t represents the output vector at time t .

- f represents the activation function, the sigmoid function or ReLU.

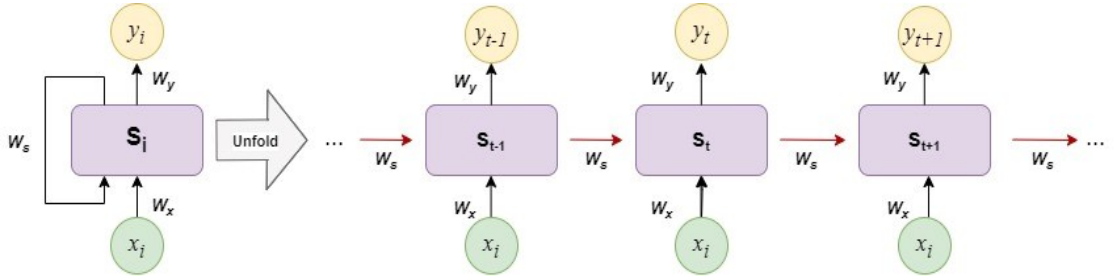


Figure 4.7 RNN architecture

Although RNN models the time series well, it is hard to learn when processing long-term dependencies due to the vanishing problem (Le & Zuidema, 2016; Shi et al., 2022; Wu et al., 2020)

- **Long Short-Term Memory (LSTM):**

LSTM networks are variants of the RNN, which aims to overcome the vanishing gradient problem by incorporating memory control units to determine the information to forget or retain (J. Xu et al., 2024) and learn long-term dependency in time series data. The LSTM network

unit has three gates (Hochreiter & Schmidhuber, 1997). These three gates comprise an input, forget, and output gate to control the information flow across the cells and avoid gradient vanishing and explosion (G. Chen, 2018). Mathematically, it is given as follows:

$$\text{- } \textbf{Forget gate layer: } f_t = \sigma (W_f S_{t-1} + W_f S_t) \quad \text{Equation (4.9)}$$

$$\text{- } \textbf{input gate: } i_t = \sigma (W_i S_{t-1} + W_i S_t) \quad \text{Equation (4.9)}$$

$$\text{- } \textbf{intermediate cell state} \quad \tilde{C} = \tanh(W_c S_{t-1} + W_c S_t) \quad \text{Equation (4.10)}$$

$$\text{- } \textbf{cell state} \quad C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad \text{Equation (4.11)}$$

$$\text{- } \textbf{output gate} \quad o_t = \sigma (W_o S_{t-1} + W_o S_t) \quad \text{Equation (4.12)}$$

$$\text{- } \textbf{hidden state} \quad S_t = O_t * \tanh(C_t) \quad \text{Equation (4.13)}$$

$$\text{- } \textbf{output vector} \quad \hat{y}_t = f (\omega_y S_t + b_y) \quad \text{Equation (4.14)}$$

Where:

- $x_t \in \mathbb{R}^m$ is input vector at time t.
- S_t : output vector.
- C_t : cell state vector.
- f_t : forget gate vector.
- i_t : input gate vector.
- O_t : output gate vector.
- W, U and f are the parameter matrix and vector.
- \hat{y}_t : output vector at time t .

σ is the non-linear sigmoid activation function, and \tanh is the hyperbolic tangent function mathematically expressed as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \text{Equation (4.15)}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \text{Equation (4.16)}$$

- **Gated Recurrent Unit (GRU)**

GRU is a variant of LSTM that aims to overcome the vanishing gradient limitation in Simple RNN by incorporating two gates: an update gate (Z_t), and a reset gate (r_t).

This study explains the mathematics behind that process and the calculation of the update gate.

- Reset gate (r_t) establishes how much of the past information to forget.

$$r_t = \sigma(W_r \cdot [s_{t-1}, x_t] + b_r) \quad \text{Equation (4.17)}$$

- Update gate (Z_t) decides how much of the past information (previous hidden state s_{t-1}) needs to be passed along to the next step.

$$z_t = \sigma(W_z \cdot [s_{t-1}, x_t] + b_z) \quad \text{Equation (4.18)}$$

- Candidate Hidden State:

$$\tilde{S}_t = \tanh(W \cdot [r_t * s_{t-1}, x_t]) \quad \text{Equation (4.19)}$$

- Final Hidden State:

$$S_t = (1 - z_t) * s_{t-1} + z_t * \tilde{S}_t \quad \text{Equation (4.20)}$$

- Output vector at time t :

$$\hat{y}_t = f(\omega_y S_t + b_y)$$

Where σ denotes the sigmoid function, W 's are the parameters matrix, x_t is the input at time t , s_{t-1} represent the hidden state from the previous time step, and b 's are the bias parameters.

\tilde{S}_t represent the candidate's hidden state determines what to collect from the current memory content. S_t is the final hidden state for the current time state.

4.3.2 Cost Forecasting Model Design

This study develops RNN, LSTM, and GRU predictive models leveraging the robust capabilities of Keras, a high-level neural network running on top of TensorFlow, a comprehensive open-source platform for a machine learning library. The development process, including model architecture design, training, and evaluation, uses the Python version 3.9.7. This study utilizes Keras version 2 and TensorFlow version 2.14.0, ensuring compatibility and leveraging the latest features available. This research's computational environment includes an Intel Core i7 processor 2.3 GHz, 16 GB of RAM, and an NVIDIA Geforce RTX 3070 laptop GPU. This hardware configuration provided the necessary computational resources to train deep learning models, allowing for rapid iteration and robust performance evaluations. The following are the main steps involved in model building.

a) Model Setting

Based on the LSTM, GRU and Simple RNN network theory introduced above, this study has designed a regression model for cost forecasting. These models consider the relationship between the estimate to complete and five inputs (AC, EV, % Duration, actual labour cost and quantity executed). The model setup consists of five components: an input layer, two hidden layers, a dropout layer, and an output layer. Figure 4.8 describes code that uses Kera's library to set up a neural network model.

```

## SETUP THE GRU MODEL
model = Sequential()
model.add(GRU(32, activation='relu', input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
model.add(GRU(32, activation='relu', return_sequences=False))
model.add(Dropout(0.01))
model.add(Dense(1))

```

Figure 4.8 GRU Neural Network Setup

Following is a breakdown of the code snippet, including the hyperparameters selected for GRU Neural Network Setup:

- **Input Layer:** This refers to the initial neural network layer, which includes five inputs (AC, EV, % Duration, Actual labour cost cumulate, and quantity performed) and one output (ETC) from the previous three weeks.
- **First Hidden Layer:** The first Hidden Layer comprises 32 units or neurons, each with internal mechanisms such as input, forget, and output gates. It is activated by the Rectified Linear Unit (ReLU) function and set up to process input sequences while returning sequences to the next layer. This layer is crucial in enabling the model to operate sequence data effectively.
- **Second Hidden Layer:** The second hidden comprises 32 units and ReLU activation, configured to return only the final output. This layer processes the sequential information from the first hidden layer and prepares the model for the final prediction stage.
- **Dropout Layer:** The dropout rate measures the percentage of dropped input units (neurons) during training, which helps prevent overfitting. The proposed model's dropout value is 0.01, which means that 1% of the neurons in the preceding layer will

be randomly set to zero during each training iteration, effectively ignored. The dropout hyperparameter enhances the model's generalization ability with new, unseen data.

- Dense Output Layer: The Dense layer has only 1 unit, corresponding to the single output value (ETC).

b) Model Training

This study trains the models using a prepared training set after setting up the model. In this process, the model continuously adapts its weights to learn the features of the input samples gradually (J. Xu et al., 2024). In each training epoch, the model is optimized by monitoring its performance. Combining these steps enables the model to automatically learn the data features from the training set and perform better in subsequent tasks. The model gradually improves its ability to predict cost forecasting through training. During the training process, this research chose the MAE loss function, the Adam algorithm, as the optimization algorithm. The MAE loss function is crucial during model training, especially for regression tasks, as it is the guiding metric for model optimization. By minimizing MAE, the model is trained to make predictions as close to the actual values as possible, which in turn helps to create accurate and reliable regression models.

c) Model Evaluation

Evaluating the performance of the proposed Simple RNN, LSTM and GRU models consists of comparing the predicted results with the actual value using four indicators: Mean absolute error (MAE), root mean square error (RMSE), coefficient of determination R^2 , and mean absolute percentage error (MAPE).

- The MAE measures the average of errors between predicted and actual values. MAE uses the absolute value of each error, ensuring that all errors are treated equally. This approach

is helpful as squared error metrics can penalize more significant errors unfairly. A minor MAE value indicates that the model has a greater accuracy and more minor deviations from the actual values.

$$MAE = \frac{1}{n} \sum_{i=1}^n |a_i - p_i| \quad \text{Equation (4.21)}$$

- The RMSE is used to evaluate how accurately a model predicts the behaviour of a dataset. A lower value of RMSE indicates that the model fits the data better. It is beneficial when penalizing more significant errors more heavily than smaller ones is necessary. The RMSE assigns a relatively higher weight to substantial errors, making it an effective measure to assess the overall performance of a model.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (a_i - p_i)^2} \quad \text{Equation (4.22)}$$

- R^2 measures how well a regression model fits the data; its value ranges from 0 to 1. A higher R^2 value means a better fit of the model, and if R^2 equals 1, it indicates that the model perfectly predicts the data.

$$R^2 = 1 - \frac{\sum_{i=1}^n (a_i - p_i)^2}{\sum_{i=1}^n (a_i - \bar{a}_i)^2} \quad \text{Equation (4.23)}$$

- MAPE evaluates the percentage deviation between the actual and predicted values. A smaller MAPE value indicates better predictive quality of the model with more minor errors.

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{(a_i - p_i)}{a_i} \right| \quad \text{Equation (4.24)}$$

From the statical indicator formulas, n is the number of samples in the validation set, a is the actual output, p is the predicted output, and $\bar{a}_i = \frac{1}{n} \sum_{i=1}^n a_i$.

d) Saving the Selected Model

After evaluating and selecting the best deep learning model for time series data forecasting, saving the chosen model for future application is necessary. Saving a model post-training means the user does not have to retrain from scratch when entering new information, analyzing new data, or refining the model hyperparameters. This process can save computational resources and time. This saving process encapsulates the model's architecture, weights, and training configuration into a single file, making integrating the model into a user interface possible. Also, it facilitates the real-time application of the model for forecasting without needing access to the original training environment.

4.4 Graphical User Interface for Cost Forecasting Model

This research develops a graphical user interface (GUI) to store all forecasting-related construction project data in a database and explicitly execute the developed deep learning model for forecasting EAC at the work package level. The interface follows the proposed DAM in Chapter 3 regarding the data saved through the GUI. The GUI development is based on essential objectives, including simplified data entry and management, one-click model deployment and real-time results

visualization. To design the user interface, this study utilizes two pivotal tools: Tkinter and DB Browser for SQLite. Tkinter is Python's most widely used module for creating graphical user interfaces. Complementing Tkinter, DB Browser for SQLite was utilized as the database management tool to store and manage the data entered through the interface. DB Browser offers an accessible means to interact with SQLite databases, providing a seamless storage and retrieval solution. This combination of Tkinter for interface development and DB Browser for data management creates a comprehensive ecosystem, ensuring the user interface is interactive and effectively supports the underlying data infrastructure required for cost forecasting. This interface is designed in alignment with the Entity-Relationship Diagram (ERD) developed in Chapter 3, section 3.4.1, ensuring a coherent and systematic structure. The interface navigation menu shows the main tasks designed for the GUI, and it is categorized into three sections: "Project Setup," "Project Tracking," and "Project Forecasting." Each section is designed to enhance user interaction and data management efficiency. The interface is engineered to verify the format of any new data entered automatically. When the input data is found to be erroneous or incomplete, the system is programmed to notify the user with an informative message. This approach minimizes data entry errors.

4.4.1 Designing GUI Application with Tkinter

In this study, a single class named "ProjectCostManagementApp" is applied in the Tkinter Python code to develop the user interface. This design choice aligns with the principles of object-oriented programming (OOP). OOP implies an instance or class variable (Moore & Harwani, 2019). A class is a blueprint or template that contains methods and variables; it facilitates encapsulation, allowing all the properties and methods necessary for the created class interface to be contained within a unit. This approach streamlines the logical grouping of related functionalities, such as initializing

the application window, connecting to the SQL database, and creating the user input fields. This approach enables manageable code, advantageous in a GUI context where the interface elements are interdependent. Moreover, the class-based architecture allows for inheritance and extension, paving the way for further development without altering the existing, well-defined structure. Therefore, implementing the entire user interface within a single class underscores the study's commitment to producing robust and well-organized code.

Figure 4.9 illustrates an approach to constructing a class in a Tkinter GUI application. It showcases how a well-designed class is the backbone for managing the graphical user interface elements and their associated functionalities. The class begins with an “__init__” method to initialize the main application window, followed by “show_frame” and “clear_frame” methods, integral to the dynamic display and management of different frames or sections within the GUI.

This piece of the Python code of a Tkinter application demonstrates how classes can be used to encapsulate functionalities, ensuring that each code segment is responsible for a particular piece of the application's operation. This approach simplifies the development process, as methods and attributes related to specific actions are categorized within the class structure.

```

class ProjectCostManagementApp:
    def __init__(self, root):
        def show_frame(self, frame_name):
        def clear_frame(self, frame):

    ## CREATE THE NAVIGATION MENU FRAME
    def create_menu_navigation_frame(self):

    ## 1.0 PROJECT SETUP FRAME
    def create_project_setup_frame(self):

    ## 1.1 SAVE PROJECT DATA
    def save_project_data(self):

    ## 1.2 ADD WORK PACKAGE
    def create_wp_setup_window(self):

    ## 1.3 SAVE WORK PACKAGE
    def save_wp_data(self):

    ## 2.0 PROJECT TRACKING
    ## 2.1 CREATE UPDATE PROJECT
    def create_update_project_info(self):
    ## 2.1.1 CURRENT BASELINE
    def create_cb_frame(self):
    def save_cb(self):
    ## 2.1.2 SHOW PROJECT INFORMATION
    def show_project_data(self):
    def edit_project_information(self):
    def update_project_information(self):

    ## 2.2 CREATE UPDATE WORK PACKAGE
    def create_update_wp_info(self):
    def open_update_cb_window(self):
    def save_baseline_data(self):
    def update_wp_names(self, event=None):
    def show_wp_details(self):
    def edit_wp_information(self):

    ## 2.3 WEEKLY PROGRESS
    def create_project_cost_status_frame(self):
    def show_information(self):
    ...

```

Figure 4.9 Code snippet for the Class ProjectCostManagementApp

- **Detailed Menu Navigation**

The navigation menu is critical to the user's interaction with the application, organized by the “create_menu_navigation_frame” method, which suggests a user-friendly approach to navigating through the system's various features. The Python code segment shown in Figure 4.10 provides an example of the "create_menu_navigation_frame" method used to build this navigation framework. This study divides the menu into sections such as 'Project Setup,' 'Project Tracking,' and 'Project Forecasting.' Each section is then populated with buttons (Button widgets) that are styled and configured to perform specific functions indicated by the command parameter.

```

## NAVIGATION MENU

def create_menu_navigation_frame(self):
    self.menu_navigation_frame = tk.LabelFrame(self.root, text="NAVIGATION MENU", font=("Didot", 11, "bold"), foreground="deeppink4",borderwidth=3)
    self.menu_navigation_frame.grid(row=0, column=0, padx=5, pady=20, sticky="nsw")

    ## Project Setup
    project_setup_frame = tk.LabelFrame(self.menu_navigation_frame, text="Project Setup", **self.fonts["labelframe2"])
    project_setup_frame.grid(row=0, padx=10, pady=10, sticky="nsew")

    self.setup_project = tk.Button(project_setup_frame, text="Project Creation", bg="#DCDCDC", command=lambda: self.show_frame("Project Setup"))
    self.setup_project.grid(row=0, padx=30, pady=8, sticky="w")

    ## Project Tracking
    project_tracking_frame = tk.LabelFrame(self.menu_navigation_frame, text="Project Tracking", **self.fonts["labelframe2"])
    project_tracking_frame.grid(row=1, padx=10, pady=10, sticky="nsew")

    self.update_project_info = tk.Button(project_tracking_frame, text="Project Update", bg="#DCDCDC", command=lambda: self.show_frame("Update Project"))
    self.update_project_info.grid(row=0, padx=30, pady=8, sticky="w")

    self.update_wp_info = tk.Button(project_tracking_frame, text="Work Package Update", bg="#DCDCDC", command=lambda: self.show_frame("Update Work Package"))
    self.update_wp_info.grid(row=1, padx=30, pady=8, sticky="w")

    self.project_cost_status_button = tk.Button(project_tracking_frame, text="Weekly Progress", bg="#DCDCDC", command=lambda: self.show_frame("Weekly Progres"))
    self.project_cost_status_button.grid(row=2, padx=30, pady=8, sticky="w")

    ## Project Forecasting
    project_forecasting_frame = tk.LabelFrame(self.menu_navigation_frame, text="Project Forecasting", **self.fonts["labelframe2"])
    project_forecasting_frame.grid(row=2, padx=10, pady=10, sticky="nsew")

    self.project_forecasting = tk.Button(project_forecasting_frame, text="Project Forecasting", bg="#DCDCDC", command=lambda: self.show_frame("Project Forecasting"))
    self.project_forecasting.grid(row=0, padx=30, pady=8, sticky="w")

```

Figure 4.10 Segment Python Code for Creating Navigation Menu

○ Project Setup Section

The Project Settings section has been designed to collect project-related information using multiple input fields and is activated by clicking the Create Project button. This frame design comprises four methods called `def save_project_data (self, def create_project_setup_frame (self, def save_wp_data (self, def create_wp_setup_window (self)`. Each method performs a specific operation, manipulating GUI elements and handling logic to ensure proper application functionality. Figure 4.11 illustrates the methods included in the project setup frame design.

```

## 1.0 PROJECT SETUP FRAME
def create_project_setup_frame(self):

## 1.1 SAVE PROJECT DATA
def save_project_data(self):

## 1.2 ADD WORK PACKAGE
def create_wp_setup_window(self):

## 1.3 SAVE WORK PACKAGE
def save_wp_data(self):

```

Figure 4.11 Methods include in the Project Setup Frame Design

The Following enumerates the elements incorporated within this frame: Project Name, Portfolio Name, Owner Company Name; Contract Type: A dropdown menu with options such as Lump Sum, Time Material, Unit Price, and Cost Plus; Project Type: A dropdown menu with options for Residential & Commercial, Infrastructure, and Industrial projects; Contract Amount, Project Manager, Description, Project ID, Project Cost Center, Contract Number, Owner Contact Name, Project Location, Contract Duration; Project Delivery Method: A dropdown menu with options such as Design-bid-build, Design-Build, IPD, and Construction Management.

Furthermore, the Project Setup Frame encompasses an “Original Baseline” subframe comprising the Budget at Completion, Finish Date, Start Date, and Schedule Baseline. The elements of the project setup frame are displayed in Figure 4.12. If any fields are left unfilled when committing the project setup to the database, an error message will prompt on the interface, as shown in Figure 4.13. This notification delineates the fields that require completion, ensuring data integrity and the coherence of the information stored.

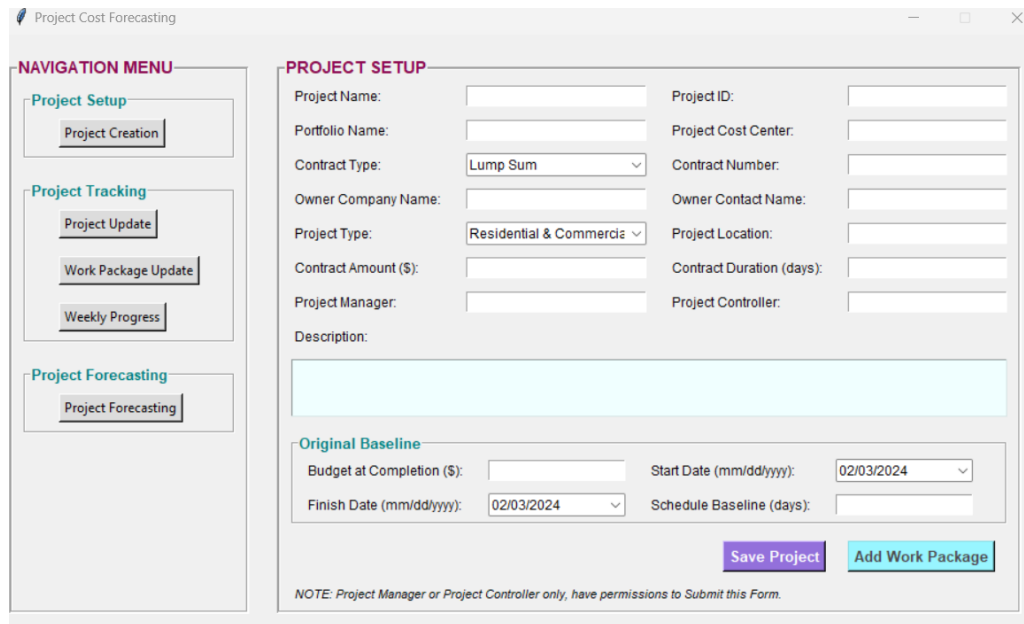


Figure 4.12 Project Setup Frame

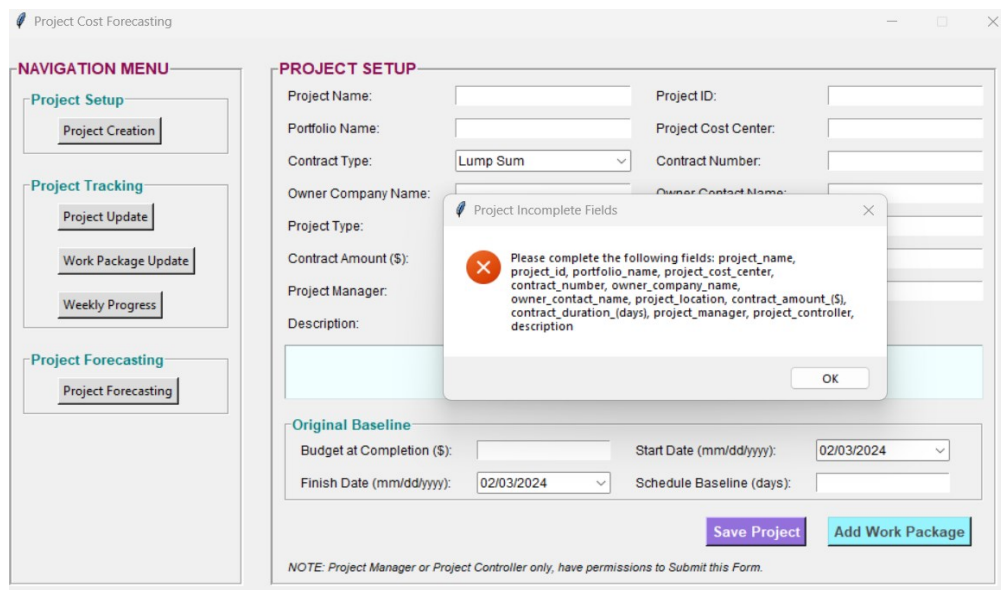


Figure 4.13 Error Message in Project Setup Frame

The Project Setup frame incorporates a button labelled Add Work Package, which, upon interaction, opens a window titled Work Package Setup. This window is designed for the detailed entry of

information about individual work packages that constitute a project. The elements of this window are elucidated in Figure 4.14

Work Package Details

WORK PACKAGE SETUP

Project Name: Tailing Cyclone Facility & Tailing Distribution System

Work Package Name: Concrete

Work Package Type: EWP-Engineering

Unit of Measure: m³

Description:

Work Package Code:

Key Quantity: Volume of Concrete

Original Baseline

Budget at Completion (\$):

Start Date (mm/dd/yyyy): 02/02/2024

Finish Date (mm/dd/yyyy): 02/02/2024

Duration (days):

Labour Cost (\$):

Equipment Cost (\$):

Material Cost (\$):

Subcontractor Cost (\$):

Quantity:

Save Work Package

NOTE: Project Manager or Project Controller only, have permissions to Submit this Form.

Figure 4.14 Work Package Setup Frame

○ Project Tracking Section

The interface's 'Project Tracking' section is segmented into three subsections: 'Project Update,' 'Work Package Update,' and 'Weekly Progress,' and each subsection is activated by clicking the button with the same name. Figure 4.17 displays a portion of a Python code, defined methods within a class designed for project tracking. The methods are categorized under several subsections, each representing a different aspect of the project tracking functionality:

- Update Project: This subsection contains methods for updating project information.
 - `def update_project_info(self):` A method for updating the general project information.
 - `def create_cb_frame(self):` This is a method to create a frame for the current baseline data.

- `def save_cb(self)`: This is likely a method for saving the current baseline data.
- `def show_project_data(self)`: A method to display project data.
- `def edit_project_information(self)`: A method to edit project information.
- `def update_project_information(self)`: A method to update the displayed project information.
- Update Work Package: This part involves methods for updating work package details.
 - `def create_update_wp_info(self)`: Creating an interface to update work package information.
 - `def open_update_cb_window(self)`: A method that might open a window to update current baseline information specific to work packages.
 - `def save_baseline_data(self)`: A method to save baseline data for work packages.
 - `def update_wp_names(self)`: This method could be used to update the names of work packages.
 - `def show_wp_details(self)`: A method for displaying work package details.
 - `def edit_wp_information(self)`: A method to edit work package information.
- Weekly Progress: This subsection lists methods concerning weekly progress information.
 - `def create_project_cost_status_frame(self)`: A method to create a frame for entering or displaying the project's cost status.
 - `def show_information(self)`: A method to show the project's progress information.
 - `def open_actual_cost_window(self)`: A method to open a window for inputting or editing actual cost data.
 - `def save_actual_cost_details(self)`: A method for saving details about actual costs.

- Callbacks for GUI Elements: This part includes callback methods likely triggered by GUI events such as user interactions with widgets.
 - `def update_wp_combobox(self)`: A callback method to update a combo box with work package options.
 - `def update_week_ending_calendar(self)`: A method to update a calendar widget for selecting the end of a week.
 - `def save_weekly_progress_data(self)`: A method to save data related to weekly progress.

Project Update frame empowers the user to select a given project previously created, view the project details, and execute modifications, rectifying errors in previous data entries. After these editions, the corrections are recorded within the SQLite database to maintain data integrity. Furthermore, the frame has a feature that facilitates updating the initial baseline, allowing it to reflect the current baseline metrics. Figure 4.16 shows the Project Update frame, while Figure 4.17 presents the Current Baseline window, which can be accessed by selecting the 'Update Original Baseline' button.

```
## 2.0 PROJECT TRACKING
## 2.1 CREATE UPDATE PROJECT
def create_update_project_info(self):
    ## 2.1.1 CURRENT BASELINE
    def create_cb_frame(self):
    def save_cb(self):
    ## 2.1.2 SHOW PROJECT INFORMATION
    def show_project_data(self):
    def edit_project_information(self):
    def update_project_information(self):

## 2.2 CREATE UPDATE WORK PACKAGE
def create_update_wp_info(self):
def open_update_cb_window(self):
def save_baseline_data(self):
def update_wp_names(self, event=None):
def show_wp_details(self):
def edit_wp_information(self):

## 2.3 WEEKLY PROGRESS
def create_project_cost_status_frame(self):
def show_information(self):
def open_actual_cost_window(self):
def save_actual_cost_details(self):

# Callback function to update Work Package combobox
def update_wp_combobox(self, event=None):
def update_week_ending_calendar(self, event=None):
def save_weekly_progress_data(self):
```

Figure 4.15 Method Structure for Project Tracking in Tkinter GUI Application

Project Cost Forecasting

NAVIGATION MENU

Project Setup

Project Creation

Project Tracking

Project Update

Work Package Update

Weekly Progress

Project Forecasting

Project Forecasting

PROJECT UPDATE

Project Name: Tailing Cyclone Facility & Tailor Show Project Information Update Original Baseline

Project Information

Project Name: Tailing Cyclone Facility & Tailor Project ID: 002

Portfolio Name: Mining Infrastructure Projects Project Cost Center: ABC Company Peru, Proce

Contract Type: Unit Price Contract Number: 2014-MIP

Owner Company Name: Mining Company Owner Contact Name: Luis Carruitero

Project Type: Infrastructure Project Location: Arequipa, Peru

Contract Amount (\$): 6526530.0 Contract Duration (days): 360

Project Manager: Nestro Cruzado Project Controller: Lucia Perez

Description:

Construction of an advanced Tailing Cyclone Facility and comprehensive Tailing Distribution System for Phase VB in the mining installation.

Original Baseline

Budget at Completion (\$): 6526530.0 Start Date (mm/dd/yyyy): 01/08/2014

Finish Date (mm/dd/yyyy): 01/08/2024 Schedule Baseline (days): 385

Edit Project Information Update Project Information

Figure 4.16 Project Update Frame

Project Cost Forecasting

NAVIGATION MENU

Project Setup

Project Creation

Project Tracking

Project Update

Work Package Update

Weekly Progress

Project Forecasting

Project Forecasting

PROJECT UPDATE

Project Name: Tailing Cyclone Facility & Tailor Show Project Information Update Original Baseline

Project Information

Current Baseline

Project Name: Tailing Cyclone Facility & Tailing Distribution System Project ID: 002

Current Baseline

Budget at Completion (\$): 6526530.0

Start Date (mm/dd/yyyy): 01/08/2014

Finish Date (mm/dd/yyyy): 01/08/2024

Schedule Baseline (days): 385

Save Current Baseline

Original Baseline

Budget at Completion (\$): 6526530.0 Start Date (mm/dd/yyyy): 01/08/2014

Finish Date (mm/dd/yyyy): 01/08/2024 Schedule Baseline (days): 385

Edit Project Information Update Project Information

Figure 4.17 Current Baseline Window

The following frame is designated for the work package update. Within this frame, the user can select both the 'Project Name' and the 'Work Package Name' from a drop-down menu. Subsequent interaction with the 'Show Information' button reveals the details about the chosen work package. This interface allows editing the work package data, facilitating the correction previously recorded, if necessary. Figure 4.18 depicts the Work Package Update Frame and this interface segment's elements.

Project Cost Forecasting

NAVIGATION MENU

- Project Setup**
 - Project Creation
- Project Tracking**
 - Project Update
 - Work Package Update
 - Weekly Progress
- Project Forecasting**
 - Project Forecasting

WORK PACKAGE UPDATE

Project Name: Tailing Cyclone Facility & Show WP Information

Work Package Name: Concrete Update WP Original Baseline

Work Package Information

Work Package Name: Concrete Work Package Code: WP-C02

Work Package Type: CWP-Construction Key Quantity: Volume of Concrete

Unit of Measure: m³ Description:

Original Baseline

Budget at Completion (\$): 2560152.4 Start Date (mm/dd/yyyy): 11/27/2015

Finish Date (mm/dd/yyyy): 12/16/2016 Duration (days): 385

Labour Cost (\$): 1356541.79 Equipment Cost (\$): 147710.19

Material Cost (\$): 1055900.42 Subcontractor Cost (\$): 0.0

Quantity: 3816.59

Edit WP Information Update WP Information

Figure 4.18 Work Package Update Frame

An additional feature of this frame is the 'Update WP Original Baseline' button, which establishes a current baseline window; this window allows the user to update the original baseline to the selected work package. Figure 4.19 shows the work package's current baseline window.

Project Cost Forecasting

NAVIGATION MENU

Project Setup

Project Creation

Project Tracking

Project Update

Work Package

Weekly Progress

Project Forecasting

Project Forecast

Update Original Baseline WP

Project Name: Tailing Cyclone Facility & Tailing Distribution System

Project ID: 002

Work Package Name: Concrete

Work Package Code: WP-C02

WP Current Baseline

Budget at Completion (\$): 2567283.33

Start Date (mm/dd/yyyy): 10/27/2014

Finish Date (mm/dd/yyyy): 12/16/2015

Duration (days): 385

Labour Cost (\$): 1356541.79

Equipment Cost (\$): 147710.19

Material Cost (\$): 1055900.42

Subcontractor Cost (\$): 0.0

Quantity: 3816.59

Save WP Current Baseline

WP Information

Original Baseline

02

WP Name of Concrete

10/27/2015

147710.19

Edit WP Information

Update WP Information

Figure 4.19 Work Package Current Baseline Window

The 'Weekly Progress per Work Package' frame facilitates the systematic entry of progress-related information for each work package within a project, which is essential for generating the inputs necessary for the forecasting model. Initially, users must select specific parameters: the project and work package name, the status upon completion, the period number, and the evaluation day—the latter signifying the concluding day of the week under review for forecasting purposes.

Upon selecting 'Show Information,' the interface displays two sub-frames: one detailing baseline information and the other showing weekly progress data. The 'Weekly Progress Information' sub-frame is comprised of four elements—Earned Value (\$), Planned Value (\$), Actual Cost (\$), and Quantity. Users must input data for these elements to calculate the current cumulative values, which is facilitated by activating the 'Current Cumulative' button. Furthermore, a comment field must be filled with information about the current period. This should include insights that facilitate understanding any critical situations that may have arisen.

Subsequently, clicking the blue 'Actual Cost' button opens a new window titled 'Actual Cost Detail.' This window is designed to enter detailed actual cost data, including labour, equipment, materials, and expenditures for subcontractor services.

To conclude the process, users should click the save buttons to ensure the information for the current week is accurately recorded and preserved. Figure 4.20 describes the weekly progress per work package window.

The screenshot shows a web application window titled 'Project Cost Forecasting'. On the left is a 'NAVIGATION MENU' with three sections: 'Project Setup' (containing 'Project Creation'), 'Project Tracking' (containing 'Project Update', 'Work Package Update', and 'Weekly Progress'), and 'Project Forecasting' (containing 'Project Forecasting'). The 'Weekly Progress' option is selected. The main area is titled 'WEEKLY PROGRESS PER WORK PACKAGE' and contains the following fields and sections:

- Form Fields:**
 - Project Name: Tailing Cyclone Facility & (dropdown)
 - Work Package Name: Concrete (dropdown)
 - Period Number: (text input)
 - Week Ending (mm/dd/yyyy): 12/12/2015 (dropdown)
 - Status at Completion: In Progress (dropdown)
 - Show Information (button)
- Baselines Information Table:**

	Original Baseline	Current Baseline
Budget at Completion (\$):	2560152.4	2567283.33
Duration (days):	385	385
Quantity:	3816.59	3816.59
- Weekly Progress Information Table:**

	Previous Cumulative 12/09/2015	Current Week	Current Cumulative
Earned Value (\$):	2560152.4		2560152.4
Planned Value (\$):	0.0		0.0
Actual Cost (\$):	2717205.38		2717205.38
Quantity:	3816.59		3816.59
Comments:			
- Buttons:** Save Progress (button)
- Footnote:** NOTE: Project Manager or Project Controller only, have permissions to Submit this Form.

Figure 4.20 Weekly Progress per Work Package Frame

4.5 Integrating SQL into GUI

When incorporating SQLite into a Python-based application, it is crucial to import the SQLite3 library, which is a standard library component of Python. This module is necessary to connect with SQLite databases and perform SQL operations. DB Browser for SQLite is used in this study. DB Browser is an open-source tool for constructing and designing database files compatible with SQLite. To design the database, it is essential to define the schema, which includes tables, fields, data types, and execute SQL commands to materialize it. After creating the database, it is necessary to connect it to the interface and write functions in Python to perform operations such as design, read, update, and delete. Finally, it links GUI actions to database functions. Figure 4.23 reveals a portion of the database schema and the structure of several tables, indicating their names, data types, and the SQL commands used to create them.

Name	Type	Schema
Tables (7)		
Project_current_baselines		
CREATE TABLE "Project_current_baselines" ("budget_at_completion_(\$)" REAL,	"start_date_(mm/dd/yyyy)" TEXT, "finish_date_(mm/dd/yyyy)" TEXT, "schedule_baseline_(days)" TEXT, "project_id" TEXT, "mod_date" TEXT
budget_at_completion_(\$)	REAL	"budget_at_completion_(\$)" REAL
start_date_(mm/dd/yyyy)	TEXT	"start_date_(mm/dd/yyyy)" TEXT
finish_date_(mm/dd/yyyy)	TEXT	"finish_date_(mm/dd/yyyy)" TEXT
schedule_baseline_(days)	TEXT	"schedule_baseline_(days)" TEXT
project_id	TEXT	"project_id" TEXT
mod_date	TEXT	"mod_date" TEXT
Project_original_baselines		
CREATE TABLE "Project_original_baselines" ("budget_at_completion_(\$)" REAL,	"start_date_(mm/dd/yyyy)" TEXT, "finish_date_(mm/dd/yyyy)" TEXT, "schedule_baseline_(days)" INTEGER, "project_id" TEXT, "mod_date" TEXT
budget_at_completion_(\$)	REAL	"budget_at_completion_(\$)" REAL
start_date_(mm/dd/yyyy)	TEXT	"start_date_(mm/dd/yyyy)" TEXT
finish_date_(mm/dd/yyyy)	TEXT	"finish_date_(mm/dd/yyyy)" TEXT
schedule_baseline_(days)	INTEGER	"schedule_baseline_(days)" INTEGER
project_id	TEXT	"project_id" TEXT
mod_date	TEXT	"mod_date" TEXT
Projects		
CREATE TABLE "Projects" ("portfolio_name" TEXT, "project_name" TEXT, "project_id" TEXT, "project_cost_center" TEXT, "contract_type" TEXT, "contract_number" TEXT, "owner_company_name" TEXT, "owner_contact_name" TEXT, "project_type" TEXT, "project_location" TEXT, "contract_amount_(\$)" REAL, "contract_duration_(days)" INTEGER, "project_manager" TEXT, "project_controller" TEXT, "description" TEXT	
portfolio_name	TEXT	"portfolio_name" TEXT
project_name	TEXT	"project_name" TEXT
project_id	TEXT	"project_id" TEXT
project_cost_center	TEXT	"project_cost_center" TEXT
contract_type	TEXT	"contract_type" TEXT
contract_number	TEXT	"contract_number" TEXT
owner_company_name	TEXT	"owner_company_name" TEXT
owner_contact_name	TEXT	"owner_contact_name" TEXT
project_type	TEXT	"project_type" TEXT
project_location	TEXT	"project_location" TEXT
contract_amount_(\$)	REAL	"contract_amount_(\$)" REAL
contract_duration_(days)	INTEGER	"contract_duration_(days)" INTEGER
project_manager	TEXT	"project_manager" TEXT
project_controller	TEXT	"project_controller" TEXT
description	TEXT	"description" TEXT

Figure 4.23 Database Structure

4.6 Project Forecasting Section

The code snippet depicted in Figure 4.24 includes a set of method definitions related to project forecasting functionality in a Python class. These methods collectively outline steps where data is collected, processed, and used to predict outcomes. The results are then presented in a graphical user interface. In an application built with Tkinter, these methods would be connected to various widgets such as buttons, entries, and plot areas. This provides an interactive way for users to perform and view project forecasts at the work package level.

```
## PROJEC FORECASTING

def create_project_forecasting_frame(self):
def predict(self):
def series_to_supervised(self, data, n_in=1, n_out=1, dropnan=True):
def prepare_data_for_prediction(self, data, n_past, n_features):
def make_prediction(self, prepared_data, scaler, actual_data=None):
def collect_prediction_data(self):
def display_prediction(self):
def graphic_budget_status(self, data_graphic):
```

Figure 4.24 Methods Developed for Cost Forecasting

Figure 4.25 illustrates the Cost Status graph for a Project related to a specific work package. It delineates three curves: Actual Cost, Earned Value, and Planned Value. The project's temporal progression is demarcated weekly. It also shows the predicted EAC, which is updated weekly as the project progresses. The graph serves as a tool, offering a visual compendium of cost forecasting. This frame also shows a tabular representation of cost data, including period number, week ending, original budget at completion, current budget at completion, earned value cumulative, actual cost cumulative, historical EAC's predictions, and variance (%), which can indicate whether the project is over or under cost respected the current baseline.

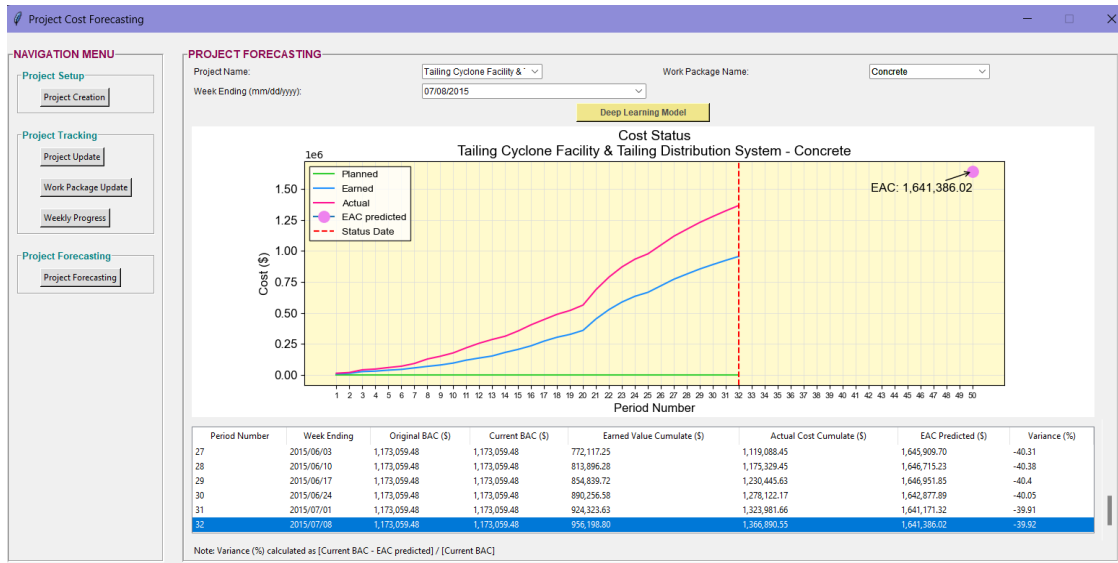


Figure 4.25 Project Forecasting Frame

Chapter 5: Application, Verification and Validation

5.1 Introduction

This chapter demonstrates the implementation of the developed computational models using real datasets, building on the foundational groundwork in the previous chapter. This chapter starts by determining the best model from three algorithms based on their performance metrics outcomes. Once the optimal model has been selected, this model is deployed using a testing dataset _data that remained unseen during the training and validation phases. The results from the testing data are compared against popular methodologies derived from Earned Value Management (EVM). To verify the model's reliability, a comprehensive sensitivity analysis examines how the model responds to changes in input parameters.

5.2 Selecting the Optimal Forecasting Model.

This section focuses on selecting the best forecasting model using the validation dataset. It compares the performance metrics of three key algorithms: Simple Recurrent Neural Network (SimpleRNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). The criteria for selection are based on an evaluation of key performance metrics, which include Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). These metrics comprehensively overview each model's accuracy, reliability, and predictive power. This helps make an informed decision about the most suitable algorithm for accurate cost forecasting.

Tables 5.1 and 5.2 display the performance metrics for the EAC and ETC, respectively. The data is from three deep learning models: Simple RNN, LSTM, and GRU. The models are categorized by work package (Concrete, Piping (HDPE), and Fill). According to the results from Tables 5.1

and 5.2, the GRU algorithm shows the lowest values for MAE, RMSE, and MAPE, suggesting that the GRU model is the most reliable and accurate approach among the model validation processes, making it a suitable choice for time-series predictions.

Table 5.1 Model Results Comparison ETC Results

Model	Validation Set (ETC)			
	RMSE	MAE	MAPE	Ranking
Concrete_SimpleRNN	42,016.58	26,088.95	10.63%	3
Concrete_LSTM	34,323.19	24,150.55	11.45%	2
Concrete_GRU	31,080.09	21,098.74	10.83%	1
Fill_SimpleRNN	28,522.80	24,745.52	12.84%	2
Fill_LSTM	56,658.67	45,069.00	16.45%	3
Fill_GRU	30,291.21	23,226.90	8.60%	1
Piping_SimpleRNN	38,550.14	30,535.96	4.39%	2
Piping_LSTM	54,372.21	45,299.86	5.07%	3
Piping_GRU	33,592.51	24,252.65	4.96%	1

Table 5.2 Model Results Comparison EAC Results

Model	Validation Set (EAC)			
	RMSE	MAE	MAPE	Ranking
Concrete_SimpleRNN	42,016.58	26,088.95	1.22%	3
Concrete_LSTM	34,323.19	24,150.55	1.13%	2
Concrete_GRU	31,080.09	21,098.74	0.99%	1
Fill_SimpleRNN	28,522.80	24,745.52	1.61%	3
Fill_LSTM	56,658.67	45,069.00	2.99%	2
Fill_GRU	30,291.21	23,226.90	1.56%	1
Piping_SimpleRNN	38,550.14	30,535.96	0.57%	2
Piping_LSTM	54,372.21	45,299.86	0.87%	3
Piping_GRU	33,592.51	24,252.65	0.44%	1

To aid in understanding the results presented in Tables 5.1 and 5.2, Table 5.3 compares actual and predicted ETC and EAC for the validation dataset (concrete work package). The table presents predicted values for each period, starting from period four. Regarding the ETC predicted values,

the % error tends to be higher in the last periods, given that the predicted values decrease to almost zero. Hence, minimal variation between actual and predicted generates a high % error. To better understand the algorithm's performance, the actual cost to date is added to the predicted ETC in each period to obtain the predicted EAC. In this scenario, the scale values are higher, and the % error between the actual and predicted values is more representative. Furthermore, Figure 5.1 displays the graphical representation of EAC results for the concrete work package on the validation dataset; this figure shows that between the LSTM, GRU, and Simple RNN results, the GRU set of results (red line) is closer to the actual cost at completion (ACC) represented by the blue line. ACC refers to the total cost incurred to complete a project from start to finish. It is a critical metric for assessing project performance and comparing it against the initial budget and the EAC. It helps project managers evaluate the accuracy of cost forecasts. This figure suggests that the GRU model has higher accuracy for the cost forecasting task. The x-axis represents the work package periods in weeks, while the y-axis represents the cost in dollars.

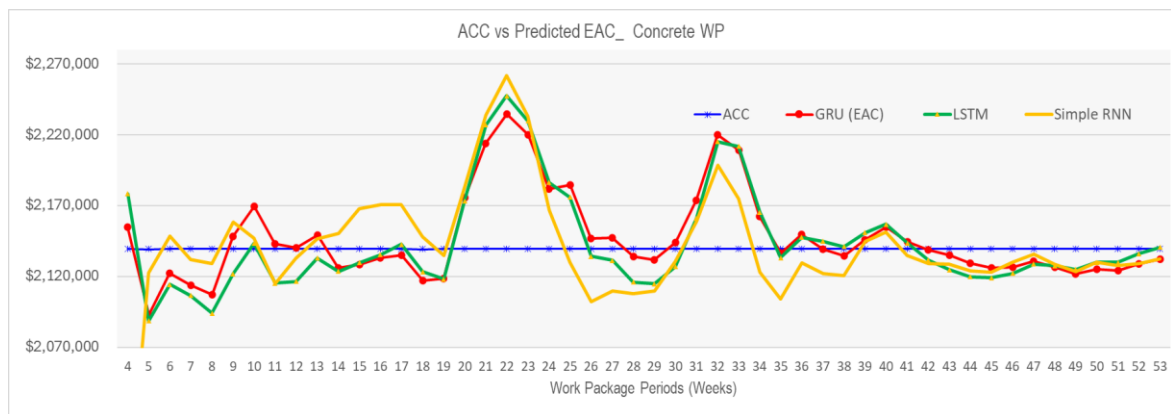


Figure 5.1 ACC vs Predicted EAC (Simple RNN, LSTM and GRU)

Table 5.3 Results of ACC vs. Deep Learning Models

ACC	GRU (EAC)	LSTM	Simple RNN
2,139,517.75	2,154,796.75	2,178,442.75	1,953,154.55
2,139,517.73	2,093,067.93	2,088,537.73	2,122,540.73
2,139,517.84	2,122,309.44	2,114,426.74	2,148,319.44
2,139,517.88	2,113,948.08	2,106,566.98	2,132,174.78
2,139,517.82	2,107,303.52	2,094,010.62	2,129,299.02
2,139,517.75	2,148,228.25	2,121,932.95	2,158,631.35
2,139,517.82	2,169,604.82	2,143,513.12	2,146,988.32
2,139,517.77	2,142,924.97	2,115,283.47	2,114,800.27
2,139,517.83	2,140,225.03	2,116,625.93	2,132,809.83
2,139,517.77	2,149,348.27	2,133,101.67	2,146,537.37
2,139,517.74	2,126,281.24	2,123,526.64	2,150,353.14
2,139,517.76	2,128,323.96	2,129,692.26	2,168,125.06
2,139,517.80	2,133,192.40	2,135,361.80	2,170,723.70
2,139,517.84	2,134,850.24	2,142,674.04	2,170,660.04
2,139,517.70	2,117,319.60	2,123,480.70	2,148,254.30
2,139,517.86	2,118,413.26	2,118,192.66	2,134,590.56
2,139,517.81	2,175,880.21	2,173,461.81	2,183,033.31
2,139,517.76	2,213,909.16	2,227,201.16	2,234,162.46
2,139,517.84	2,234,607.24	2,247,766.34	2,261,844.24
2,139,517.77	2,220,035.67	2,229,830.07	2,233,208.97
2,139,517.77	2,181,721.17	2,186,523.27	2,167,150.97
2,139,517.77	2,184,681.46	2,176,073.58	2,130,143.12
2,139,517.82	2,146,721.17	2,134,250.17	2,102,385.82
2,139,517.78	2,147,481.62	2,131,454.02	2,109,664.42
2,139,517.77	2,134,159.87	2,115,957.57	2,107,732.72
2,139,517.81	2,131,911.55	2,114,951.36	2,109,910.21
2,139,517.76	2,144,009.46	2,127,012.86	2,130,440.56
2,139,517.79	2,173,890.71	2,161,262.41	2,158,817.16
2,139,517.81	2,219,915.01	2,215,370.99	2,198,771.17
2,139,517.79	2,209,291.95	2,211,925.61	2,175,086.23
2,139,517.80	2,162,633.26	2,165,335.99	2,122,938.36
2,139,517.81	2,136,494.15	2,133,386.90	2,104,061.23
2,139,517.79	2,149,595.79	2,147,614.85	2,129,851.07
2,139,517.80	2,139,494.16	2,144,793.33	2,121,992.75
2,139,517.80	2,134,624.27	2,141,091.17	2,120,690.56
2,139,517.79	2,146,148.88	2,151,092.14	2,144,609.42
2,139,517.80	2,154,694.65	2,156,904.03	2,151,453.35
2,139,517.79	2,144,367.90	2,143,764.71	2,135,048.07
2,139,517.80	2,138,913.39	2,131,532.50	2,129,319.86
2,139,517.80	2,135,232.42	2,124,790.43	2,128,719.24
2,139,517.80	2,129,383.55	2,119,802.94	2,123,854.64
2,139,517.80	2,125,935.39	2,119,161.32	2,122,967.26
2,139,517.80	2,126,604.65	2,122,269.56	2,130,206.55
2,139,517.80	2,130,800.93	2,128,852.99	2,135,623.54
2,139,517.80	2,126,544.90	2,127,576.14	2,128,556.67
2,139,517.80	2,121,789.04	2,124,915.86	2,123,611.24
2,139,517.80	2,125,353.05	2,129,990.28	2,130,111.69
2,139,517.80	2,124,376.79	2,130,083.79	2,127,824.15
2,139,517.80	2,129,044.45	2,136,240.08	2,129,022.37
2,139,517.80	2,132,175.76	2,140,677.33	2,132,450.47
MAE	21,098.74	24,150.55	21,746.30
MAPE	0.99%	1.13%	1.22%
RMSE	31,080.09	34,323.19	42,016.58

5.3 Applying the Forecasting Model to the Testing Dataset

In this section, the selected forecasting models are subjected to an essential test assessing their predictive capability on an unseen dataset. This step is crucial as it reveals the model's predictive power and generalizability beyond the data it was trained. In supervised learning, a model is typically configured to use an input feature set to forecast a subsequent target value. These features comprise historical outputs as part of their composition. About the forecasting model proposed in this study, the initiation phase requires input data from the three antecedent periods, including historical outputs. This input data comprises Actual Cost Cumulative, Earned Value Cumulative, Quantity Cumulative, Percentage of Project Duration Completed, Labour Cost Cumulative, and ETC (output). The methodology of this study incorporates the ETC values derived from an EVM calculation as part of the initial input set. Then, the predicted ETC will serve as a feedback loop for the upcoming periods, refining the forecasting precision for future periods.

The results of this application are depicted through two graphical representations for each work package. First, the ETC (actual vs. predicted) graph compares the actual data and the predictions made by the GRU model across each work package period. Second is the EAC (actual vs. predicted) graph, where the predicted EAC is calculated by adding the actual cost to date to each ETC predicted periodically.

The ETC output is generated through the weekly data input, mirroring the conditions of a real project scenario from the fourth week until the last project week. Subsequently, this output is stored and compared with the actual estimates to complete, facilitating a comprehensive analysis of project forecasting accuracy. The EAC (actual vs. predicted) graph analyzes the actual cost estimate at completion value versus the predicted estimate at completion over time. This graph

demonstrates the long-term accuracy of the model, highlighting the GRU’s ability to capture the overall trend of the project's cost requirements.

Applying these models to the testing dataset provides valuable insights into their effectiveness and reliability. When the predicted and actual values are close, it confirms that the models are robust and have the potential to be helpful in real-world project forecasting scenarios.

- Forecasting Model on the Concrete Dataset**

This section applies the Concrete-GRU forecasting model to the concrete dataset, comprising unseen data during the model's training and validation phases, i.e. testing data set. The analyzed dataset consists of fifty weeks, with results starting from the fourth period due to the three previous periods' data needed for the model's initialization. Figures 5.2 and 5.3 show the result between actual and predicted values for the ETC and EAC calculation.

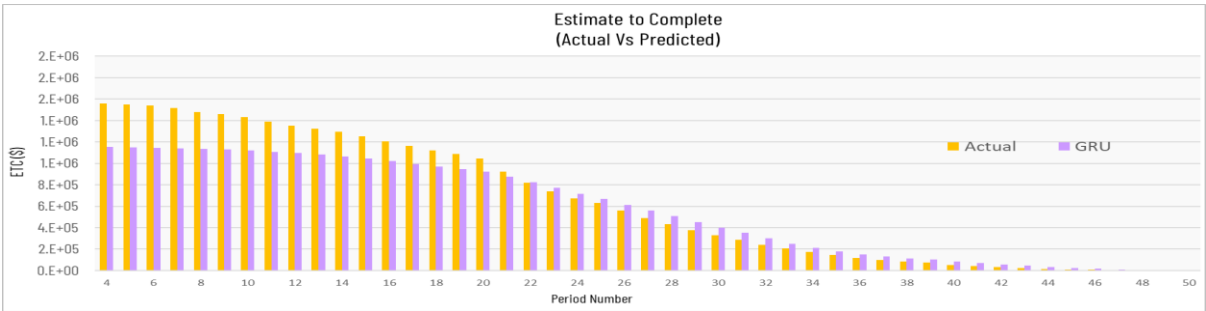


Figure 5.2 Actual vs. Predicted ETC for the Concrete Work Package

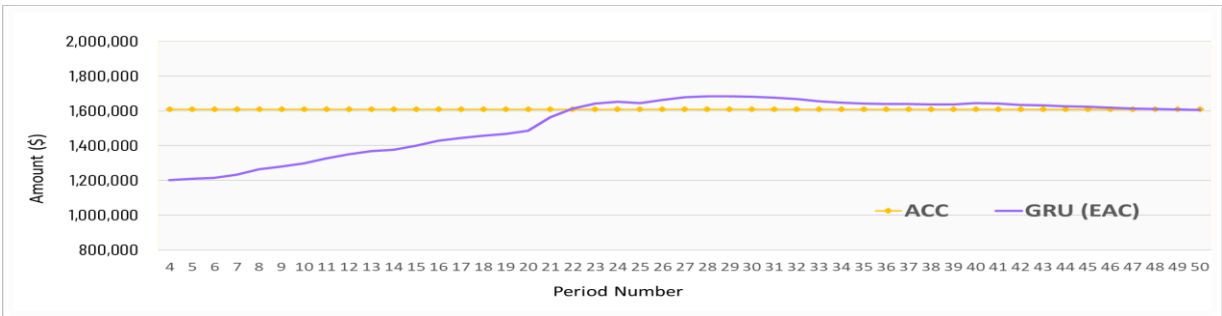


Figure 5.3 ACC vs. Predicted EAC for the Concrete Work Package

Table 5.4 displays the outcomes of applying the Concrete-GRU model on the testing dataset. It enumerates the predicted amount for ETC and EAC. Additionally, the table quantifies the model's accuracy by presenting the percentage error for each evaluated period. The MAE is reported as \$ 118,789, and the MAPE is 7.38%. Figures 5.2, 5.3 and Table 5.4 show that the difference between actual and predicted values are higher during the initial weeks, at around 25%. However, it decreases notably over the subsequent periods as more actual project data is entered into the model, leading to improved precision in cost forecasting efforts over time.

Table 5.4 Results of Concrete-GRU on Testing Data

Period	Work Package	Testing Set				
		ETC (\$)		ACC (\$)	GRU - EAC (\$)	% Error GRU
		Actual	GRU			
4	Concrete	1,561,490.47	1,153,693.75	1,609,043.24	1,201,246.52	25.34%
5	Concrete	1,549,276.47	1,148,860.00	1,609,043.24	1,208,626.77	24.89%
6	Concrete	1,538,577.85	1,144,198.88	1,609,043.24	1,214,664.26	24.51%
7	Concrete	1,516,877.54	1,140,845.63	1,609,043.24	1,233,011.32	23.37%
8	Concrete	1,480,902.31	1,134,778.00	1,609,043.24	1,262,918.92	21.51%
9	Concrete	1,458,457.82	1,128,217.50	1,609,043.24	1,278,802.91	20.52%
10	Concrete	1,431,382.52	1,119,382.63	1,609,043.24	1,297,043.35	19.39%
11	Concrete	1,390,327.22	1,108,038.63	1,609,043.24	1,326,754.64	17.54%
12	Concrete	1,353,371.52	1,095,330.75	1,609,043.24	1,351,002.47	16.04%
13	Concrete	1,322,851.20	1,081,210.00	1,609,043.24	1,367,402.04	15.02%
14	Concrete	1,296,704.16	1,063,130.13	1,609,043.24	1,375,469.20	14.52%
15	Concrete	1,254,318.10	1,044,265.81	1,609,043.24	1,398,990.95	13.05%
16	Concrete	1,204,567.87	1,022,598.00	1,609,043.24	1,427,073.36	11.31%
17	Concrete	1,161,985.56	996,081.19	1,609,043.24	1,443,138.86	10.31%
18	Concrete	1,119,888.18	968,463.44	1,609,043.24	1,457,618.49	9.41%
19	Concrete	1,088,960.83	947,265.75	1,609,043.24	1,467,348.15	8.81%
20	Concrete	1,045,516.53	922,673.81	1,609,043.24	1,486,200.52	7.63%
21	Concrete	920,830.69	874,725.50	1,609,043.24	1,562,938.05	2.87%
22	Concrete	819,725.38	825,213.63	1,609,043.24	1,614,531.48	0.34%
23	Concrete	737,652.15	770,540.25	1,609,043.24	1,641,931.34	2.04%
24	Concrete	675,248.09	717,731.56	1,609,043.24	1,651,526.71	2.64%
25	Concrete	632,271.78	668,746.75	1,609,043.24	1,645,518.20	2.27%
26	Concrete	561,180.78	613,741.94	1,609,043.24	1,661,604.39	3.27%
27	Concrete	489,954.78	559,570.13	1,609,043.24	1,678,658.58	4.33%
28	Concrete	433,713.78	508,236.47	1,609,043.24	1,683,565.92	4.63%
29	Concrete	378,597.60	453,429.56	1,609,043.24	1,683,875.20	4.65%
30	Concrete	330,921.06	401,698.59	1,609,043.24	1,679,820.77	4.40%
31	Concrete	285,061.58	351,148.69	1,609,043.24	1,675,130.35	4.11%
32	Concrete	242,152.69	301,371.78	1,609,043.24	1,668,262.33	3.68%
33	Concrete	205,465.10	251,198.64	1,609,043.24	1,654,776.78	2.84%
34	Concrete	176,115.03	213,033.30	1,609,043.24	1,645,961.51	2.29%
35	Concrete	144,930.58	178,344.08	1,609,043.24	1,642,456.74	2.08%
36	Concrete	119,669.99	150,959.34	1,609,043.24	1,640,332.59	1.94%
37	Concrete	99,711.94	129,831.07	1,609,043.24	1,639,162.37	1.87%
38	Concrete	85,741.30	114,464.03	1,609,043.24	1,637,765.97	1.79%
39	Concrete	73,028.02	101,013.27	1,609,043.24	1,637,028.49	1.74%
40	Concrete	50,074.18	85,568.80	1,609,043.24	1,644,537.85	2.21%
41	Concrete	40,889.89	72,484.07	1,609,043.24	1,640,637.42	1.96%
42	Concrete	33,688.12	57,784.17	1,609,043.24	1,633,139.29	1.50%
43	Concrete	25,337.54	46,587.79	1,609,043.24	1,630,293.48	1.32%
44	Concrete	16,022.16	34,019.42	1,609,043.24	1,627,040.50	1.12%
45	Concrete	9,701.00	23,425.05	1,609,043.24	1,622,767.29	0.85%
46	Concrete	7,837.31	16,315.38	1,609,043.24	1,617,521.30	0.53%
47	Concrete	5,698.80	9,854.60	1,609,043.24	1,613,199.04	0.26%
48	Concrete	3,603.06	4,536.61	1,609,043.24	1,609,976.78	0.06%
49	Concrete	1,602.75	837.71	1,609,043.24	1,608,278.20	0.05%
50	Concrete	-	2,929.96	1,609,043.24	1,606,113.27	0.18%
				MAE		\$ 118,789
				MAPE		7.38%

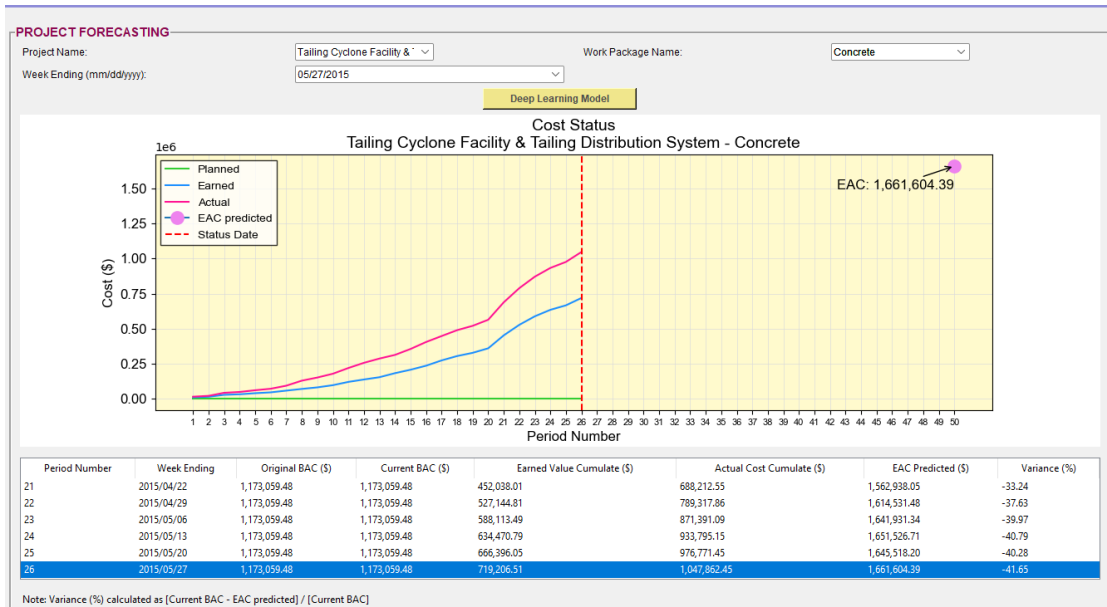


Figure 5.4 Cost Forecasting for the Concrete Work Package - Period 26

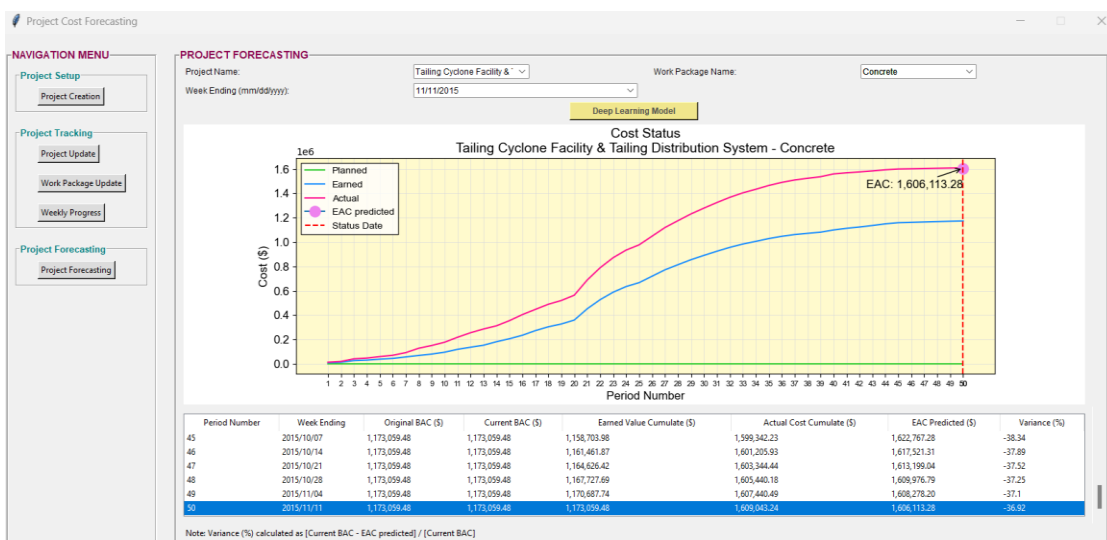


Figure 5.5 Cost Forecasting for the Concrete Work Package - Period 50

Figures 5.4 and 5.5 depict the GUI's deep learning model deployment outcomes tailored for the concrete work package covering periods 26 and 50, respectively. The GUI deploys the forecasting model according to the entered data and provides the project manager with a tool for making decisions during the project execution. To deploy the forecasting model through the GUI, the user

should select the project name, work package, and period to be evaluated, then click the bottom “Deep learning model” to display the actual, planned, and earned value cumulate, as well as the predicted EAC for the selecting period. The green line within the graphical representation denotes the cumulative planned value. As mentioned in Chapter 3, the planned values were not periodically provided among the collected data for this study, so they are shown as zero throughout the periods. However, this research considers it necessary to include this value between the graphical and tabular representations when implementing the GUI with new project data. This inclusion will provide a more complete view of the project results.

The GUI also shows a record table of the previous cost-related data entered, including the predicted EAC for the previous periods and the % variance between the current budget and EAC forecasting. This historical data from the ongoing project helps the project manager periodically observe how the predicted EAC changes over time based on the decisions made to mitigate the percentage variance.

As listed in Table 5.4, the actual estimate at completion versus the predicted estimate at completion for period 50 revealed \$1,609,043.24 and \$1,606,113.28, respectively. Similarly, for period 26, the actual estimate at completion versus the predicted estimate at completion was \$1,609,043.24 and \$1,661,604.39. These predicted EAC values correspond to the EAC results shown in Figures 5.4 and 5.5.

- **Forecasting Model on the Piping Dataset**

This section applies the Piping-GRU model. This testing dataset consists of 45 periods, with results starting from the fourth period. Figure 5.6 shows the result between actual ETC and predicted ETC, respectively. Figure 5.7 illustrates the actual cost at completion (ACC) and the predicted EAC.

Table 5.5 exhibits the results of the Piping-GRU model. This model's MAE is \$ 278,917, and MAPE is 3.997%. MAPE values decrease notably as more actual project data enters the model.

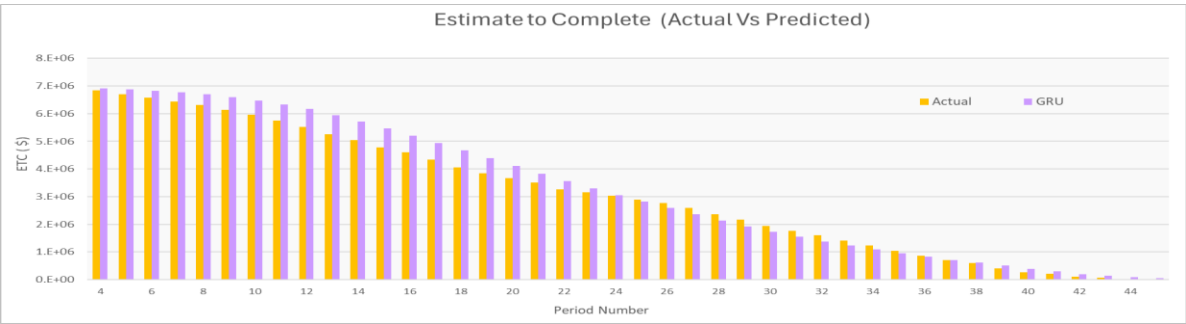


Figure 5.6 Actual vs. Predicted ETC for the Piping Work Package

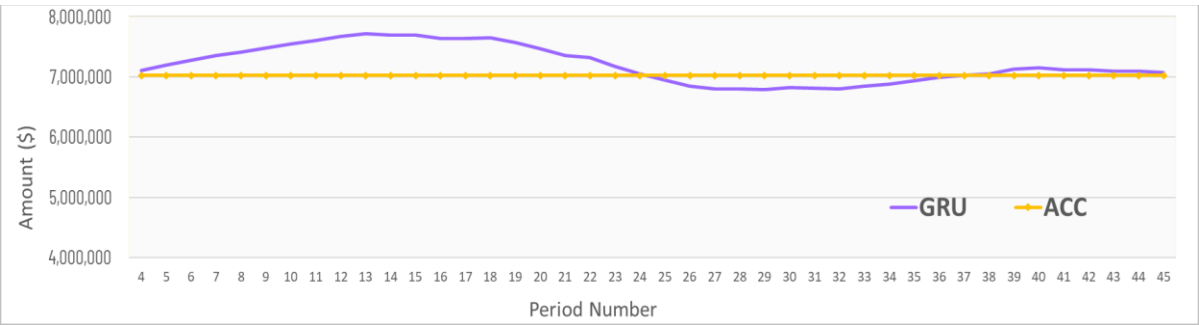


Figure 5.7 ACC vs. Predicted EAC for the Piping Work Package

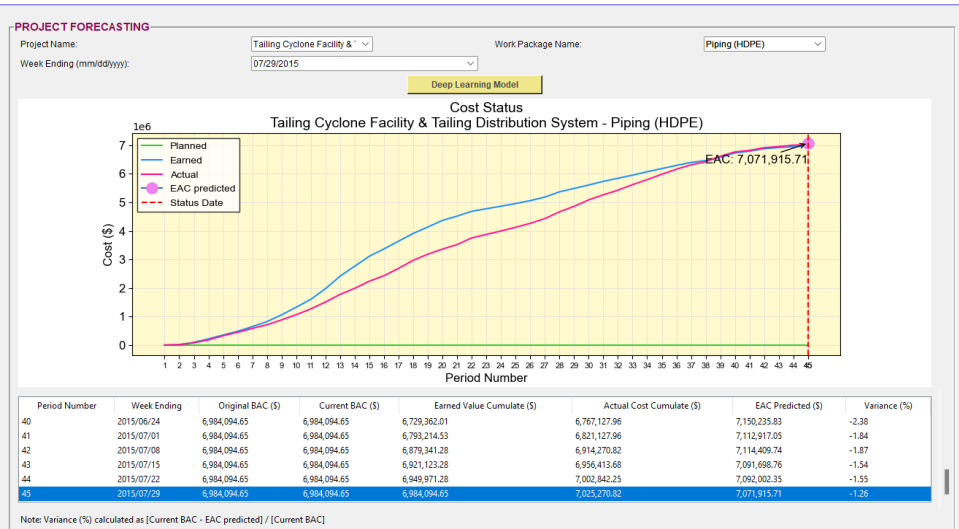


Figure 5.8 Cost Forecasting for the Piping Work Package - Period 45

Figure 5.8 explains GUI's deep learning model deployment outcomes tailored for the piping work package covering period 45.

Table 5.5 Results of Piping-GRU on Testing Data

Testing Set						
Period	Work Package	ETC (\$)		ACC (\$)	GRU - EAC (\$)	% Error
		Actual	GRU			GRU
4	Piping (HDPE)	6,841,070.82	6,916,824.00	7,025,270.82	7,101,024.00	1.08%
5	Piping (HDPE)	6,695,213.68	6,868,212.50	7,025,270.82	7,198,269.64	2.46%
6	Piping (HDPE)	6,570,499.39	6,816,588.00	7,025,270.82	7,271,359.43	3.50%
7	Piping (HDPE)	6,441,070.82	6,766,390.50	7,025,270.82	7,350,590.50	4.63%
8	Piping (HDPE)	6,309,927.96	6,691,546.00	7,025,270.82	7,406,888.86	5.43%
9	Piping (HDPE)	6,139,856.53	6,596,190.50	7,025,270.82	7,481,604.79	6.50%
10	Piping (HDPE)	5,958,856.53	6,477,663.50	7,025,270.82	7,544,077.79	7.38%
11	Piping (HDPE)	5,754,570.82	6,336,723.50	7,025,270.82	7,607,423.50	8.29%
12	Piping (HDPE)	5,514,427.96	6,161,209.00	7,025,270.82	7,672,051.86	9.21%
13	Piping (HDPE)	5,248,142.25	5,943,509.50	7,025,270.82	7,720,638.07	9.90%
14	Piping (HDPE)	5,035,285.10	5,708,247.00	7,025,270.82	7,698,232.71	9.58%
15	Piping (HDPE)	4,785,981.53	5,460,569.00	7,025,270.82	7,699,858.29	9.60%
16	Piping (HDPE)	4,592,981.53	5,206,747.50	7,025,270.82	7,639,036.79	8.74%
17	Piping (HDPE)	4,335,838.68	4,943,782.50	7,025,270.82	7,633,214.64	8.65%
18	Piping (HDPE)	4,052,981.53	4,671,514.50	7,025,270.82	7,643,803.79	8.80%
19	Piping (HDPE)	3,839,695.82	4,389,153.50	7,025,270.82	7,574,728.50	7.82%
20	Piping (HDPE)	3,661,981.53	4,103,313.00	7,025,270.82	7,466,602.29	6.28%
21	Piping (HDPE)	3,503,981.53	3,831,362.25	7,025,270.82	7,352,651.54	4.66%
22	Piping (HDPE)	3,269,695.82	3,561,909.50	7,025,270.82	7,317,484.50	4.16%
23	Piping (HDPE)	3,147,489.37	3,301,774.25	7,025,270.82	7,179,555.69	2.20%
24	Piping (HDPE)	3,027,024.75	3,054,577.75	7,025,270.82	7,052,823.82	0.39%
25	Piping (HDPE)	2,899,256.04	2,817,745.50	7,025,270.82	6,943,760.28	1.16%
26	Piping (HDPE)	2,760,550.60	2,586,874.50	7,025,270.82	6,851,594.72	2.47%
27	Piping (HDPE)	2,593,501.74	2,364,232.00	7,025,270.82	6,796,001.08	3.26%
28	Piping (HDPE)	2,355,587.46	2,135,168.25	7,025,270.82	6,804,851.61	3.14%
29	Piping (HDPE)	2,164,444.60	1,925,255.25	7,025,270.82	6,786,081.47	3.40%
30	Piping (HDPE)	1,937,873.17	1,732,478.50	7,025,270.82	6,819,876.15	2.92%
31	Piping (HDPE)	1,760,587.46	1,545,553.38	7,025,270.82	6,810,236.74	3.06%
32	Piping (HDPE)	1,598,301.74	1,378,901.75	7,025,270.82	6,805,870.83	3.12%
33	Piping (HDPE)	1,403,873.17	1,229,031.50	7,025,270.82	6,850,429.15	2.49%
34	Piping (HDPE)	1,228,887.46	1,086,846.00	7,025,270.82	6,883,229.36	2.02%
35	Piping (HDPE)	1,041,173.17	955,246.88	7,025,270.82	6,939,344.52	1.22%
36	Piping (HDPE)	864,030.31	831,313.13	7,025,270.82	6,992,553.63	0.47%
37	Piping (HDPE)	710,173.17	713,462.13	7,025,270.82	7,028,559.77	0.05%
38	Piping (HDPE)	601,000.00	624,247.94	7,025,270.82	7,048,518.76	0.33%
39	Piping (HDPE)	410,428.57	516,214.78	7,025,270.82	7,131,057.03	1.51%
40	Piping (HDPE)	258,142.86	383,107.88	7,025,270.82	7,150,235.84	1.78%
41	Piping (HDPE)	204,142.86	291,789.09	7,025,270.82	7,112,917.06	1.25%
42	Piping (HDPE)	111,000.00	200,138.92	7,025,270.82	7,114,409.74	1.27%
43	Piping (HDPE)	68,857.14	135,285.08	7,025,270.82	7,091,698.75	0.95%
44	Piping (HDPE)	22,428.57	89,160.10	7,025,270.82	7,092,002.35	0.95%
45	Piping (HDPE)	-	46,644.89	7,025,270.82	7,071,915.71	0.66%
MAE						278,917
MAPE						3.97%

- **Forecasting Model on the Backfill Dataset**

This section applies the Fill-GRU forecasting model to the backfill dataset, comprising unseen data during the model's training and validation phases. The dataset consists of 53 weeks, with results starting from the fourth period due to the three previous periods' data needed for the model's initialization. Figures 5.9 and 5.10 show the result between actual and predicted values for the ETC and EAC calculation, respectively.

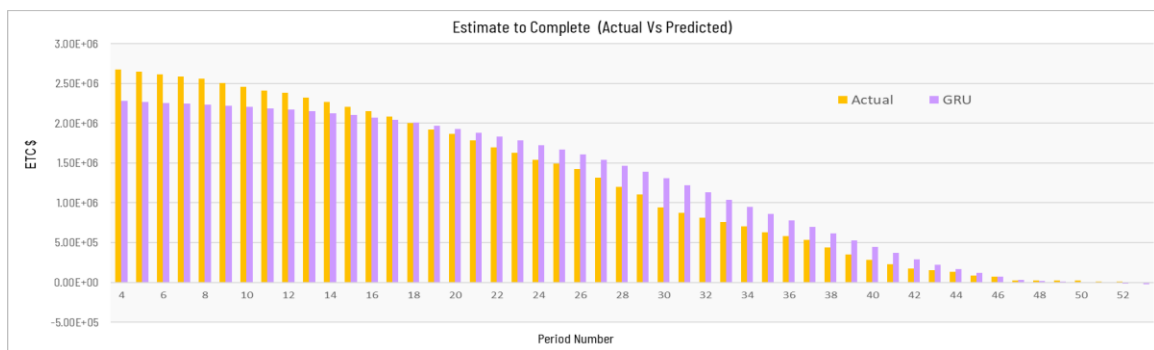


Figure 5.9 Actual vs. Predicted ETC for the Backfill Work Package

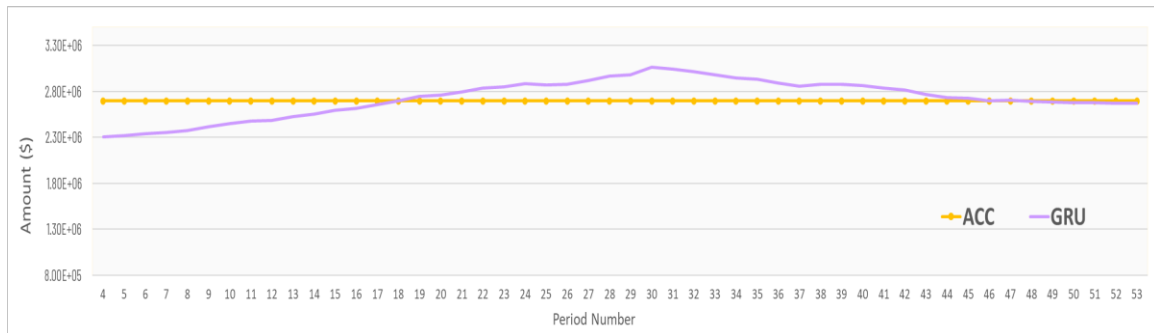


Figure 5.10 ACC vs. Predicted EAC for the Backfill Work Package

The following information is derived from the results of the GRU forecasting model's Backfill Package testing dataset. Table 5.6 reports the estimated monetary values for ETC and EAC and the percentage error for each evaluated period, representing the model's accuracy. This model's MAE is \$165,698, and MAPE is 6.14%.

Table 5.6 Results of Backfill-GRU on Testing Data

Testing Set						
Period	Work Package	ETC (\$)		ACC (\$)	GRU- EAC (\$)	% Error
		Actual	GRU			GRU
4	Barrow Fill	2,671,055.83	2,277,869.25	2,698,620.08	2,305,433.50	14.57%
5	Barrow Fill	2,644,156.58	2,266,044.75	2,698,620.08	2,320,508.25	14.01%
6	Barrow Fill	2,613,001.33	2,254,810.25	2,698,620.08	2,340,429.00	13.27%
7	Barrow Fill	2,588,097.08	2,247,143.00	2,698,620.08	2,357,666.00	12.63%
8	Barrow Fill	2,558,404.83	2,233,622.75	2,698,620.08	2,373,838.00	12.04%
9	Barrow Fill	2,501,813.33	2,221,353.75	2,698,620.08	2,418,160.50	10.39%
10	Barrow Fill	2,456,526.83	2,206,868.25	2,698,620.08	2,448,961.50	9.25%
11	Barrow Fill	2,409,411.58	2,187,977.25	2,698,620.08	2,477,185.75	8.21%
12	Barrow Fill	2,383,845.48	2,169,094.00	2,698,620.08	2,483,868.60	7.96%
13	Barrow Fill	2,322,366.23	2,149,881.75	2,698,620.08	2,526,135.60	6.39%
14	Barrow Fill	2,268,035.73	2,126,745.75	2,698,620.08	2,557,330.10	5.24%
15	Barrow Fill	2,205,326.23	2,099,913.50	2,698,620.08	2,593,207.35	3.91%
16	Barrow Fill	2,151,494.48	2,071,545.50	2,698,620.08	2,618,671.10	2.96%
17	Barrow Fill	2,083,681.11	2,040,795.38	2,698,620.08	2,655,734.35	1.59%
18	Barrow Fill	2,003,914.36	2,006,403.38	2,698,620.08	2,701,109.10	0.09%
19	Barrow Fill	1,918,894.11	1,967,957.75	2,698,620.08	2,747,683.72	1.82%
20	Barrow Fill	1,864,232.51	1,926,131.63	2,698,620.08	2,760,519.19	2.29%
21	Barrow Fill	1,785,084.21	1,882,257.75	2,698,620.08	2,795,793.61	3.60%
22	Barrow Fill	1,698,021.08	1,834,109.63	2,698,620.08	2,834,708.62	5.04%
23	Barrow Fill	1,625,266.51	1,780,752.13	2,698,620.08	2,854,105.70	5.76%
24	Barrow Fill	1,538,816.51	1,724,812.88	2,698,620.08	2,884,616.45	6.89%
25	Barrow Fill	1,491,545.70	1,665,715.88	2,698,620.08	2,872,790.25	6.45%
26	Barrow Fill	1,424,081.45	1,604,407.50	2,698,620.08	2,878,946.13	6.68%
27	Barrow Fill	1,317,016.45	1,538,298.75	2,698,620.08	2,919,902.38	8.20%
28	Barrow Fill	1,198,689.81	1,466,519.00	2,698,620.08	2,966,449.27	9.92%
29	Barrow Fill	1,104,258.48	1,387,985.00	2,698,620.08	2,982,346.60	10.51%
30	Barrow Fill	939,222.18	1,305,793.13	2,698,620.08	3,065,191.02	13.58%
31	Barrow Fill	876,379.68	1,218,332.63	2,698,620.08	3,040,573.02	12.67%
32	Barrow Fill	814,165.61	1,129,488.88	2,698,620.08	3,013,943.34	11.68%
33	Barrow Fill	758,072.86	1,039,678.63	2,698,620.08	2,980,225.84	10.44%
34	Barrow Fill	703,775.61	950,257.06	2,698,620.08	2,945,101.53	9.13%
35	Barrow Fill	627,566.61	861,697.56	2,698,620.08	2,932,751.03	8.68%
36	Barrow Fill	582,213.61	775,357.69	2,698,620.08	2,891,764.16	7.16%
37	Barrow Fill	536,890.63	697,427.69	2,698,620.08	2,859,157.13	5.95%
38	Barrow Fill	435,492.55	612,416.38	2,698,620.08	2,875,543.90	6.56%
39	Barrow Fill	350,307.25	528,633.63	2,698,620.08	2,876,946.46	6.61%
40	Barrow Fill	282,643.50	447,642.50	2,698,620.08	2,863,619.08	6.11%
41	Barrow Fill	226,451.00	367,397.91	2,698,620.08	2,839,566.99	5.22%
42	Barrow Fill	174,348.25	288,983.75	2,698,620.08	2,813,255.58	4.25%
43	Barrow Fill	152,968.50	223,636.64	2,698,620.08	2,769,288.22	2.62%
44	Barrow Fill	134,049.25	167,197.72	2,698,620.08	2,731,768.55	1.23%
45	Barrow Fill	85,205.00	116,390.48	2,698,620.08	2,729,805.56	1.16%
46	Barrow Fill	68,314.00	71,084.54	2,698,620.08	2,701,390.62	0.10%
47	Barrow Fill	23,275.00	31,485.42	2,698,620.08	2,706,830.49	0.30%
48	Barrow Fill	21,113.75	16,600.63	2,698,620.08	2,694,106.96	0.17%
49	Barrow Fill	21,113.75	9,276.23	2,698,620.08	2,686,782.56	0.44%
50	Barrow Fill	21,113.75	1,442.77	2,698,620.08	2,678,949.09	0.73%
51	Barrow Fill	9,376.50	7,492.06	2,698,620.08	2,681,751.52	0.63%
52	Barrow Fill	9,376.50	16,371.75	2,698,620.08	2,672,871.83	0.95%
53	Barrow Fill	-	25,619.38	2,698,620.08	2,673,000.69	0.95%
MAE					\$	165,698
MAPE						6.14%

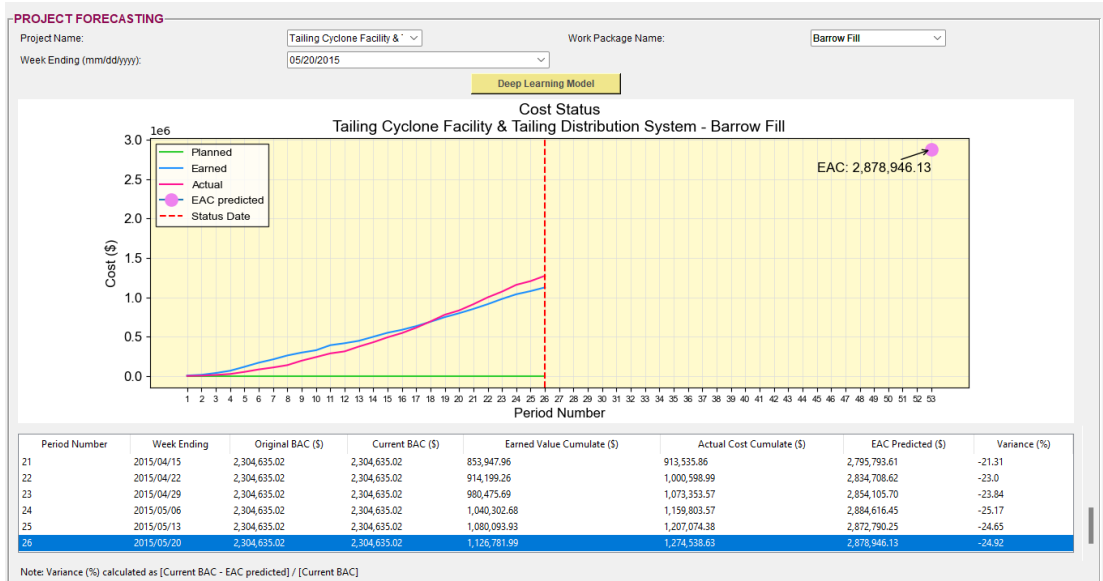


Figure 5.11 Cost Forecasting for the Fill Work Package - Period 26

Figure 5.11 depicts the deep learning model deployment outcomes within the GUI tailored for the backfill work package covering period 26. As listed in Table 5.5, the comparison between the actual estimate at completion and the predicted estimate at completion revealed \$2,698,620.08 and \$2,8878,946.13, respectively.

5.4 Comparative Analysis of Forecasting Models and EVM Methods

This section compares the proposed forecasting model with traditional methodologies used to calculate the EAC. The conventional method selected is Earned Value Management, which uses mathematical formulations to calculate EAC. These methodologies serve as a benchmark for evaluating the proposed Artificial Intelligence (AI)—based model. The equations from earned value methods applied in this comparative analysis were selected according to the available data; as mentioned before, the SPI values are not presented between the data collected, so formulas that include the SPI were excluded. The EVM formulas chosen are shown in equations 5.1 and 5.2.

Method 1 from Earned Value Management (EVM 1)

$$EAC = \frac{BAC}{CPI} \quad \text{Equation 5.1}$$

Method 2 from Earned Value Management (EVM 2)

$$EAC = AC + BAC - EV \quad \text{Equation 5.2}$$

- **Comparison results on the Concrete Dataset**

Table 5.7 compares the Concrete-GRU model and two traditional Earned Value Management methods through MAE and MAPE across 50 periods. The results report MAE and MAPE values for GRU, EVM 1, and EVM 2 are \$118,789 (7.38%), \$182,171 (11.32%), and \$147,784 (9.18%), respectively. These results show that the GRU model's established predictive precision indicates that it can be a reliable tool for cost forecasting. Table 5.7 presents the results of the comparative analysis for each period. Figure 5.12 provides a visual representation of these results.

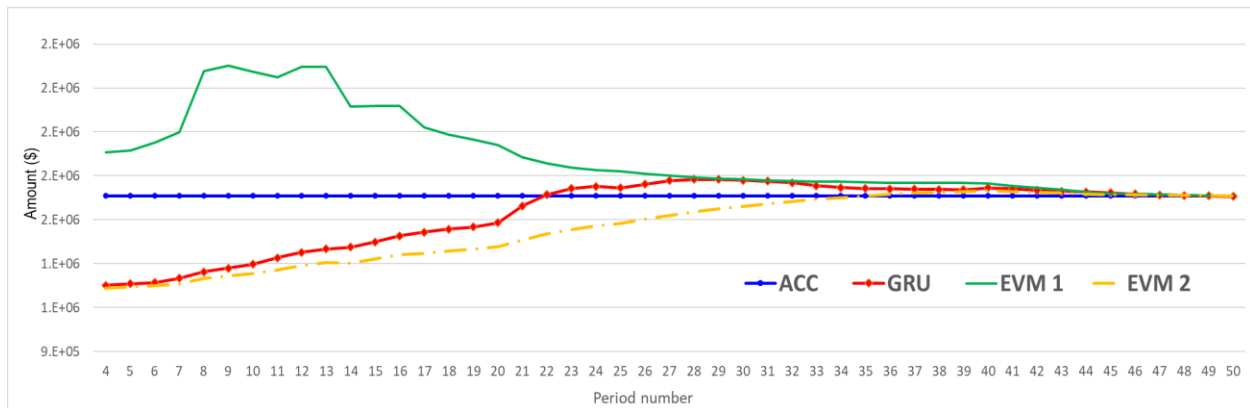


Figure 5.12 Comparative Analysis of EAC Predictions for Concrete WP

Table 5.7 Comparative Analysis of EAC Predictions for Concrete WP

Testing Set							
Period	ACC	EAC (\$)			% Error		
		GRU	EVM 1	EVM 2	GRU	EVM 1	EVM 2
4	1,609,043.24	1,201,246.52	1,807,351.97	1,189,748.18	25.34%	12.32%	26.06%
5	1,609,043.24	1,208,626.77	1,814,919.26	1,194,196.44	24.89%	12.79%	25.78%
6	1,609,043.24	1,214,664.26	1,850,177.27	1,198,848.02	24.51%	14.99%	25.49%
7	1,609,043.24	1,233,011.32	1,898,935.38	1,208,290.20	23.37%	18.02%	24.91%
8	1,609,043.24	1,262,918.92	2,177,070.07	1,232,154.89	21.51%	35.30%	23.42%
9	1,609,043.24	1,278,802.91	2,202,442.63	1,243,440.47	20.52%	36.88%	22.72%
10	1,609,043.24	1,297,043.35	2,174,346.15	1,254,872.26	19.39%	35.13%	22.01%
11	1,609,043.24	1,326,754.64	2,149,758.30	1,272,428.64	17.54%	33.60%	20.92%
12	1,609,043.24	1,351,002.47	2,196,966.06	1,292,216.51	16.04%	36.54%	19.69%
13	1,609,043.24	1,367,402.04	2,195,746.10	1,306,355.75	15.02%	36.46%	18.81%
14	1,609,043.24	1,375,469.20	2,015,313.29	1,303,594.41	14.52%	25.25%	18.98%
15	1,609,043.24	1,398,990.95	2,017,091.84	1,321,490.74	13.05%	25.36%	17.87%
16	1,609,043.24	1,427,073.36	2,018,706.59	1,342,496.40	11.31%	25.46%	16.57%
17	1,609,043.24	1,443,138.86	1,921,715.05	1,347,222.77	10.31%	19.43%	16.27%
18	1,609,043.24	1,457,618.49	1,886,491.62	1,358,047.81	9.41%	17.24%	15.60%
19	1,609,043.24	1,467,348.15	1,864,905.31	1,366,000.56	8.81%	15.90%	15.10%
20	1,609,043.24	1,486,200.52	1,839,243.63	1,377,171.95	7.63%	14.31%	14.41%
21	1,609,043.24	1,562,938.05	1,785,943.32	1,409,234.02	2.87%	10.99%	12.42%
22	1,609,043.24	1,614,531.48	1,756,475.23	1,435,232.53	0.34%	9.16%	10.80%
23	1,609,043.24	1,641,931.34	1,738,088.98	1,456,337.08	2.04%	8.02%	9.49%
24	1,609,043.24	1,651,526.71	1,726,473.88	1,472,383.83	2.64%	7.30%	8.49%
25	1,609,043.24	1,645,518.20	1,719,414.48	1,483,434.88	2.27%	6.86%	7.81%
26	1,609,043.24	1,661,604.39	1,709,112.72	1,501,715.43	3.27%	6.22%	6.67%
27	1,609,043.24	1,678,658.58	1,700,204.63	1,520,030.68	4.33%	5.67%	5.53%
28	1,609,043.24	1,683,565.92	1,693,989.02	1,534,492.66	4.63%	5.28%	4.63%
29	1,609,043.24	1,683,875.20	1,688,487.18	1,548,665.39	4.65%	4.94%	3.75%
30	1,609,043.24	1,679,820.77	1,684,136.19	1,560,925.07	4.40%	4.67%	2.99%
31	1,609,043.24	1,675,130.35	1,680,265.64	1,572,717.51	4.11%	4.43%	2.26%
32	1,609,043.24	1,668,262.33	1,676,893.87	1,583,751.22	3.68%	4.22%	1.57%
33	1,609,043.24	1,654,776.78	1,674,184.30	1,593,185.17	2.84%	4.05%	0.99%
34	1,609,043.24	1,645,961.51	1,672,122.43	1,600,732.34	2.29%	3.92%	0.52%
35	1,609,043.24	1,642,456.74	1,670,027.48	1,608,751.19	2.08%	3.79%	0.02%
36	1,609,043.24	1,640,332.59	1,668,935.48	1,615,583.74	1.94%	3.72%	0.41%
37	1,609,043.24	1,639,162.37	1,668,099.52	1,620,982.05	1.87%	3.67%	0.74%
38	1,609,043.24	1,637,765.97	1,667,527.86	1,624,760.86	1.79%	3.63%	0.98%
39	1,609,043.24	1,637,028.49	1,667,017.02	1,628,199.58	1.74%	3.60%	1.19%
40	1,609,043.24	1,644,537.85	1,664,085.59	1,633,068.58	2.21%	3.42%	1.49%
41	1,609,043.24	1,640,637.42	1,653,441.00	1,628,662.00	1.96%	2.76%	1.22%
42	1,609,043.24	1,633,139.29	1,645,274.35	1,625,206.63	1.50%	2.25%	1.00%
43	1,609,043.24	1,630,293.48	1,635,996.88	1,621,200.06	1.32%	1.68%	0.76%
44	1,609,043.24	1,627,040.50	1,625,882.91	1,616,730.59	1.12%	1.05%	0.48%
45	1,609,043.24	1,622,767.29	1,619,156.91	1,613,697.73	0.85%	0.63%	0.29%
46	1,609,043.24	1,617,521.30	1,617,194.54	1,612,803.54	0.53%	0.51%	0.23%
47	1,609,043.24	1,613,199.04	1,614,954.25	1,611,777.49	0.26%	0.37%	0.17%
48	1,609,043.24	1,609,976.78	1,612,770.54	1,610,771.97	0.06%	0.23%	0.11%
49	1,609,043.24	1,608,278.20	1,610,697.06	1,609,812.23	0.05%	0.10%	0.05%
50	1,609,043.24	1,606,113.27	1,609,043.24	1,609,043.24	0.18%	0.00%	0.00%
MAE		\$			118,789	\$	182,171
MAPE					7.38%		11.32%
						\$	147,784
							9.18%

- **Comparison results on the Piping (HDPE) Dataset**

Table 5.8 assesses the Piping-GRU model and two Earned Value Management methods through MAE and MAPE through 45 weeks. The results show MAE and MAPE values for GRU, EVM 1, and EVM 2 are \$278,917 (3.97%), \$922,266 (13.13%), and \$ 474,996 (7.62%), respectively. These results show that the GRU model presents the lowest MAE and MAPE, indicating it can be a reliable tool for cost forecasting. Figure 5.13 visually represents comparative analysis, showing that the GRU model's results (red line) are closer to the ACC (blue line).

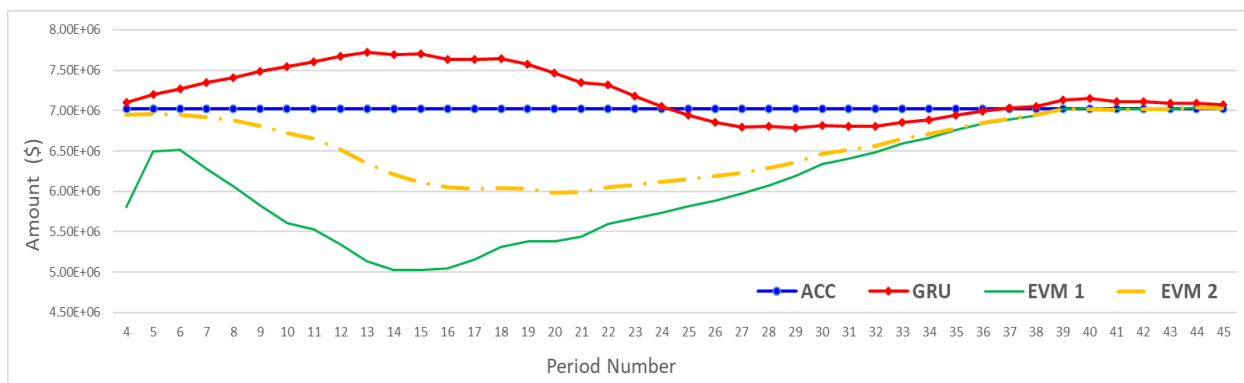


Figure 5.13 Comparative Analysis of EAC Predictions for Piping WP

Table 5.8 Comparative Analysis of EAC Predictions for Piping (HDPE) WP

Period	Testing Set						
	ACC	EAC (\$)			% Error		
		GRU	EVM 1	EVM 2	GRU	EVM 1	EVM 2
4	2,698,620	2,305,434	921,854	2,263,289	14.57%	65.84%	16.13%
5	2,698,620	2,320,508	1,055,440	2,240,173	14.01%	60.89%	16.99%
6	2,698,620	2,340,429	1,151,984	2,218,967	13.27%	57.31%	17.77%
7	2,698,620	2,357,666	1,194,415	2,201,903	12.63%	55.74%	18.41%
8	2,698,620	2,373,838	1,227,950	2,181,692	12.04%	54.50%	19.16%
9	2,698,620	2,418,161	1,512,967	2,201,655	10.39%	43.94%	18.42%
10	2,698,620	2,448,962	1,695,115	2,217,584	9.25%	37.19%	17.83%
11	2,698,620	2,477,186	1,694,156	2,200,421	8.21%	37.22%	18.46%
12	2,698,620	2,483,869	1,733,741	2,200,985	7.96%	35.75%	18.44%
13	2,698,620	2,526,136	1,930,634	2,231,747	6.39%	28.46%	17.30%
14	2,698,620	2,557,330	1,984,379	2,235,144	5.24%	26.47%	17.17%
15	2,698,620	2,593,207	2,062,469	2,246,715	3.91%	23.57%	16.75%
16	2,698,620	2,618,671	2,142,934	2,263,350	2.96%	20.59%	16.13%
17	2,698,620	2,655,734	2,230,632	2,284,234	1.59%	17.34%	15.36%
18	2,698,620	2,701,109	2,318,531	2,308,799	0.09%	14.08%	14.45%
19	2,698,620	2,747,684	2,397,962	2,334,981	1.82%	11.14%	13.47%
20	2,698,620	2,760,519	2,406,183	2,339,849	2.29%	10.84%	13.29%
21	2,698,620	2,795,794	2,465,451	2,364,223	3.60%	8.64%	12.39%
22	2,698,620	2,834,709	2,522,443	2,391,035	5.04%	6.53%	11.40%
23	2,698,620	2,854,106	2,522,947	2,397,513	5.76%	6.51%	11.16%
24	2,698,620	2,884,616	2,569,371	2,424,136	6.89%	4.79%	10.17%
25	2,698,620	2,872,790	2,575,578	2,431,615	6.45%	4.56%	9.89%
26	2,698,620	2,878,946	2,606,845	2,452,392	6.68%	3.40%	9.12%
27	2,698,620	2,919,902	2,651,476	2,485,363	8.20%	1.75%	7.90%
28	2,698,620	2,966,449	2,727,124	2,537,006	9.92%	1.06%	5.99%
29	2,698,620	2,982,347	2,696,869	2,536,520	10.51%	0.06%	6.01%
30	2,698,620	3,065,191	2,785,855	2,608,548	13.58%	3.23%	3.34%
31	2,698,620	3,040,573	2,765,363	2,608,232	12.67%	2.47%	3.35%
32	2,698,620	3,013,943	2,746,688	2,607,920	11.68%	1.78%	3.36%
33	2,698,620	2,980,226	2,731,073	2,607,638	10.44%	1.20%	3.37%
34	2,698,620	2,945,102	2,732,392	2,616,928	9.13%	1.25%	3.03%
35	2,698,620	2,932,751	2,754,919	2,643,143	8.68%	2.09%	2.06%
36	2,698,620	2,891,764	2,749,700	2,647,195	7.16%	1.89%	1.91%
37	2,698,620	2,859,157	2,759,739	2,661,123	5.95%	2.26%	1.39%
38	2,698,620	2,875,544	2,780,984	2,692,281	6.56%	3.05%	0.23%
39	2,698,620	2,876,946	2,797,639	2,718,458	6.61%	3.67%	0.74%
40	2,698,620	2,863,619	2,798,348	2,730,886	6.11%	3.70%	1.20%
41	2,698,620	2,839,567	2,768,082	2,718,539	5.22%	2.57%	0.74%
42	2,698,620	2,813,256	2,737,833	2,704,042	4.25%	1.45%	0.20%
43	2,698,620	2,769,288	2,713,975	2,688,587	2.62%	0.57%	0.37%
44	2,698,620	2,731,769	2,692,712	2,674,244	1.23%	0.22%	0.90%
45	2,698,620	2,729,806	2,709,600	2,695,224	1.16%	0.41%	0.13%
46	2,698,620	2,701,391	2,686,035	2,678,122	0.10%	0.47%	0.76%
47	2,698,620	2,706,830	2,695,889	2,692,908	0.30%	0.10%	0.21%
48	2,698,620	2,694,107	2,695,017	2,692,481	0.17%	0.13%	0.23%
49	2,698,620	2,686,783	2,695,017	2,692,481	0.44%	0.13%	0.23%
50	2,698,620	2,678,949	2,695,017	2,692,481	0.73%	0.13%	0.23%
51	2,698,620	2,681,752	2,697,026	2,695,894	0.63%	0.06%	0.10%
52	2,698,620	2,672,872	2,697,026	2,695,894	0.95%	0.06%	0.10%
53	2,698,620	2,673,001	2,698,620	2,698,620	0.95%	0.00%	0.00%
MAE					\$ 165,698	\$ 362,196	\$ 214,655
MAPE					6.14%	13.42%	7.95%

- **Comparison results on the Backfill Dataset**

Table 5.9 assesses the Fill-GRU model and two Earned Value Management methods through MAE and MAPE over 53 periods. The results show that the MAE and MAPE values for GRU, EVM 1, and EVM 2 are \$1165,698 (6.14%), \$362,195 (13.42%), and \$214,654 (7.95%), respectively. These results show that the GRU model presents the lowest MAE and MAPE, indicating superiority in terms of accuracy against the traditional EVM method. Figure 5.14 delivers a graphical representation of comparative analysis.

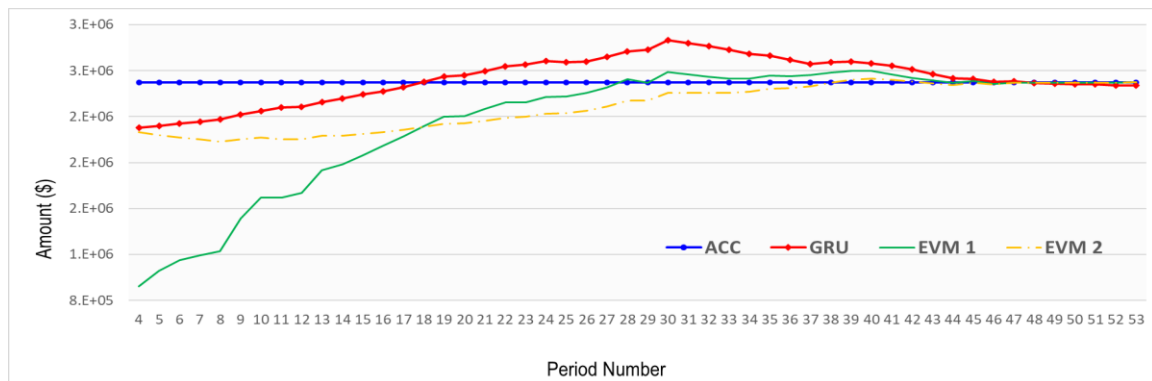


Figure 5.14 Comparative Analysis of EAC Predictions for Backfill WP

In this comparative analysis, the GRU model demonstrated superior performance metrics over traditional forecasting methodologies across three distinct work package datasets. These outcomes position the GRU model as a promising alternative for estimating the final costs of ongoing construction projects with enhanced accuracy and reliability.

Table 5.9 Comparative Analysis of EAC Predictions for Backfill WP

Testing Set							
Period	ACC	EAC (\$)			% Error		
		GRU	EVM 1	EVM 2	GRU	EVM 1	EVM 2
4	2,698,620	2,305,434	921,854	2,263,289	14.57%	65.84%	16.13%
5	2,698,620	2,320,508	1,055,440	2,240,173	14.01%	60.89%	16.99%
6	2,698,620	2,340,429	1,151,984	2,218,967	13.27%	57.31%	17.77%
7	2,698,620	2,357,666	1,194,415	2,201,903	12.63%	55.74%	18.41%
8	2,698,620	2,373,838	1,227,950	2,181,692	12.04%	54.50%	19.16%
9	2,698,620	2,418,161	1,512,967	2,201,655	10.39%	43.94%	18.42%
10	2,698,620	2,448,962	1,695,115	2,217,584	9.25%	37.19%	17.83%
11	2,698,620	2,477,186	1,694,156	2,200,421	8.21%	37.22%	18.46%
12	2,698,620	2,483,869	1,733,741	2,200,985	7.96%	35.75%	18.44%
13	2,698,620	2,526,136	1,930,634	2,231,747	6.39%	28.46%	17.30%
14	2,698,620	2,557,330	1,984,379	2,235,144	5.24%	26.47%	17.17%
15	2,698,620	2,593,207	2,062,469	2,246,715	3.91%	23.57%	16.75%
16	2,698,620	2,618,671	2,142,934	2,263,350	2.96%	20.59%	16.13%
17	2,698,620	2,655,734	2,230,632	2,284,234	1.59%	17.34%	15.36%
18	2,698,620	2,701,109	2,318,531	2,308,799	0.09%	14.08%	14.45%
19	2,698,620	2,747,684	2,397,962	2,334,981	1.82%	11.14%	13.47%
20	2,698,620	2,760,519	2,406,183	2,339,849	2.29%	10.84%	13.29%
21	2,698,620	2,795,794	2,465,451	2,364,223	3.60%	8.64%	12.39%
22	2,698,620	2,834,709	2,522,443	2,391,035	5.04%	6.53%	11.40%
23	2,698,620	2,854,106	2,522,947	2,397,513	5.76%	6.51%	11.16%
24	2,698,620	2,884,616	2,569,371	2,424,136	6.89%	4.79%	10.17%
25	2,698,620	2,872,790	2,575,578	2,431,615	6.45%	4.56%	9.89%
26	2,698,620	2,878,946	2,606,845	2,452,392	6.68%	3.40%	9.12%
27	2,698,620	2,919,902	2,651,476	2,485,363	8.20%	1.75%	7.90%
28	2,698,620	2,966,449	2,727,124	2,537,006	9.92%	1.06%	5.99%
29	2,698,620	2,982,347	2,696,869	2,536,520	10.51%	0.06%	6.01%
30	2,698,620	3,065,191	2,785,855	2,608,548	13.58%	3.23%	3.34%
31	2,698,620	3,040,573	2,765,363	2,608,232	12.67%	2.47%	3.35%
32	2,698,620	3,013,943	2,746,688	2,607,920	11.68%	1.78%	3.36%
33	2,698,620	2,980,226	2,731,073	2,607,638	10.44%	1.20%	3.37%
34	2,698,620	2,945,102	2,732,392	2,616,928	9.13%	1.25%	3.03%
35	2,698,620	2,932,751	2,754,919	2,643,143	8.68%	2.09%	2.06%
36	2,698,620	2,891,764	2,749,700	2,647,195	7.16%	1.89%	1.91%
37	2,698,620	2,859,157	2,759,739	2,661,123	5.95%	2.26%	1.39%
38	2,698,620	2,875,544	2,780,984	2,692,281	6.56%	3.05%	0.23%
39	2,698,620	2,876,946	2,797,639	2,718,458	6.61%	3.67%	0.74%
40	2,698,620	2,863,619	2,798,348	2,730,886	6.11%	3.70%	1.20%
41	2,698,620	2,839,567	2,768,082	2,718,539	5.22%	2.57%	0.74%
42	2,698,620	2,813,256	2,737,833	2,704,042	4.25%	1.45%	0.20%
43	2,698,620	2,769,288	2,713,975	2,688,587	2.62%	0.57%	0.37%
44	2,698,620	2,731,769	2,692,712	2,674,244	1.23%	0.22%	0.90%
45	2,698,620	2,729,806	2,709,600	2,695,224	1.16%	0.41%	0.13%
46	2,698,620	2,701,391	2,686,035	2,678,122	0.10%	0.47%	0.76%
47	2,698,620	2,706,830	2,695,889	2,692,908	0.30%	0.10%	0.21%
48	2,698,620	2,694,107	2,695,017	2,692,481	0.17%	0.13%	0.23%
49	2,698,620	2,686,783	2,695,017	2,692,481	0.44%	0.13%	0.23%
50	2,698,620	2,678,949	2,695,017	2,692,481	0.73%	0.13%	0.23%
51	2,698,620	2,681,752	2,697,026	2,695,894	0.63%	0.06%	0.10%
52	2,698,620	2,672,872	2,697,026	2,695,894	0.95%	0.06%	0.10%
53	2,698,620	2,673,001	2,698,620	2,698,620	0.95%	0.00%	0.00%
MAE					\$ 165,698	\$ 362,196	\$ 214,655
MAPE					6.14%	13.42%	7.95%

5.5 Sensitivity Analysis

The GRU model exhibits reliable predictive accuracy and outperforms traditional methodologies. However, a model verification process must be developed to recognize the relationship between variables and output targets. A sensitivity analysis method is used for the proposed forecasting model. In this method, the input parameters are varied to observe the model's response and assess how it aligns with the actual system's behaviour (Sargen, 2007). The testing dataset, with the same conditions discussed in section 5.2, is considered to achieve this. By varying the amounts of earned value and actual cost, the model's output is also expected to vary. This study creates four scenarios by increasing and decreasing each period's earned value and actual cost.

- **Scenario 01** This first scenario examines the impact of increasing the EV input by 10% to 15%. This range of a variable increase, rather than a fixed increase, such as 10%, is strategically employed to eliminate the potential homogenization of the target value resulting from the normalization process (specifically, max-min normalization) before applying the forecasting model. This scenario only increases the EV input between the proposed range and keeps all other variables constant. The results show a direct relationship between the increased EV input and the EAC output; in other words, when the amount of EV increases, the EAC output also increases.
- **Scenario 02** examines the impact of decreasing the EV input by 10% to 15%. This scenario only decreases the EV input within the proposed range every period and keeps all other variables constant. The study found a direct correlation between EV and EAC.
- **Scenario 03** considers the impact of increasing the AC input by 10% to 15%. This approach uses a variable increase instead of a fixed increase (e.g. 15%) to avoid homogenizing the target value during the normalization process (max-min normalization). This scenario only increases

the AC input between the proposed range and keeps all other variables constant. The results found a direct relationship between the increased AC input and the EAC output; in other words, when the amount of AC increases, the EAC output also increases.

- **Scenario 04** investigates the impact of decreasing the AC input by 10% to 15%. This scenario only decreases the AC input within the proposed range and keeps all other variables constant. The results found a direct relationship between the increased AC input and the EAC output; in other words, when the amount of AC decreases, the EAC output also decreases.

The results of the prediction model for the GRU model and the model applied after the sensitivity analysis method are presented in Figures 5.15, 5.16, and 5.17. Figure 5.15 shows the values of the first eighteen periods for the concrete work package, while Figure 5.16 displays the values for the piping (HDPE) work package. Lastly, Figure 5.17 denotes the values of the first thirty-one periods for the backfill work package. From the sensitivity analysis, this study found a direct relationship between EV and AC values and the EAC output. As the values of EV and AC increased, EAC also increased, and as the values of EV and AC decreased, EAC decreased. Also, the EV input was found to be the most sensitive parameter.

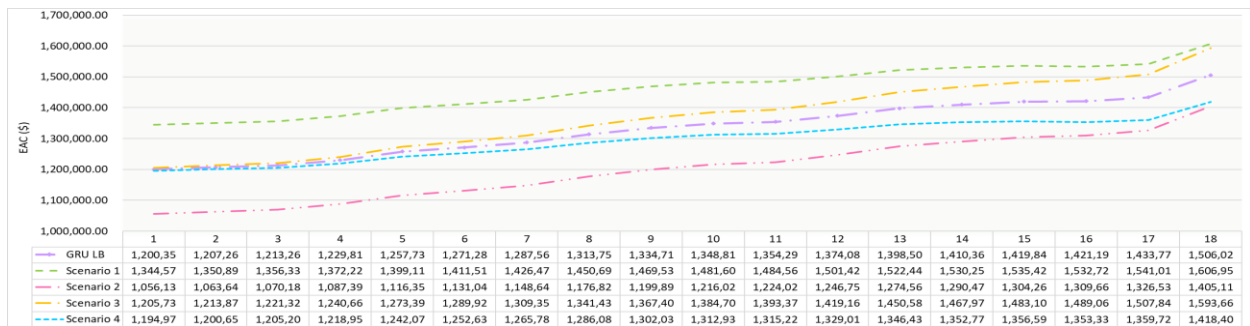


Figure 5.15 Results - Sensitivity Analysis for Concrete Work Package

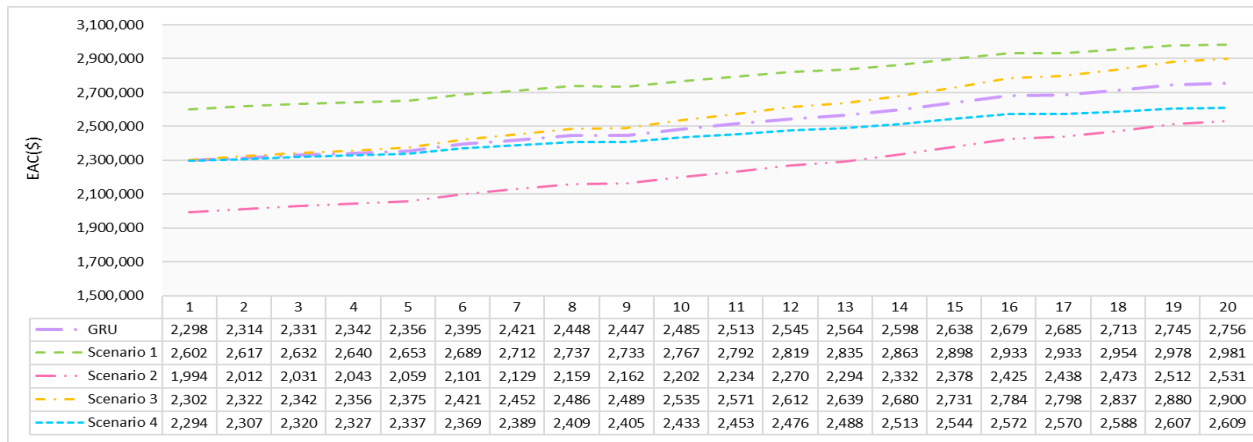


Figure 5.16 Results - Sensitivity Analysis for Piping (HDPE) Work Package

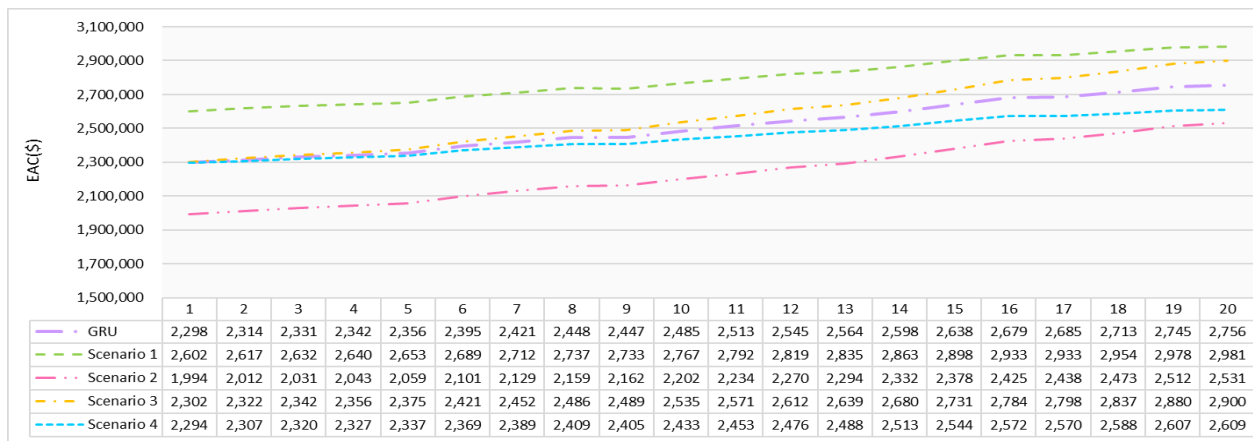


Figure 5.17 Results - Sensitivity Analysis for Backfill Work Package

Chapter 6: Conclusion

6.1 Research Summary

Construction projects are complex systems involving different phases. This study focuses on the execution phase, which is essential due to its impact on project success and financial outcomes. During the execution phase, controlling and monitoring is a pivotal stage that helps the project manager detect potential cost deviations early, avoiding or mitigating cost overruns. Cost overruns are prevalent in most construction projects and are influenced by many factors including inaccuracies in cost estimate at completion (EAC). Inaccurate cost forecasting impacts effective budget control, resource management, and decision-making, leading to cost overruns. Moreover, inaccurate cost forecasting can harm contractors' reputations, leading clients to avoid collaborating on future projects.

In the construction industry, conventional approaches to cost forecasting are bottom-up estimating and earned value management. The bottom-up estimating method entails a detailed process that breaks down the remaining work into minor components, such as individual tasks or work packages, and sums them to get the total ETC. This method is commonly used when a high degree of accuracy is required and when detailed information about the project's cost elements is available. Despite the high level of accuracy that can provide this method, it is notably exhaustive and thereby consumes a significant amount of time and resources. On the other hand, Earned Value Management is commonly employed in construction project because of its easy integration with current project control systems. Nonetheless, its ability to forecast early costs can be imprecise because of the incorrect presumption that the initial cost performance will remain the same for the project's remainder.

To address these issues, this study aims to enhance the accuracy of EAC calculation by using historical data from completed projects and artificial intelligence capabilities, specifically deep learning algorithms like gate recurrent unit (GRU).

This approach combined quantitative data analysis with qualitative insights, leveraging the GRU's proficiency in recognizing and modelling long-term dependencies in time series data. This study begins by exploring factors that influence the EAC calculation; for this purpose, three criteria were combined: current practices in cost forecasting, literature review, and available data for the study. Based on these criteria, the following attributes were proposed to be included in the model: earned value to date, actual cost to date, labour cost to date, percentage of duration completed, and quantity performed. However, this study considers that future investigations should include additional factors in their analysis, such as unexpected weather conditions, material price escalation, rate inflation, and planned value; for this study, these factors were excluded, given the unavailable data.

Continuing from the outlined starting point, the collected dataset was preprocessed to be used as the training, validation, and testing set for the forecasting model. Three deep learning algorithms (RNN, LSTM, and GRU) were selected to evaluate the forecasting model; one of the main criteria for choosing these algorithms was their capability to work with time series data, like the data collected from construction project execution. After conducting a comparative analysis, the GRU algorithm outperformed other algorithms regarding RMSE, MAE, and MAPE. Therefore, it was chosen to construct the proposed model, and the performance results are presented in Table 5.1. Then, a forecasting model was developed for each work package (concrete, backfill, and piping) to better capture their variability and complexity. The model's output is the ETC for every

evaluated period, and the EAC is calculated by adding the incurred actual cost for the completed work and the predicted ETC.

Building upon the established groundwork, the next phase of the study evaluated the forecasting GRU models using the testing dataset (unseen data) for each work package. . The results of this study reveal that the Gated Recurrent Unit (GRU) model outperforms traditional cost forecasting techniques. The GRU model recorded Mean Absolute Percentage Errors (MAPE) of 7.38%, 6.14%, and 3.97% for the concrete, backfill, and piping work packages, respectively. Conversely, the Earned Value Management (EVM) method showed higher MAPE values of 11.32%, 13.42%, and 13.3% for these work packages. Additionally, GRU models generated more stable and reasonable percentage errors during early project execution than EVM.

A sensitivity analysis was developed to verify the model by varying one attribute and keeping the others. The results reflect that variation of the actual cost and earned value had a direct relationship with the EAC as expected, thereby demonstrating verification of the model. Finally, this research developed a user interface to correctly store related construction project data in real time and execute the proposed deep learning models for forecasting EAC at the work package level. The GUI was developed to provide an easy-to-use interface that benefits project managers, cost controllers, or project teams, allowing them to use the forecasting model developed easily.

Overall, the findings underscore the potential of artificial intelligence-based models with the GRU algorithm to revolutionize cost forecasting by providing more precise and dynamic real-time forecasting. Expanding the dataset used in the training and validation phases is recommended to enhance the forecasting accuracy of the proposed model. An enriched dataset will improve the model's robustness and ability to generalize across various project scenarios, thereby solidifying its advantage over conventional methods. This analysis paves the way for adopting more

sophisticated, AI-driven approaches in the construction industry, offering significant improvements in cost forecasting.

6.2 Research Contributions

6.2.1 Academic Contributions

This study introduces a novel forecasting framework to predict cost estimate at completion of ongoing construction projects by applying a deep-learning model. The academic contributions are as follows:

- 1 Explore the significant factors that influence the cost estimate at completion of construction projects, including but not limited to periodic information about the actual cost, earned value, executed quantity, labour cost, and percentage duration.
- 2 Highlight the potential of using innovative technology, such as deep learning methods, by providing a systematic framework with a step-by-step process that can be effectively applied to predict cost forecasting at the work package level.
- 3 Investigate the applicability of deep learning algorithms to forecast the cost estimate at completion (EAC) using a time series dataset.
- 4 This study uses data from multiple completed projects to train and test the deep learning model. By leveraging a diverse dataset, the model can identify complex patterns and relationships within the data.

6.2.2 Industrial Contributions:

The following are considered among the industrial contributions:

- 1 Propose a data acquisition model that periodically gathers data during project execution. Considering that the accuracy of the predicted model depends on the quality and quantity

of the data available to train it, the developed DAM contains the essential input parameters required to deploy the proposed cost forecasting model. It is important to note that forecasting models require high-level data; therefore, data will be a premium asset for the reliability of model results.

- 2 This investigation offers construction companies an advanced deep learning framework to improve cost forecasting accuracy at the work package level. This framework provides step-by-step instructions on building a cost forecasting model, including data collection, preprocessing, model building, and model deployment. By adapting this framework to their historical data, construction companies can achieve more accurate cost forecasting models.
- 3 This study developed a graphical user interface engineered to help the project managers and project controllers interact with the deep learning model without the need for programming expertise. This interface ensures straightforward data entry and updating, showing an error handling mechanism that alerts user data entry mistakes, thereby maintaining the reliability of the forecasting process. Moreover, the interface is connected to a SQLite database, enabling a comprehensive recording of all the data entries. This integration facilitates an organized data management system to save historical data from completed projects. Finally, the interface offers real-time visualizations about insightful cost forecasts at the work package level, enabling project managers to make informed decisions efficiently.

6.3 Limitations

One central limitation of this research was the difficulty in obtaining complete and high-quality cost-related data from completed projects to train the GRU model. Construction companies keep such data confidential, which is unavailable for academic purposes. Despite finding a construction

company that agreed to share information, only a limited dataset was obtained for this study. The shared dataset included five projects with several work packages; however, only three had more completed information. This limited dataset hindered the potential benefits of the proposed approach. Deep learning models require extensive and diverse datasets, which significantly affect the accuracy of forecasting models. The performance metrics results from this study suggest the potential for improvement by training the model using a more expansive dataset.

The second limitation is the need for standardized procedures for data acquisition, which can vary across projects, locations, or managerial approaches. From the data provided by the contractor, only five of the required attributes for the model were provided. This absence of uniformity can lead to the omission of collecting crucial factors necessary for the model's training, thus affecting the overall accuracy of the forecasting model.

6.4 Future Work and Recommendations

This study presents a foundational framework representing an initial step towards deep learning-based cost control solutions, specifically in cost forecasting for construction projects with time series data. To further enhance the applicability and accuracy of the proposed forecasting model, the following recommendations are offered for future investigations:

- Integrating a more comprehensive range of work package data and following the proposed approach could enhance the model's potency and applicability. Diverse work package datasets should be systematically included in future training to enrich the model's predictive capabilities and adaptability across different project types in the construction industry.
- Determining additional parameters to expand requires judicious consideration. A team of professionals with experience in cost control management, specifically in cost forecasting,

is recommended to participate in the input selection. Their insights will be invaluable in identifying the most influential parameters that correlate strongly with the cost estimate at completion in different scenarios and realities. This approach, guided by experts, ensures that the model's enhancements are practical and grounded in industry realities.

- Considering the increasing trend of utilizing Power BI for enhanced reportability within the construction industry, it is recommended that the proposed model's outcomes be integrated with Power BI. This integration would enable users to incorporate the model's results into their reporting seamlessly. Such a linkage leverages Power BI's robust data visualization and analysis capabilities. It ensures that the model's results are readily accessible to the project team and actionable within the industry's prevalent reporting practices.
- The model should be updated with historical data from completed projects to stay relevant and practical. This means the model can start working with the built-in initial data set. However, the model should be rebuilt after a certain period with a considerable amount of historical data stored from completed projects. This update process involves data preprocessing, hyperparameter optimization, and benchmarking against similar algorithms to ensure the model's superiority. In other words, follow all the steps in the proposed framework in this research. It guarantees that the model evolves in response to a new dataset, improving its accuracy and applicability in real-world scenarios.
- Assessing the model intensively by testing it in real-world case studies is essential. To improve the model, it should be tested across various projects to identify its strengths and areas for improvement. The feedback received from these practical applications will help refine the model continuously.

References

- Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-94463-0>
- Amos, S. J. (2012). *Skills & knowledge of cost engineering: A continuing project of the AACE International Education Board* (5th ed. rev). AACE International.
- Araba, A. M., Memon, Z. A., Alhawati, M., Ali, M., & Milad, A. (2021). Estimation at Completion in Civil Engineering Projects: Review of Regression and Soft Computing Models. *Knowledge-Based Engineering and Sciences*, 2(2), 1–12. <https://doi.org/10.51526/kbes.2021.2.2.1-12>
- Atout, M. M. (2019). Monitoring and Control Process of Construction Projects. *Proceedings of the Creative Construction Conference 2019*, 584–590. <https://doi.org/10.3311/CCC2019-080>
- Awad, M., & Khanna, R. (2015). *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Apress. <https://doi.org/10.1007/978-1-4302-5990-9>
- Baloi, D., & Price, A. D. F. (2003). Modelling global risk factors affecting construction cost performance. *International Journal of Project Management*, 21(4), 261–269. [https://doi.org/10.1016/S0263-7863\(02\)00017-0](https://doi.org/10.1016/S0263-7863(02)00017-0)
- Barraza, G. A., Back, W. E., & Mata, F. (2004). Probabilistic Forecasting of Project Performance Using Stochastic S Curves. *Journal of Construction Engineering and Management*, 130(1), 25–32. [https://doi.org/10.1061/\(ASCE\)0733-9364\(2004\)130:1\(25\)](https://doi.org/10.1061/(ASCE)0733-9364(2004)130:1(25))
- Barrera-Animas, A. Y., Oyedele, L. O., Bilal, M., Akinosho, T. D., Delgado, J. M. D., & Akanbi, L. A. (2022). Rainfall prediction: A comparative analysis of modern machine learning

- algorithms for time-series forecasting. *Machine Learning with Applications*, 7, 100204.
<https://doi.org/10.1016/j.mlwa.2021.100204>
- Bartlett, J. W., & Keogh, R. H. (2018). Bayesian correction for covariate measurement error: A frequentist evaluation and comparison with regression calibration. *Statistical Methods in Medical Research*, 27(6), 1695–1708. <https://doi.org/10.1177/0962280216667764>
- Botchkarev, A. (2019). A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 14, 045–076. <https://doi.org/10.28945/4184>
- Caron, F., Ruggeri, F., & Merli, A. (2013). A Bayesian Approach to Improve Estimate at Completion in Earned Value Management. *Project Management Journal*, 44(1), 3–16.
<https://doi.org/10.1002/pmj.21303>
- Caron, F., Ruggeri, F., & Pierini, B. (2016). A Bayesian approach to improving estimate to complete. *International Journal of Project Management*, 34(8), 1687–1702.
<https://doi.org/10.1016/j.ijproman.2016.09.007>
- Chen, G. (2018). *A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation* (arXiv:1610.02583). arXiv. <http://arxiv.org/abs/1610.02583>
- Chen, P. P.-S. (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9–36. <https://doi.org/10.1145/320434.320440>
- Cheng, M.-Y., Chang, Y.-H., & Korir, D. (2019). Novel Approach to Estimating Schedule to Completion in Construction Projects Using Sequence and Non sequence Learning. *Journal of Construction Engineering and Management*, 145(11), 04019072.
[https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0001697](https://doi.org/10.1061/(ASCE)CO.1943-7862.0001697)

- Cheng, M.-Y., & Hoang, N.-D. (2014). Interval Estimation of Construction Cost at Completion Using Least Squares Support Vector Machine. *Journal of Civil Engineering and Management*, 20(2), 223–236. <https://doi.org/10.3846/13923730.2013.801891>
- Cheng, M.-Y., Hoang, N.-D., Roy, A. F. V., & Wu, Y.-W. (2012). A novel time-depended evolutionary fuzzy SVM inference model for estimating construction project at completion. *Engineering Applications of Artificial Intelligence*, 25(4), 744–752. <https://doi.org/10.1016/j.engappai.2011.09.022>
- Cheng, M.-Y., Peng, H.-S., Wu, Y.-W., & Chen, T.-L. (2010). Estimate at Completion for construction projects using Evolutionary Support Vector Machine Inference Model. *Automation in Construction*, 19(5), 619–629. <https://doi.org/10.1016/j.autcon.2010.02.008>
- Cheng, M.-Y., & Roy, A. F. V. (2010). Evolutionary fuzzy decision model for construction management using support vector machine. *Expert Systems with Applications*, 37(8), 6061–6069. <https://doi.org/10.1016/j.eswa.2010.02.120>
- Cheng, M.-Y., Wu, Y.-W., Dan, L. T., & Van Roy, A. F. (2013). Enhanced Time-Dependent Evolutionary Fuzzy Support Vector Machines Inference Model for Cash Flow Prediction and Estimate at Completion. *International Journal of Information Technology & Decision Making*, 12(04), 679–710. <https://doi.org/10.1142/S0219622013500259>
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. <https://doi.org/10.48550/ARXIV.1412.3555>
- Dastgheib, S. R., Feylizadeh, M. R., Bagherpour, M., & Mahmoudi, A. (2022). Improving estimate at completion (EAC) cost of construction projects using adaptive neuro-fuzzy

- inference system (ANFIS). *Canadian Journal of Civil Engineering*, 49(2), 222–232.
<https://doi.org/10.1139/cjce-2020-0399>
- Desell, T., ElSaid, A., & Ororbia, A. G. (2020). An Empirical Exploration of Deep Recurrent Connections Using Neuro-Evolution. In P. A. Castillo, J. L. Jiménez Laredo, & F. Fernández De Vega (Eds.), *Applications of Evolutionary Computation* (Vol. 12104, pp. 546–561). Springer International Publishing. https://doi.org/10.1007/978-3-030-43722-0_35
- Du, J., Kim, B.-C., & Zhao, D. (2016). Cost Performance as a Stochastic Process: EAC Projection by Markov Chain Simulation. *Journal of Construction Engineering and Management*, 142(6), 04016009. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0001115](https://doi.org/10.1061/(ASCE)CO.1943-7862.0001115)
- Ead, R. (2020). *Using Monte Carlo Simulation to Evaluate Performance of Forecasting Models in Project Control*. <https://doi.org/10.7939/R3-AVYR-AG82>
- El-Omari, S., & Moselhi, O. (2011). Integrating automated data acquisition technologies for progress reporting of construction projects. *Automation in Construction*, 20(6), 699–705.
<https://doi.org/10.1016/j.autcon.2010.12.001>
- Fan, H., Jiang, M., Xu, L., Zhu, H., Cheng, J., & Jiang, J. (2020). Comparison of Long Short Term Memory Networks and the Hydrological Model in Runoff Simulation. *Water*, 12(1), 175. <https://doi.org/10.3390/w12010175>
- Fang, K., Shen, C., Kifer, D., & Yang, X. (2017). Prolongation of SMAP to Spatiotemporally Seamless Coverage of Continental U.S. Using a Deep Learning Neural Network. *Geophysical Research Letters*, 44(21). <https://doi.org/10.1002/2017GL075619>

- Feylizadeh, M. R., Hendalianpour, A., & Bagherpour, M. (2012). A fuzzy neural network to estimate at completion costs of construction projects. *International Journal of Industrial Engineering Computations*, 3(3), 477–484. <https://doi.org/10.5267/j.ijiec.2011.11.003>
- Fleming, Q. W. (2016). *Earned Value Project Management—Fourth Edition*. Project Management Institute.
- Hammad, Ahmed Mohamed. (2009). *An Integrated Framework for Managing Labour Resources Data in Industrial Construction Projects: A Knowledge Discovery in Data (KDD) Approach*. <https://doi.org/10.7939/R3-A5NH-FJ58>
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer New York. <https://doi.org/10.1007/978-0-387-84858-7>
- He, S., Du, J., & Huang, J. Z. (2017). Singular-Value Decomposition Feature-Extraction Method for Cost-Performance Prediction. *Journal of Computing in Civil Engineering*, 31(5), 04017043. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0000694](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000694)
- Heravi, G., & Eslamdoost, E. (2015). Applying Artificial Neural Networks for Measuring and Predicting Construction-Labor Productivity. *Journal of Construction Engineering and Management*, 141(10), 04015032. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0001006](https://doi.org/10.1061/(ASCE)CO.1943-7862.0001006)
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hopfe, D. H., Lee, K., & Yu, C. (2024). Short-term forecasting airport passenger flow during periods of volatility: Comparative investigation of time series vs. neural network models. *Journal of Air Transport Management*, 115, 102525. <https://doi.org/10.1016/j.jairtraman.2023.102525>

- Hossen, T., Nair, A. S., Chinnathambi, R. A., & Ranganathan, P. (2018). Residential Load Forecasting Using Deep Neural Networks (DNN). *2018 North American Power Symposium (NAPS)*, 1–5. <https://doi.org/10.1109/NAPS.2018.8600549>
- Howes, R. (2000). Improving the performance of Earned Value Analysis as a construction project management tool. *Engineering, Construction and Architectural Management*, 7(4), 399–411. <https://doi.org/10.1108/eb021162>
- İnan, T., Narbaev, T., & Hazir, Ö. (2022). A Machine Learning Study to Enhance Project Cost Forecasting. *IFAC-PapersOnLine*, 55(10), 3286–3291. <https://doi.org/10.1016/j.ifacol.2022.10.127>
- Jiang, W., & Zhang, L. (2019). Geospatial data to images: A deep-learning framework for traffic forecasting. *Tsinghua Science and Technology*, 24(1), 52–64. <https://doi.org/10.26599/TST.2018.9010033>
- Jiang, W., & Zhang, L. (2020). Edge-SiamNet and Edge-TripleNet: New Deep Learning Models for Handwritten Numeral Recognition. *IEICE Transactions on Information and Systems*, E103.D(3), 720–723. <https://doi.org/10.1587/transinf.2019EDL8199>
- Kaming, P. F., Olomolaiye, P. O., Holt, G. D., & Harris, F. C. (1997). Factors influencing construction time and cost overruns on high-rise projects in Indonesia. *Construction Management and Economics*, 15(1), 83–94. <https://doi.org/10.1080/014461997373132>
- Kareem Kamoona, K. R., & Budayan, C. (2019). Implementation of Genetic Algorithm Integrated with the Deep Neural Network for Estimating at Completion Simulation. *Advances in Civil Engineering*, 2019, 1–15. <https://doi.org/10.1155/2019/7081073>

- Kim, B.-C. (2015). Integrating Risk Assessment and Actual Performance for Probabilistic Project Cost Forecasting: A Second Moment Bayesian Model. *IEEE Transactions on Engineering Management*, 62(2), 158–170. <https://doi.org/10.1109/TEM.2015.2404935>
- Kim, B.-C., & Reinschmidt, K. F. (2011). Combination of Project Cost Forecasts in Earned Value Management. *Journal of Construction Engineering and Management*, 137(11), 958–966. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0000352](https://doi.org/10.1061/(ASCE)CO.1943-7862.0000352)
- Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer New York. <https://doi.org/10.1007/978-1-4614-6849-3>
- Le, P., & Zuidema, W. (2016). Quantifying the Vanishing Gradient and Long Distance Dependency Problem in Recursive Neural Networks and Recursive LSTMs. *Proceedings of the 1st Workshop on Representation Learning for NLP*, 87–93. <https://doi.org/10.18653/v1/W16-1610>
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- Lema, N. M., & Price, A. D. F. (1996). Construction process performance variability: Focus on labour productivity. *Building Research & Information*, 24(6), 339–350. <https://doi.org/10.1080/09613219608727554>
- Lipke, W., Zwikaël, O., Henderson, K., & Anbari, F. (2009). Prediction of project outcome. *International Journal of Project Management*, 27(4), 400–407. <https://doi.org/10.1016/j.ijproman.2008.02.009>
- Liu, Y., Mu, Y., Chen, K., Li, Y., & Guo, J. (2020). Daily Activity Feature Selection in Smart Homes Based on Pearson Correlation Coefficient. *Neural Processing Letters*, 51(2), 1771–1787. <https://doi.org/10.1007/s11063-019-10185-8>

- Mirtaheri, S. L. (2022). *Machine learning: Theory to applications* (First edition). CRC Press.
- Mohsen, O. (2021). *A Machine Learning Approach to Predict Production Time in Industrialized Building Construction*. <https://doi.org/10.7939/R3-31AQ-VW56>
- Moore, A. D., & Harwani, B. M. (2019). *Python GUI Programming - a Complete Reference Guide: Develop responsive and powerful GUI applications with PyQt and Tkinter*. Packt Publishing, Limited.
- Moselhi, O., Hegazy, T., & Fazio, P. (1991). Neural Networks as Tools in Construction. *Journal of Construction Engineering and Management*, 117(4), 606–625.
[https://doi.org/10.1061/\(ASCE\)0733-9364\(1991\)117:4\(606\)](https://doi.org/10.1061/(ASCE)0733-9364(1991)117:4(606))
- Narbaev, T., & De Marco, A. (2014). An Earned Schedule-based regression model to improve cost estimate at completion. *International Journal of Project Management*, 32(6), 1007–1018. <https://doi.org/10.1016/j.ijproman.2013.12.005>
- Nassar, Nadim Kamil. (n.d.). *An integrated framework for evaluation, forecasting and optimization of performance of construction projects*. <https://doi.org/10.7939/R3-AWDJ-RT05>
- Ottaviani, F. M., & Marco, A. D. (2022). Multiple Linear Regression Model for Improved Project Cost Forecasting. *Procedia Computer Science*, 196, 808–815.
<https://doi.org/10.1016/j.procs.2021.12.079>
- Patel, D. A., & Jha, K. N. (2015). Neural Network Model for the Prediction of Safe Work Behavior in Construction Projects. *Journal of Construction Engineering and Management*, 141(1), 04014066. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0000922](https://doi.org/10.1061/(ASCE)CO.1943-7862.0000922)

- Pewdum, W., Rujirayanyong, T., & Sooksatra, V. (2009). Forecasting final budget and duration of highway construction projects. *Engineering, Construction and Architectural Management*, 16(6), 544–557. <https://doi.org/10.1108/09699980911002566>
- Prince, S. J. D. (2023). *Understanding deep learning*. The MIT Press.
- Project Management Institute (Ed.). (2016). *Construction extension to the PMBOK guide*. Project Management Institute, Inc.
- Project Management Institute (Ed.). (2018). *The standard for portfolio management* (Fourth Edition). Project Management Institute.
- Project Management Institute (Ed.). (2019). *The standard for earned value management*. Project Management Institute, Inc.
- Rebala, G., Ravi, A., & Churiwala, S. (2019). Machine Learning Definition and Basics. In G. Rebala, A. Ravi, & S. Churiwala, *An Introduction to Machine Learning* (pp. 1–17). Springer International Publishing. https://doi.org/10.1007/978-3-030-15729-6_1
- Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.
- Shi, J., Jain, M., & Narasimhan, G. (2022). *Time Series Forecasting (TSF) Using Various Deep Learning Models*. <https://doi.org/10.48550/ARXIV.2204.11115>
- Statistics Canada. (2024). *Investment in building construction* [Dataset]. [object Object]. <https://doi.org/10.25318/3410028601-ENG>
- Tian, C., Ma, J., Zhang, C., & Zhan, P. (2018). A Deep Neural Network Model for Short-Term Load Forecast Based on Long Short-Term Memory Network and Convolutional Neural Network. *Energies*, 11(12), 3493. <https://doi.org/10.3390/en11123493>

- Tyagi, A. K., & Abraham, A. (2022). *Recurrent Neural Networks* (1st ed.). CRC Press.
<https://doi.org/10.1201/9781003307822>
- Vapnik, V. N. (2000). *The nature of statistical learning theory* (2nd ed). Springer.
- Vu, N.-T., & Do, K.-U. (2021). Prediction of Ammonium Removal by Biochar Produced From Agricultural Wastes Using Artificial Neural Networks: Prospects and Bottlenecks. In *Soft Computing Techniques in Solid Waste and Wastewater Management* (pp. 455–467). Elsevier. <https://doi.org/10.1016/B978-0-12-824463-0.00012-4>
- Wauters, M., & Vanhoucke, M. (2014). Support Vector Machine Regression for project control forecasting. *Automation in Construction*, 47, 92–106.
<https://doi.org/10.1016/j.autcon.2014.07.014>
- Wen, X., & Li, W. (2023). Time Series Prediction Based on LSTM-Attention-LSTM Model. *IEEE Access*, 11, 48322–48331. <https://doi.org/10.1109/ACCESS.2023.3276628>
- Wu, N., Green, B., Ben, X., & O'Banion, S. (2020). *Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case* (arXiv:2001.08317). arXiv.
<http://arxiv.org/abs/2001.08317>
- Xu, J., Liu, J., Yu, S., Xu, K., & Zhang, T. (2024). Real-time temperature prediction of lunar regolith drilling based on ATT-Bi-LSTM network. *International Journal of Heat and Mass Transfer*, 218, 124783. <https://doi.org/10.1016/j.ijheatmasstransfer.2023.124783>
- Xu, Y., Zhou, Y., Sekula, P., & Ding, L. (2021). Machine learning in construction: From shallow to deep learning. *Developments in the Built Environment*, 6, 100045.
<https://doi.org/10.1016/j.dibe.2021.100045>
- Yamak, P. T., Yujian, L., & Gadosey, P. K. (2019). A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting. *Proceedings of the 2019 2nd International Conference*

on Algorithms, Computing and Artificial Intelligence, 49–55.

<https://doi.org/10.1145/3377713.3377722>

Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11), 3212–3232. <https://doi.org/10.1109/TNNLS.2018.2876865>

Appendix A: Python Scrip for Deep Learning Model

1.0 Import Libraries

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import GRU
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import TimeSeriesSplit
import numpy as np
from sklearn.metrics import r2_score
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

##2.0 Get Data Using Pandas

```
dataset = read_csv('Concretedataset.csv', header=0)
values = dataset.values
dataset
#Specify columns to plot
groups = [0, 1, 2, 3,4,5]
i = 1
# Plot each column
pyplot.figure()
for group in groups:
    pyplot.subplot(len(groups), 1, i)
    pyplot.plot(values[:, group], color='g')
    pyplot.title(dataset.columns[group], y=0.5, loc='right')
    i += 1
pyplot.show()
```

3.0 Data Preprocessing

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
```

```

cols, names = list(), list()

# input sequence (t-n, ... t-1)
for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
# forecast sequence (t, t+1, ... t+n)
for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
# Put it all together
agg = concat(cols, axis=1)
agg.columns = names
# Drop rows with NaN values
if drop an:
    agg.dropna(inplace=True)
return agg
#Ensure all data is float
values = values.astype('float32')

```

4.0 Normalization

```

scaler = MinMaxScaler()
scaled = scaler.fit_transform(values)

```

5.0 Define Parameters

```

n_past = 3
n_output = 1
n_features = 6 #features include output

```

6.0 Convert to Supervised Learning, Splid Data, And Reshape Data.

```

# frame as supervised learning
reframed = series_to_supervised(scaled, n_past, n_output)
print(reframed.shape)
print(reframed)
# Split into train and validation sets
values = reframed.values

```

```

# Define the number of splits
n_train_weeks = 336
train = values[:n_train_weeks, :]
test = values[n_train_weeks:, :]

```

```

# Split into input and outputs
n_obs = n_past * n_features
train_X, train_y = train[:, :n_obs], train[:, -(n_features)]
test_X, test_y = test[:, :n_obs], test[:, -(n_features)]

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_past, n_features))
test_X = test_X.reshape((test_X.shape[0], n_past, n_features))
print('Train_X.shape=', train_X.shape, '-', 'Train_y.shape=', train_y.shape)
print('Test_X.shape=', test_X.shape, '-', 'Test_y.shape=', test_y.shape)

```

7.0 Setup the Gru Model

```

model = Sequential ()
model.add(GRU(32, activation='relu', input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
model.add(GRU(32, activation='relu', return_sequences=False))
model.add(Dropout(0.01))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mae')

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=50, verbose=1, restore_best_weights=True)
checkpoint_filepath = '/tmp/model_lstm_checkpoint.h5'
model_checkpoint_callback = ModelCheckpoint(filepath=checkpoint_filepath, monitor='val_loss', save_best_only=True)

```

8.0 Model development: Training and Validation using split datasets

```

history = model.fit(train_X, train_y, epochs=350, batch_size=64, validation_data=(test_X, test_y), verbose=2,
                    shuffle=False, callbacks=[early_stopping, model_checkpoint_callback])

# plot history
pyplot.plot(history.history['loss'], label='Training loss')
pyplot.plot(history.history['val_loss'], label='Validation loss')

# Add labels to axes
plt.xlabel('Epoch')
plt.ylabel('Loss (Normalized MAE)')
pyplot.legend()
pyplot.show()
y_train_predicted = model.predict(train_X)

# Make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_past*n_features))

# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)

```

```

inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:,-(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

```

9.0 Results and Performance Metrics

```

df_result = pd.DataFrame({'Actual': inv_y, 'Forecast': inv_yhat, 'Error': ((inv_y - inv_yhat) / inv_y)})
# Calculate RMSE, MAE, R2, and MAPE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
mae = mean_absolute_error(inv_y, inv_yhat)
r2 = r2_score(inv_y, inv_yhat)
mape = np.mean(np.abs((inv_y - inv_yhat) / inv_y)) * 100
# Display metrics
print(f'R2: {r2}')
print(f'MAE: {mae}')
# Create a bar plot for RMSE, MAE, R2, and MAPE
metrics = ['R2', 'MAE']
values = [r2, mae]
plt.bar(metrics, values)
plt.xlabel('Metrics')
plt.ylabel('Value')
plt.title('Performance Metrics Comparison')
plt.show()
# Plot actual vs. predicted values with a line plot
plt.figure(figsize=(12, 6))
plt.plot(inv_y, label='Actual', linestyle='-', marker='o')
plt.plot(inv_yhat, label='Predicted', linestyle='-', marker='o')
plt.xlabel('Data Points')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()

```

10.0 Saving Model

```

model.save("model2GRU.h5")
# load and evaluate a saved model
from numpy import loadtxt
from keras.models import load_model

# load model
model1re = load_model('model2GRU.h5')
# Summarize model.
model1re.summary()

```

```

model1re.summary()
df_result.to_csv('df_resultGRU.csv', index=False)
# Calculate RMSE and MAE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
mae = mean_absolute_error(inv_y, inv_yhat)
# Plot the actual vs. predicted values as a bar plot
plt.figure(figsize=(12, 6))
plt.bar(np.arange(len(inv_y)), inv_y, width=0.4, label='Actual', align='center', alpha=0.7)
plt.bar(np.arange(len(inv_y)) + 0.4, inv_yhat, width=0.4, label='Predicted', align='edge', alpha=0.7)
plt.xlabel('Data Points')
plt.ylabel('Value')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()

```


Appendix B: Python Code for GUI Creation, SQLite, and Model Deployment

IMPORTING LIBRARIES AND DEFINING FUNCTIONS

```
import sqlite3
import pandas as pd
import numpy as np
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import DateEntry
from datetime import datetime
from numpy import concatenate
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
import matplotlib.pyplot as plt
from warnings import filterwarnings
filterwarnings('ignore')
def convert_to_float(value, default=None):
    try:
        result = float(value)
    except ValueError:
        result = 0
    return round(result, 3)
def update_sum_entries(entry_sum, *entries):
    states = [entry["state"] for entry in entries]
    entry_sum_state = entry_sum["state"]
    [entry.config(state="normal") for entry in entries]
    summ = sum([convert_to_float(entry.get()) for entry in entries])
    [entry.config(state=state) for entry, state in zip(entries, states)]
    entry_sum.config(state="normal")
    entry_sum.delete(0, tk.END)
    entry_sum.insert(0, round(summ, 3))
    entry_sum.config(state=entry_sum_state)
class ProjectCostManagementApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Project Cost Forecasting")
        self.root.resizable(False, False)
        self.db = 'project_data.db'
        self.project_setup_labels = ["Project Name", "Project ID", "Portfolio Name", "Project Cost Center",
                                     "Contract Type", "Contract Number", "Owner Company Name", "Owner Contact Name",
                                     "Project Type", "Project Location", "Contract Amount ($)", "Contract Duration (days)",
                                     "Project Manager", "Project Controller", "Description"]
        self.project_baseline_labels = ["Budget at Completion ($)", "Start Date (mm/dd/yyyy)", "Finish Date (mm/dd/yyyy)",
                                         "Schedule Baseline (days)"]
        self.wp_setup_labels = ["Work Package Name", "Work Package Code", "Work Package Type", "Key Quantity",
                                "Unit of Measure", "Description"]
        self.wp_baseline_labels = ["Budget at Completion ($)", "Start Date (mm/dd/yyyy)", "Finish Date (mm/dd/yyyy)", "Duration
(days)",
                                   "Labour Cost ($)", "Equipment Cost ($)", "Material Cost ($)", "Subcontractor Cost ($)",
                                   "Quantity"]
```

```

self.actual_cost_details_labels = ["Labour Cost ($) ", "Equipment Cost ($) ", "Material Cost ($) ", "Subcontractor Cost ($) "]
self.dict_combobox = {"Work Package Name": ["Concrete", "Piping (HDPE)", "Baroe Fill", "Cut"],
    "Contract Type": ["Lump Sum", "Time and Materials", "Unit Price", "Cost Plus"],
    "Project Type": ["Residential & Commercial", "Infrastructure", "Industrial"],
    "Unit of Measure": ["m³", "yd³", "m²", "kg", "m", "per unit"],
    "Key Quantity": ["Volume of Concrete", "Weight of Structural Steel", "Area of Masonry", "Length of Piping",
"Area of Roofing", "Volume of Excavation", "Volume of fill", "Area of Flooring", "Length of Electrical Wiring"],
    "Work Package Type": ["EWP-Engineering", "CWP-Construction", "PWP-Procurement"],
    "Status at Completion": ["Completed", "On Hold", "In Progress", "Nor Yet Started"]}
self.fonts = {"label1": {"anchor": "w", "font": ("didot", 9)},
    "label2": {"anchor": "w", "font": ("didot", 10)},
    "entry1": {"font": ("didot", 9)},
    "combobox1": {"font": ("didot", 9)},
    "labelframe1": {"font": ("didot", 11, "bold"), "foreground": "deeppink4", "borderwidth": 3},
    "labelframe2": {"font": ("didot", 10, "bold"), "foreground": "teal"},
    "button1": {"font": ("didot", 9, "bold"), "bg": "palegreen", "fg": "gray25"},
    "button2": {"font": ("didot", 9, "bold"), "bg": "khaki", "fg": "gray30"},
    "button3": {"font": ("didot", 9, "bold"), "bg": "pale green", "fg": "gray25"},
    "dateentry1": {"font": ("didot", 9), "date_pattern": "mm/dd/yyyy", "background": "darkblue", "foreground": "white",
"borderwidth": 2},
    }
self.create_menu_navigation_frame()
self.frame = None
self.frames = {"Project Setup": self.create_project_setup_frame,
    "Update Project": self.create_update_project_info,
    "Update Work Package": self.create_update_wp_info,
    "Weekly Progres": self.create_project_cost_status_frame,
    "Project Forecasting": self.create_project_forecasting_frame,}
self.show_frame("Project Setup")
def show_frame(self, frame_name):
    if self.frame:
        self.frame.grid_forget()
    self.frame = self.frames[frame_name]()
    self.frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
def clear_frame(self, frame):
    """Clear all widgets from a frame if it's not None."""
    if frame is not None:
        for widget in frame.winfo_children():
            widget.destroy()

```

CREATE THE NAVIGATION MENU FRAME

```

def create_menu_navigation_frame(self):
    self.menu_navigation_frame = tk.LabelFrame(self.root, text="NAVIGATION MENU", font=("Didot", 11, "bold"),
foreground="deeppink4", borderwidth=3)
    self.menu_navigation_frame.grid(row=0, column=0, padx=5, pady=20, sticky="nsw")
    ## Project Setup
    project_setup_frame = tk.LabelFrame(self.menu_navigation_frame, text="Project Setup", **self.fonts["labelframe2"])
    project_setup_frame.grid(row=0, padx=10, pady=10, sticky="nsew")
    self.setup_project = tk.Button(project_setup_frame, text="Project Creation", bg="#DCDCDC", command=lambda:
self.show_frame("Project Setup"))
    self.setup_project.grid(row=0, padx=30, pady=8, sticky="w")
    ## Project Tracking
    project_tracking_frame = tk.LabelFrame(self.menu_navigation_frame, text="Project Tracking", **self.fonts["labelframe2"])

```

```

project_tracking_frame.grid(row=1, padx=10, pady=10, sticky="nsew")
self.update_project_info = tk.Button(project_tracking_frame, text="Project Update", bg="#DCDCDC", command=lambda:
self.show_frame("Update Project"))
self.update_project_info.grid(row=0, padx=30, pady=8, sticky="w")
self.update_wp_info = tk.Button(project_tracking_frame, text="Work Package Update", bg="#DCDCDC", command=lambda:
self.show_frame("Update Work Package"))
self.update_wp_info.grid(row=1, padx=30, pady=8, sticky="w")
self.project_cost_status_button = tk.Button(project_tracking_frame, text="Weekly Progress", bg="#DCDCDC",
command=lambda: self.show_frame("Weekly Progres"))
self.project_cost_status_button.grid(row=2, padx=30, pady=8, sticky="w")
## Project Forecasting
project_forecasting_frame = tk.LabelFrame(self.menu_navigation_frame, text="Project Forecasting",
**self.fonts["labelframe2"])
project_forecasting_frame.grid(row=2, padx=10, pady=10, sticky="nsew")
self.project_forecasting = tk.Button(project_forecasting_frame, text="Project Forecasting", bg="#DCDCDC",
command=lambda: self.show_frame("Project Forecasting"))
self.project_forecasting.grid(row=0, padx=30, pady=8, sticky="w")

```

1.0 PROJECT SETUP

1.1 PROJECT SETUP FRAME

```

def create_project_setup_frame(self):
self.project_setup_frame = tk.LabelFrame(self.root, text="PROJECT SETUP", **self.fonts["labelframe1"])
self.project_setup_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
self.project_setup_widgets = {}
for i, label in enumerate(self.project_setup_labels):
row = i // 2
col = 0 if i % 2 == 0 else 2
label_widget = tk.Label(self.project_setup_frame, text=f"{label}:", **self.fonts["label1"])
label_widget.grid(row=row, column=col, padx=10, pady=5, sticky="ew")
if label in self.dict_combobox:
widget = ttk.Combobox(self.project_setup_frame, values=self.dict_combobox[label], state="readonly",
**self.fonts["combobox1"])
widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
widget.set(self.dict_combobox[label][0])
elif label == "Description":
widget = tk.Text(self.project_setup_frame, height=3)
widget.grid(row=8, column=0, columnspan=4, padx=10, pady=5, sticky="ew")
widget.configure(bg="azure")
elif label.endswith('(mm/dd/yyyy)'):
widget = DateEntry(self.project_setup_frame, **self.fonts["dateentry1"])
widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
else:
widget = tk.Entry(self.project_setup_frame, **self.fonts["entry1"])
widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
self.project_setup_widgets[label] = widget

```

Adds a note label below the description text widget

```

note_label_text = "NOTE: Project Manager or Project Controller only, have permissions to Submit this Form."
note_label = tk.Label(self.project_setup_frame, text=note_label_text, font=("Didot", 8, "italic"))
note_label.grid(row=11, column=0, columnspan=4, padx=10, pady=5, sticky="w")

```

Adds save and work package buttons

```

save_project_button = tk.Button(self.project_setup_frame, text="Save Project", command=self.save_project_data,
bg="#9370DB", fg="white", font=("Didot", 10, "bold"))

```

```

save_project_button.grid(row=10, column=2, padx=10, pady=5, sticky="se")

wp_setup_button = tk.Button(self.project_setup_frame, text="Add Work Package", command=self.create_wp_setup_window,
bg="cadetblue1", fg="gray30",font=("Didot", 10,"bold"))
wp_setup_button.grid(row=10, column=3, padx=10, pady=5, sticky="sw")

# Original Baseline subframe
ob_frame = tk.LabelFrame(self.project_setup_frame, text="Original Baseline", **self.fonts['labelframe2'])
ob_frame.grid(row=9, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")
self.project_ob_widgets = {}

# Adjusting grid layout for two columns
for i, label in enumerate(self.project_baseline_labels):
    ob_row = i // 2
    ob_col = 0 if i % 2 == 0 else 2
    label_widget = tk.Label(ob_frame, text=f"{label}:", **self.fonts["label1"])
    label_widget.grid(row=ob_row, column=ob_col, padx=10, pady=5, sticky="ew")
    if label.endswith('(mm/dd/yyyy)'):
        widget = DateEntry(ob_frame, **self.fonts["dateentry1"])
    else:
        widget = tk.Entry(ob_frame)
    widget.grid(row=ob_row, column=ob_col + 1, padx=10, pady=5, sticky="ew")
    self.project_ob_widgets[label] = widget
return self.project_setup_frame

```

1.1.1 SAVE PROJECT DATA

```

def save_project_data(self):
    project_data = {}
    for label in self.project_setup_labels:
        try:
            project_data[label.replace(" ", "_").lower()] = self.project_setup_widgets[label].get().strip()
        except:
            project_data[label.replace(" ", "_").lower()] = self.project_setup_widgets[label].get("1.0",'end-1c').strip()
    bs_data = {label.replace(" ", "_").lower(): self.project_ob_widgets[label].get().strip() for label in self.project_baseline_labels}

    # Check for incomplete fields
    incomplete_fields = [label for label, value in project_data.items() if not value.strip()]
    if incomplete_fields:
        messagebox.showerror("Project Incomplete Fields", "Please complete the following fields: " + " ".join(incomplete_fields))
        return
    incomplete_fields = [label for label, value in bs_data.items() if not value.strip()]
    if incomplete_fields:
        messagebox.showerror("Project Original Baseline Incomplete Fields", "Please complete the following fields: " + " ".join(incomplete_fields))
        return
    conn = sqlite3.connect(self.db)
    kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
    mod_date = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
    project_data_df = pd.DataFrame(project_data, index=[0], columns=project_data.keys())
    project_data_df['mod_date'] = mod_date
    bs_data_df = pd.DataFrame(bs_data, index=[0], columns=bs_data.keys())
    bs_data_df['project_id'] = project_data['project_id']
    bs_data_df['mod_date'] = mod_date
    project_data_df.fillna("").to_sql("Projects", **kwargs)

```

```

bs_data_df.fillna("").to_sql("Project_original_baselines", **kwargs)
bs_data_df.fillna("").to_sql("Project_current_baselines", **kwargs)
conn.close()
messagebox.showinfo("Data Saved", "Data has been saved.")

```

1.1.2 ADD WORK PACKAGE

```

def create_wp_setup_window(self):
    wp_setup_window = tk.Toplevel(self.root)
    wp_setup_window.resizable(False, False)
    wp_setup_window.title("Work Package Details")
    main_frame = tk.LabelFrame(wp_setup_window, text="WORK PACKAGE SETUP", **self.fonts["labelframe1"])
    main_frame.grid(row=0, padx=20, pady=20, sticky="nsew")

    # Populate ComboBox with project names
    conn = sqlite3.connect(self.db)
    project_names = pd.read_sql('SELECT DISTINCT "project_name" FROM "Projects"', conn)['project_name'].to_list()
    conn.close()
    project_label = tk.Label(main_frame, text="Project Name:", **self.fonts["label1"])
    project_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")
    self.project_combobox = ttk.Combobox(main_frame, values=project_names, state="readonly", **self.fonts["combobox1"])
    self.project_combobox.grid(row=1, column=1, columnspan=3, padx=10, pady=5, sticky="ew")
    self.wp_setup_widgets = {}
    for i, label in enumerate(self.wp_setup_labels):
        row = i // 2
        col = 0 if i % 2 == 0 else 2
        label_widget = tk.Label(main_frame, text=f"{label}:", **self.fonts["label1"])
        if label in self.dict_combobox:
            label_widget.grid(row=row + 2, column=col, padx=10, pady=2, sticky="ew")
            widget = ttk.Combobox(main_frame, values=self.dict_combobox[label], state="readonly", **self.fonts["combobox1"])
            widget.grid(row=row + 2, column=col + 1, padx=10, pady=5, sticky="ew")
            widget.set(self.dict_combobox[label][0])
        elif label == "Description":
            label_widget.grid(row=row + 3, column=0, padx=10, pady=2, sticky="ew")
            widget = tk.Text(main_frame, height=3)
            widget.grid(row=row + 4, column=0, columnspan=4, padx=10, pady=2, sticky="ew")
            widget.configure(bg="azure")
        elif label.endswith('(mm/dd/yyyy)'):
            label_widget.grid(row=row + 2, column=col, padx=10, pady=2, sticky="ew")
            widget = DateEntry(main_frame, **self.fonts["dateentry1"])
            widget.grid(row=row + 2, column=col + 1, padx=10, pady=5, sticky="ew")
        else:
            label_widget.grid(row=row + 2, column=col, padx=10, pady=2, sticky="ew")
            widget = tk.Entry(main_frame, **self.fonts["entry1"])
            widget.grid(row=row + 2, column=col + 1, padx=10, pady=2, sticky="ew")
        self.wp_setup_widgets[label] = widget

    # Original Baseline Frame inside main_frame
    ob_frame = tk.LabelFrame(main_frame, text="Original Baseline", **self.fonts["labelframe2"])
    ob_frame.grid(row=row + 5, column=0, columnspan=4, padx=10, pady=5, sticky="ew")

    # Widgets for Original Baseline Frame
    self.wp_ob_widgets = {}
    for i, label in enumerate(self.wp_baseline_labels):
        ob_row = i // 2

```

```

ob_col = 0 if i % 2 == 0 else 2
label_widget = tk.Label(ob_frame, text=f"{label}:", **self.fonts["label1"])
label_widget.grid(row=ob_row, column=ob_col, sticky="w", padx=10, pady=5)
if label.endswith('(mm/dd/yyyy)'):
    widget = DateEntry(ob_frame, **self.fonts["dateentry1"])
else:
    widget = tk.Entry(ob_frame, **self.fonts["entry1"])
    widget.grid(row=ob_row, column=ob_col + 1, sticky="ew", padx=10, pady=5)
self.wp_ob_widgets[label] = widget
save_button = tk.Button(main_frame, text="Save Work Package", command=self.save_wp_data, bg="#9370DB", fg="white",
font=("Didot", 10, "bold"))
save_button.grid(row=row + 6, column=0, columnspan=4, pady=10)
note_label_text = "NOTE: Project Manager or Project Controller only, have permissions to Submit this Form."
note_label = tk.Label(main_frame, text=note_label_text, font=("Didot", 8, "italic"))
note_label.grid(row=row + 7, column=0, columnspan=4, padx=10, pady=5, sticky="w")

```

1.1.3 SAVE WORK PACKAGE

```

def save_wp_data(self):

    # Gather data from the entries and other components
    wp_data = {}
    for label in self.wp_setup_labels:
        try:
            wp_data[label.replace(" ", "_").lower()] = self.wp_setup_widgets[label].get().strip()
        except:
            wp_data[label.replace(" ", "_").lower()] = self.wp_setup_widgets[label].get("1.0", 'end-1c').strip()
    bs_data = {label.replace(" ", "_").lower(): self.wp_ob_widgets[label].get().strip() for label in self.wp_baseline_labels}

    # Check for incomplete fields
    incomplete_fields = [label for label, value in wp_data.items() if not value.strip()]
    if incomplete_fields:
        messagebox.showerror("Work Package Incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields))
        return
    incomplete_fields = [label for label, value in bs_data.items() if not value.strip()]
    if incomplete_fields:
        messagebox.showerror("Work Package Original Baseline Incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields))
        return
    conn = sqlite3.connect(self.db)
    kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
    wp_data_df = pd.DataFrame(wp_data, index=[0], columns=wp_data.keys())
    bs_data_df = pd.DataFrame(bs_data, index=[0], columns=bs_data.keys())
    self.project_name = self.project_combobox.get()
    self.project_id = pd.read_sql(f"SELECT \"project_id\" FROM \"Projects\" WHERE \"project_name\"='{self.project_name}' ORDER
BY \"mod_date\" DESC LIMIT 1", conn).iloc[0, 0]
    mod_date = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
    wp_data_df['project_id'] = self.project_id
    wp_data_df['mod_date'] = mod_date
    bs_data_df['work_package_code'] = wp_data['work_package_code']
    bs_data_df['mod_date'] = mod_date
    wp_data_df.fillna("").to_sql("Work_packages", **kwargs)
    bs_data_df.fillna("").to_sql("Work_package_original_baselines", **kwargs)
    bs_data_df.fillna("").to_sql("Work_package_current_baselines", **kwargs)

```

```

conn.close()
messagebox.showinfo("Success", "Work Package data saved successfully")

```

2.0 PROJECT TRACKING

2.1 PROJECT UPDATE FRAME

```

def create_update_project_info(self):
    self.update_project_frame = tk.LabelFrame(self.root, text="PROJECT UPDATE", **self.fonts["labelframe1"])
    self.update_project_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
    project_name_label = tk.Label(self.update_project_frame, text="Project Name:", **self.fonts["label1"])
    project_name_label.grid(row=0, column=0, padx=10, pady=5, sticky="w")

    # Populate ComboBox with project names
    conn = sqlite3.connect(self.db)
    project_names = pd.read_sql('SELECT DISTINCT "project_name" FROM "Projects"', conn)['project_name'].to_list()
    conn.close()
    self.project_combobox = ttk.Combobox(self.update_project_frame, values=project_names, state="readonly",
    **self.fonts["combobox1"])
    self.project_combobox.set("Select Project Name")
    self.project_combobox.grid(row=0, column=1, padx=20, pady=10, sticky="w")
    show_info_button = tk.Button(self.update_project_frame, text="Show Project Information",
    command=self.show_project_data, bg="cadetblue1", fg="gray30", font=("Didot", 9, "bold"))
    show_info_button.grid(row=0, column=2, padx=10, pady=10, sticky="w")
    self.update_cb_button = tk.Button(self.update_project_frame, text="Update Original Baseline",
    command=self.create_cb_frame, state="disabled", **self.fonts["button2"])
    self.update_cb_button.grid(row=0, column=3, padx=10, pady=10, sticky="nw")
    return self.update_project_frame

```

2.1.1 CURRENT BASELINE

```

def create_cb_frame(self):
    self.cb_window = tk.Toplevel(self.root)
    self.cb_window.resizable(False, False)
    self.cb_window.title("Current Baseline")
    project_name_label = tk.Label(self.cb_window, text=f"Project Name: {self.project_name}", **self.fonts["label2"])
    project_name_label.grid(row=0, column=0, padx=10, pady=2, sticky="w")
    project_id_label = tk.Label(self.cb_window, text=f"Project ID: {self.project_id}", **self.fonts["label2"])
    project_id_label.grid(row=0, column=1, padx=10, pady=2, sticky="w")

    # Create a frame for current baseline fields
    cb_frame = tk.LabelFrame(self.cb_window, text="Current Baseline", **self.fonts["labelframe2"])
    cb_frame.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky="ns")
    self.cb_widgets = {}
    conn = sqlite3.connect(self.db)
    cb_data = pd.read_sql(f'SELECT * FROM "Project_current_baselines" WHERE "project_id"="{self.project_id}" ORDER BY "mod_date" DESC LIMIT 1', conn)
    conn.close()
    for i, label in enumerate(self.project_baseline_labels):
        label_widget = tk.Label(cb_frame, text=f"{label}:", **self.fonts["label1"])
        label_widget.grid(row=i, column=0, padx=10, pady=5, sticky="ew")
        value = cb_data.loc[0, label.replace(' ', '_').lower()]
        if label in self.dict_combobox:
            widget = ttk.Combobox(cb_frame, values=self.dict_combobox[label], state="readonly", **self.fonts["combobox1"])
            widget.set(value)
        elif label.endswith('(mm/dd/yyyy)'):

```

```

        widget = DateEntry(cb_frame, **self.fonts["dateentry1"])
        widget.delete(0, tk.END)
        widget.insert(tk.END, value)
    else:
        widget = tk.Entry(cb_frame, **self.fonts["entry1"])
        widget.insert(tk.END, value)
    widget.grid(row=i, column=1, padx=10, pady=5, sticky="ew")
    self.cb_widgets[label] = widget

# Add a save button
    save_button = tk.Button(self.cb_window, text="Save Current Baseline", command=self.save_cb, bg="#9370DB",
fg="white", font=("Didot", 10, "bold"))
    save_button.grid(row=3, columnspan=2, padx=10, pady=10)
    def save_cb(self):

# Validate and save the current baseline data
    cb_data = {label.replace(' ', '_').lower(): self.cb_widgets[label].get().strip() for label in self.project_baseline_labels}

# Check for incomplete fields
    incomplete_fields = [label for label, value in cb_data.items() if not value]
    if incomplete_fields:
        messagebox.showerror("Incomplete Fields", "Please complete the following fields: " + ", ".join(incomplete_fields))
        return
    conn = sqlite3.connect(self.db)
    kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
    cb_data_df = pd.DataFrame(cb_data, index=[0], columns=cb_data.keys())
    cb_data_df['project_id'] = self.project_id
    cb_data_df['mod_date'] = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
    cb_data_df.fillna("").to_sql("Project_current_baselines", **kwargs)
    conn.close()
    messagebox.showinfo("Data Saved", "Data has been saved.")

```

2.1.2 SHOW PROJECT INFORMATION

```

def show_project_data(self):
    selected_project_name = self.project_combobox.get()
    if selected_project_name == "Select Project Name" or (not selected_project_name):
        messagebox.showwarning("Select Project", "Please select a project Name to view its information.")
        return
    self.project_name = selected_project_name
    conn = sqlite3.connect(self.db)
    project_data = pd.read_sql(f"SELECT * FROM \"Projects\" WHERE \"project_name\"='{selected_project_name}' ORDER BY
\"mod_date\" DESC LIMIT 1", conn)
    self.project_id = project_data.loc[0, 'project_id']
    project_ob = pd.read_sql(f"SELECT * FROM \"Project_original_baselines\" WHERE \"project_id\"='{self.project_id}' ORDER BY
\"mod_date\" DESC LIMIT 1", conn)
    conn.close()
    project_info_frame = tk.LabelFrame(self.update_project_frame, text="Project Information", **self.fonts["labelframe2"])
    project_info_frame.grid(row=1, column=0, columnspan=5, padx=10, pady=10, sticky="nsew")
    self.project_info_widgets = {}

# Create labels and entry widgets for each field
    for i, label in enumerate(self.project_setup_labels):
        row = i // 2
        col = 0 if i % 2 == 0 else 2

```



```

label_widget = tk.Label(project_info_frame, text=f"{label}:", **self.fonts["label1"])
label_widget.grid(row=row, column=col, padx=10, pady=5, sticky="w")
value = project_data.loc[0, label.replace(' ', '_').lower()]
if label in self.dict_combobox:
    widget = ttk.Combobox(project_info_frame, values=self.dict_combobox[label], **self.fonts["combobox1"])
    widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
    widget.set(value)
    widget.config(state="disabled")
elif label == "Description":
    widget = tk.Text(project_info_frame, height=3)
    widget.grid(row=row + 1, column=0, columnspan=4, padx=10, pady=5, sticky="ew")
    widget.insert(tk.END, value)
    widget.config(state="disabled")
else:
    widget = tk.Entry(project_info_frame, **self.fonts["entry1"])
    widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
    widget.insert(tk.END, value)
    widget.config(state="readonly")
self.project_info_widgets[label] = widget

# Add Original Baseline information
ob_frame = tk.LabelFrame(project_info_frame, text="Original Baseline", **self.fonts["labelframe2"])
ob_frame.grid(row=row + 2, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")
self.project_ob_widgets = {}
for i, label in enumerate(self.project_baseline_labels):
    ob_row = i // 2
    ob_col = 0 if i % 2 == 0 else 2
    label_widget = tk.Label(ob_frame, text=f"{label}:", **self.fonts["label1"])
    label_widget.grid(row=ob_row, column=ob_col, padx=10, pady=2, sticky="w")
    value = project_ob.loc[0, label.replace(' ', '_').lower()]
    if label.endswith('(mm/dd/yyyy)'):
        widget = DateEntry(ob_frame, **self.fonts["dateentry1"])
        widget.delete(0, tk.END)
        widget.insert(tk.END, value)
        widget.config(state="disabled")
    else:
        widget = tk.Entry(ob_frame, **self.fonts["entry1"])
        widget.insert(tk.END, value)
        widget.config(state="readonly")
    self.project_ob_widgets[label] = widget
    widget.grid(row=ob_row, column=ob_col + 1, padx=10, pady=2, sticky="ew")
self.edit_project_button = tk.Button(self.update_project_frame, text="Edit Project Information",
command=self.edit_project_information,
**self.fonts["button1"])
self.edit_project_button.grid(row=3, column=0, padx=10, pady=10, sticky="e")
self.update_project_button = tk.Button(self.update_project_frame, text="Update Project Information",
command=self.update_project_information,
**self.fonts["button2"])
self.update_project_button.grid(row=3, column=1, padx=10, pady=10, sticky="e")

# Enable the "Update Original Baseline" button
self.update_project_button.config(state="disabled")
self.update_cb_button.config(state="normal")
def edit_project_information(self):

```

```

        [self.project_info_widgets[label].config(state="normal") for label in self.project_setup_labels if label not in ("Project Name",
"Project ID")]
        [self.project_ob_widgets[label].config(state="normal") for label in self.project_baseline_labels]
        self.update_project_button.config(state="normal")
    def update_project_information(self):

        # Store the project data
        [self.project_info_widgets[label].config(state="normal") for label in ("Project Name", "Project ID")]
        project_data = {}
        for label in self.project_setup_labels:
            if label == "Description":
                project_data[label.replace(" ", "_").lower()] = self.project_info_widgets[label].get("1.0", 'end-1c').strip()
            else:
                project_data[label.replace(" ", "_").lower()] = self.project_info_widgets[label].get().strip()

        bs_data = {label.replace(" ", "_").lower(): self.project_ob_widgets[label].get().strip() for label in self.project_baseline_labels}

        # Check for incomplete fields
        incomplete_fields = [label for label, value in project_data.items() if not value.strip()]
        if incomplete_fields:
            messagebox.showerror("Project Incomplete Fields", "Please complete the following fields: " + ", ".join(incomplete_fields))
            return
        incomplete_fields = [label for label, value in bs_data.items() if not value.strip()]
        if incomplete_fields:
            messagebox.showerror("Project Original Baseline Incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields))
            return
        conn = sqlite3.connect(self.db)
        kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
        mod_date = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
        project_data_df = pd.DataFrame(project_data, index=[0], columns=project_data.keys())
        project_data_df['mod_date'] = mod_date
        bs_data_df = pd.DataFrame(bs_data, index=[0], columns=bs_data.keys())
        bs_data_df['project_id'] = project_data['project_id']
        bs_data_df['mod_date'] = mod_date
        project_data_df.fillna("").to_sql("Projects", **kwargs)
        bs_data_df.fillna("").to_sql("Project_original_baselines", **kwargs)
        bs_data_df.fillna("").to_sql("Project_current_baselines", **kwargs)
        conn.close()
        messagebox.showinfo("Data Saved", "Data has been saved.")
        self.update_project_button.config(state="disabled")
        [self.project_info_widgets[label].config(state="disabled") if (label == "Description" or label in self.dict_combobox) else
self.project_info_widgets[label].config(state="readonly") for label in self.project_setup_labels]
        [self.project_ob_widgets[label].config(state="disabled") if (label.endswith('(mm/dd/yyyy)') or label in self.dict_combobox)
else self.project_ob_widgets[label].config(state="readonly") for label in self.project_baseline_labels]

```

2.2 WORK PACKAGE UPDATE FRAME

```

def create_update_wp_info(self):
    self.update_wp_info_frame = tk.LabelFrame(self.root, text="WORK PACKAGE UPDATE", **self.fonts["labelframe1"])
    self.update_wp_info_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")

```

Project Name ComboBox

```

project_name_label = tk.Label(self.update_wp_info_frame, text="Project Name:", **self.fonts["label1"])

```

```

project_name_label.grid(row=0, column=0, padx=10, pady=10, sticky="w")

# Populate ComboBox with project names
conn = sqlite3.connect(self.db)
project_names = pd.read_sql('SELECT DISTINCT "project_name" FROM "Projects"', conn)['project_name'].to_list()
conn.close()
self.project_name_combobox = ttk.Combobox(self.update_wp_info_frame, values=project_names, state="readonly",
**self.fonts["combobox1"])
self.project_name_combobox.set("Select Project Name")
self.project_name_combobox.grid(row=0, column=1, padx=10, pady=10, sticky="w")
self.project_name_combobox.bind("<<ComboboxSelected>>", self.update_wp_names)

# Work Package Name ComboBox
wp_name_label = tk.Label(self.update_wp_info_frame, text="Work Package Name:", **self.fonts["label1"])
wp_name_label.grid(row=1, column=0, padx=10, pady=10, sticky="w")
self.wp_name_combobox = ttk.Combobox(self.update_wp_info_frame, **self.fonts["combobox1"])
self.wp_name_combobox.grid(row=1, column=1, padx=10, pady=10, sticky="w")
self.wp_name_combobox.config(state="disabled")

# Show Work Package Button
show_wp_button = tk.Button(self.update_wp_info_frame, text="Show WP Information", command=self.show_wp_details,
bg="cadetblue1", fg="gray25", font=("Didot", 9, 'bold'))
show_wp_button.grid(row=0, column=2, padx=10, pady=5, sticky="e")

#Update Original Baseline Button
self.update_cb_button = tk.Button(self.update_wp_info_frame, text="Update WP Original Baseline",
command=self.open_update_cb_window, **self.fonts["button2"])
self.update_cb_button.grid(row=1, column=2, padx=10, pady=5, sticky="e")
self.update_cb_button.config(state="disabled")
return self.update_wp_info_frame
def open_update_cb_window(self):
    selected_wp_name = self.wp_name_combobox.get()
    if selected_wp_name == "Select Work Package Name" or (not selected_wp_name):
        messagebox.showerror("Error", "Please select both Project Name and Work Package Name.")
        return
    self.wp_name = selected_wp_name

# Create a new window
self.cb_window = tk.Toplevel(self.root)
self.cb_window.resizable(False, False)
self.cb_window.title("Update Original Baseline WP")
project_name_label = tk.Label(self.cb_window, text= f'Project Name: {self.project_name}', **self.fonts["label2"])
project_name_label.grid(row=0, column=0, padx=20, pady=4, sticky="w")
wp_name_label = tk.Label(self.cb_window, text= f'Work Package Name: {self.wp_name}', **self.fonts["label2"])
wp_name_label.grid(row=1, column=0, padx=20, pady=4, sticky="w")
project_id_label = tk.Label(self.cb_window, text= f'Project ID: {self.project_id}', **self.fonts["label2"])
project_id_label.grid(row=0, column=1, padx=20, pady=4, sticky="w")
wp_code_label = tk.Label(self.cb_window, text= f'Work Package Code: {self.wp_code}', **self.fonts["label2"])
wp_code_label.grid(row=1, column=1, padx=20, pady=4, sticky="w")

# Create a frame for Current Baseline
cb_frame = tk.LabelFrame(self.cb_window, text="WP Current Baseline", **self.fonts["labelframe2"])
cb_frame.grid(row=2, column=0, columnspan=2, padx=20, pady=2, sticky="ns")

```

```

# Entry labels and fields
self.baseline_widgets = {}
conn = sqlite3.connect(self.db)
wp_cb_data = pd.read_sql(f'SELECT * FROM "Work_package_current_baselines" WHERE
"work_package_code"="{self.wp_code}" ORDER BY "mod_date" DESC LIMIT 1', conn)
conn.close()
for i, label in enumerate(self.wp_baseline_labels):
    label_widget = tk.Label(cb_frame, text=f"{label}:", **self.fonts["label1"])
    label_widget.grid(row=i, column=0, padx=10, pady=5, sticky="w")
    value = wp_cb_data.loc[0, label.replace(' ', '_').lower()]
    if label.endswith('(mm/dd/yyyy)'):
        widget = DateEntry(cb_frame, **self.fonts["dateentry1"])
        widget.delete(0, tk.END)
        widget.insert(tk.END, value)
    else:
        widget = tk.Entry(cb_frame, **self.fonts["entry1"])
        widget.insert(tk.END, value)
    widget.grid(row=i, column=1, padx=10, pady=5, sticky="ew")
    self.baseline_widgets[label] = widget

# Save Button
save_button = tk.Button(self.cb_window, text="Save WP Current Baseline", command=self.save_baseline_data,
bg="#9370DB", fg="white", font=("Didot", 10, "bold"))
save_button.grid(row=3, column=0, columnspan=2, padx=10, pady=10)
def save_baseline_data(self):

# Extracting data from entries
bs_data = {label.replace(' ', '_').lower(): widget.get() for label, widget in self.baseline_widgets.items()}

# Check for incomplete fields
incomplete_fields = [label for label, value in bs_data.items() if not value.strip()]
if incomplete_fields:
    messagebox.showerror("Work Package Current Baseline Incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields))
    return
bs_data_df = pd.DataFrame(bs_data, index=[0], columns=bs_data.keys())
bs_data_df['work_package_code'] = self.wp_code
bs_data_df['mod_date'] = datetime.now().strftime("%Y/%m/%d %H:%M:%S")

# Update the project data dictionary
conn = sqlite3.connect(self.db)
kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
bs_data_df.fillna("").to_sql("Work_package_current_baselines", **kwargs)
messagebox.showinfo("Success", "Data saved successfully.")

# Close the window after saving
self.cb_window.destroy()
def update_wp_names(self, event=None):
    selected_project_name = self.project_name_combobox.get()
    if selected_project_name == "Select Project Name" or (not selected_project_name):
        messagebox.showwarning("Select Project", "Please select a project Name to view its information.")
        return
    self.project_name = selected_project_name
    conn = sqlite3.connect(self.db)

```

```

        self.project_id = pd.read_sql(f'SELECT "project_id" FROM "Projects" WHERE "project_name"="{self.project_name}" ORDER
BY "mod_date" DESC LIMIT 1', conn).iloc[0, 0]
        wps = pd.read_sql(f'SELECT "work_package_name" FROM "Work_packages" WHERE "project_id"="{self.project_id}"',
conn)['work_package_name'].to_list()
        conn.close()
        self.wp_name_combobox['values'] = wps
        self.wp_name_combobox.config(state="readonly")
        self.wp_name_combobox.set("Select Work Package Name")
def show_wp_details(self):
    selected_wp_name = self.wp_name_combobox.get()
    if selected_wp_name == "Select Work Package Name" or (not selected_wp_name):
        messagebox.showerror("Error", "Please select both Project Name and Work Package Name.")
        return
    self.wp_name = selected_wp_name

    # Retrieve work package data
    conn = sqlite3.connect(self.db)
    wp_data = pd.read_sql(f'SELECT * FROM "Work_packages" WHERE "work_package_name"="{self.wp_name}" ORDER BY
"mod_date" DESC LIMIT 1', conn)
    self.wp_code = wp_data.loc[0, 'work_package_code']
    wp_ob_data = pd.read_sql(f'SELECT * FROM "Work_package_original_baselines" WHERE
"work_package_code"="{self.wp_code}" ORDER BY "mod_date" DESC LIMIT 1', conn)
    conn.close()
    self.wp_info_frame = tk.LabelFrame(self.update_wp_info_frame, text="Work Package Information", fg= "teal" ,font=("Didot",
10, "bold"))
    self.wp_info_frame.grid(row=3, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")
    self.wp_info_widgets = {}

    # Create labels and entry widgets for each field
    for i, label in enumerate(self.wp_setup_labels):
        row = i // 2
        col = 0 if i % 2 == 0 else 2
        label_widget = tk.Label(self.wp_info_frame, text=f"{label}:", **self.fonts["label1"])
        label_widget.grid(row=row, column=col, padx=10, pady=5, sticky="ew")
        value = wp_data.loc[0, label.replace(' ', '_').lower()]
        if label in self.dict_combobox:
            widget = ttk.Combobox(self.wp_info_frame, values=self.dict_combobox[label], **self.fonts["combobox1"])
            widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
            widget.set(value)
            widget.config(state="disabled")
        elif label == "Description":
            widget = tk.Text(self.wp_info_frame, height=3)
            widget.grid(row=row + 1, column=0, columnspan=4, padx=10, pady=5, sticky="ew")
            widget.insert(tk.END, value)
            widget.config(state="disabled")
        else:
            widget = tk.Entry(self.wp_info_frame, **self.fonts["entry1"])
            widget.grid(row=row, column=col + 1, padx=10, pady=5, sticky="ew")
            widget.insert(tk.END, value)
            widget.config(state="readonly")
        self.wp_info_widgets[label] = widget

    # Add Original Baseline subframe
    ob_frame = tk.LabelFrame(self.wp_info_frame, text="Original Baseline", font=("Didot", 10, "bold"), foreground="teal")

```

```

ob_frame.grid(row=row+1, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")
self.wp_ob_widgets = {}
for i, label in enumerate(self.wp_baseline_labels):
    ob_row = i // 2
    ob_col = 0 if i % 2 == 0 else 2
    label_widget = tk.Label(ob_frame, text=f"{label}:", **self.fonts["label1"])
    label_widget.grid(row=ob_row, column=ob_col, padx=10, pady=5, sticky="w")
    value = wp_ob_data.loc[0, label.replace(' ', '_').lower()]
    if label in self.dict_combobox:
        widget = ttk.Combobox(ob_frame, values=self.dict_combobox[label], **self.fonts["combobox1"])
        widget.set(value)
        widget.config(state="disabled")
    elif label.endswith('(mm/dd/yyyy)'):
        widget = DateEntry(ob_frame, **self.fonts["dateentry1"])
        widget.delete(0, tk.END)
        widget.insert(tk.END, value)
        widget.config(state="disabled")

    else:
        widget = tk.Entry(ob_frame, **self.fonts["entry1"])
        widget.insert(tk.END, value)
        widget.config(state="readonly")
        self.wp_ob_widgets[label] = widget
        widget.grid(row=ob_row, column=ob_col + 1, padx=10, pady=2, sticky="ew")
self.edit_wp_button = tk.Button(self.update_wp_info_frame, text="Edit WP Information", command=self.edit_wp_information,
                                **self.fonts["button1"])
self.edit_wp_button.grid(row=4, column=0, padx=10, pady=10, sticky="e")
self.update_wp_button = tk.Button(self.update_wp_info_frame, text="Update WP Information",
command=self.update_wp_information,
                                **self.fonts["button2"])
self.update_wp_button.grid(row=4, column=1, padx=10, pady=10, sticky="e")

# Enable the "Update Original Baseline WP" button
self.update_cb_button.config(state="normal")
def edit_wp_information(self):
    [self.wp_info_widgets[label].config(state="normal") for label in self.wp_setup_labels if label not in ("Work Package Name",
"Work Package Code")]
    [self.wp_ob_widgets[label].config(state="normal") for label in self.wp_baseline_labels]
    self.update_wp_button.config(state="normal")
def update_wp_information(self):

# Create a dictionary to store the work package data
[self.wp_info_widgets[label].config(state="normal") for label in ("Work Package Name", "Work Package Code")]
wp_data = {}
for label in self.wp_setup_labels:
    if label == "Description":
        wp_data[label.replace(" ", "_").lower()] = self.wp_info_widgets[label].get("1.0", 'end-1c').strip()
    else:
        wp_data[label.replace(" ", "_").lower()] = self.wp_info_widgets[label].get().strip()
bs_data = {label.replace(" ", "_").lower(): self.wp_cb_widgets[label].get().strip() for label in self.wp_baseline_labels}

# Check for incomplete fields
incomplete_fields = [label for label, value in wp_data.items() if not value.strip()]
if incomplete_fields:

```

```

        messagebox.showerror("Project Incomplete Fields", "Please complete the following fields: " + " ".join(incomplete_fields))
    return
incomplete_fields = [label for label, value in bs_data.items() if not value.strip()]
if incomplete_fields:
    messagebox.showerror("Project Original Baseline Incomplete Fields", "Please complete the following fields: " + " ".join(incomplete_fields))
    return
conn = sqlite3.connect(self.db)
kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
mod_date = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
wp_data_df = pd.DataFrame(wp_data, index=[0], columns=wp_data.keys())
wp_data_df['mod_date'] = mod_date
bs_data_df = pd.DataFrame(bs_data, index=[0], columns=bs_data.keys())
bs_data_df['work_package_code'] = wp_data['work_package_code']
bs_data_df['mod_date'] = mod_date
wp_data_df.fillna("").to_sql("Work_packages", **kwargs)
bs_data_df.fillna("").to_sql("Work_package_original_baselines", **kwargs)
bs_data_df.fillna("").to_sql("Work_package_current_baselines", **kwargs)

conn.close()
messagebox.showinfo("Data Saved", "Data has been saved.")
self.update_wp_button.config(state="disabled")
[self.wp_info_widgets[label].config(state="disabled") if (label == "Description" or label in self.dict_combobox) else
self.wp_info_widgets[label].config(state="readonly") for label in self.wp_setup_labels]
[self.wp_cb_widgets[label].config(state="disabled") if (label.endswith('(mm/dd/yyyy)') or label in self.dict_combobox) else
self.wp_cb_widgets[label].config(state="readonly") for label in self.wp_baseline_labels]

```

2.3 WEEKLY PROGRESS

```

def create_project_cost_status_frame(self):
    self.project_cost_status_frame = tk.LabelFrame(self.root, text="WEEKLY PROGRESS PER WORK PACKAGE",
**self.fonts["labelframe1"])
    self.project_cost_status_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")

    # Project Name ComboBox
    project_name_label = tk.Label(self.project_cost_status_frame, text="Project Name:", **self.fonts["label1"])
    project_name_label.grid(row=0, column=0, padx=10, pady=5, sticky="w")

    # Populate ComboBox with project names
    conn = sqlite3.connect(self.db)
    project_names = pd.read_sql('SELECT DISTINCT "project_name" FROM "Projects"', conn)['project_name'].to_list()
    conn.close()
    self.project_name_combobox = ttk.Combobox(self.project_cost_status_frame, values=project_names, state="readonly",
**self.fonts["combobox1"])
    self.project_name_combobox.set('Select Project Name')
    self.project_name_combobox.bind("<<ComboboxSelected>>", self.update_wp_combobox)
    self.project_name_combobox.grid(row=0, column=1, padx=10, pady=5, sticky="ew")

    # Work Package Name ComboBox
    wp_name_label = tk.Label(self.project_cost_status_frame, text="Work Package Name:", **self.fonts["label1"])
    wp_name_label.grid(row=0, column=2, padx=10, pady=5, sticky="w")
    self.wp_name_combobox = ttk.Combobox(self.project_cost_status_frame, **self.fonts["combobox1"])
    self.wp_name_combobox.grid(row=0, column=3, padx=10, pady=5, sticky="ew")
    self.wp_name_combobox.config(state="disabled")

```

```

# Other labels and entries
labels = ["Period Number", "Week Ending (mm/dd/yyyy)", "Status at Completion"]
self.progress_widgets = {} # Dictionary to store entry widgets
for i, label in enumerate(labels):
    row = i // 2
    col = 0 if i % 2 == 0 else 2
    label_widget = tk.Label(self.project_cost_status_frame, text=f"{label}.", **self.fonts["label1"])
    label_widget.grid(row=row + 1, column=col, padx=10, pady=5, sticky="w")
    if label in self.dict_combobox:
        widget = ttk.Combobox(self.project_cost_status_frame, values=self.dict_combobox[label], state="readonly",
**self.fonts["combobox1"])
        widget.set(self.dict_combobox[label][0])
    elif label.endswith("(mm/dd/yyyy)":
        widget = DateEntry(self.project_cost_status_frame, **self.fonts["dateentry1"])
    else:
        widget = tk.Entry(self.project_cost_status_frame, **self.fonts["entry1"])
    widget.grid(row=row + 1, column=col + 1, padx=10, pady=5, sticky="ew")
    self.progress_widgets[label] = widget

self.show_information_button = tk.Button(self.project_cost_status_frame, text="Show Information",
command=self.show_information, bg="cadetblue1", fg="gray25", font=("Didot", 9, 'bold'))
self.show_information_button.grid(row=3, columnspan=4, padx=10, pady=5)
return self.project_cost_status_frame
def show_information(self):
    selected_project_name = self.project_name_combobox.get()
    if selected_project_name == "Select Project Name" or (not selected_project_name):
        messagebox.showwarning("Select Project", "Please select a project Name to view its information.")
        return
    selected_wp_name = self.wp_name_combobox.get()
    if selected_wp_name == "Select Work Package Name" or (not selected_wp_name):
        messagebox.showwarning("Select Work Package", "Please select a Work Package Name to view its information.")
        return
    selected_week_ending = self.progress_widgets["Week Ending (mm/dd/yyyy)"].get()
    if not selected_week_ending:
        messagebox.showwarning("Select Week Ending", "Please insert a Week Ending to view cumulate information.")
        return
    else:
        try:
            week_ending = pd.to_datetime(selected_week_ending, format="%m/%d/%Y").strftime(format="%Y/%m/%d")
        except:
            messagebox.showwarning("Wrong Week Ending", "Please insert a week ending with 'mm/dd/yyyy' format to view
cumulate information.")
            return
    self.wp_name = selected_wp_name
    conn = sqlite3.connect(self.db)
    wp_data = pd.read_sql(f'SELECT * FROM "Work_packages" WHERE "work_package_name"="{self.wp_name}" ORDER BY
"mod_date" DESC LIMIT 1', conn)
    self.wp_code = wp_data.loc[0, 'work_package_code']
    wp_ob_data = pd.read_sql(f'SELECT * FROM "Work_package_original_baselines" WHERE
"work_package_code"="{self.wp_code}" ORDER BY "mod_date" DESC LIMIT 1', conn).fillna("")
    wp_cb_data = pd.read_sql(f'SELECT * FROM "Work_package_current_baselines" WHERE
"work_package_code"="{self.wp_code}" ORDER BY "mod_date" DESC LIMIT 1', conn).fillna("")

```



```

self.last_weekly_progress_data = pd.read_sql(f'SELECT * FROM "Weekly_progress" WHERE
"work_package_code"="{self.wp_code}" AND "week_ending"<="{week_ending}" ORDER BY "week_ending" DESC LIMIT 1',
conn).fillna("")
past_week_ending = pd.to_datetime(self.last_weekly_progress_data.loc[0, 'week_ending'],
format="%Y/%m/%d").strftime("%m/%d/%Y")
self.baseline_information_frame = tk.LabelFrame(self.project_cost_status_frame, text="Baselines Information",
**self.fonts["labelframe2"])
self.baseline_information_frame.grid(row=4, columnspan=4, padx=10, pady=5)

# Creating the table baselines information
self.baseline_table_entries = {}
for t1_col, (bs, bs_data) in enumerate(zip(["Original Baseline", "Current Baseline"], [wp_ob_data, wp_cb_data])):
    self.baseline_table_entries[bs] = {}
    col_label = tk.Label(self.baseline_information_frame, text=bs, font=("Didot", 9, "bold"))
    col_label.grid(row=0, column=t1_col + 1, padx=10, pady=5)
    for t1_row, label in enumerate(["Budget at Completion ($)", "Duration (days)", "Quantity"]):
        if t1_col == 0:
            row_label = tk.Label(self.baseline_information_frame, text=f"{label}:", **self.fonts["label1"])
            row_label.grid(row=t1_row + 1, column=0, padx=10, pady=5, sticky="w")
            entry = tk.Entry(self.baseline_information_frame, **self.fonts["entry1"])
            entry.grid(row=t1_row + 1, column=t1_col + 1, padx=10, pady=5, sticky="ew")
            entry.insert(0, bs_data.loc[0, label.replace(' ', '_').lower()])
            entry.config(state="readonly")
            self.baseline_table_entries[bs][label] = entry
self.weekly_progress_frame = tk.LabelFrame(self.project_cost_status_frame, text="Weekly Progress Information",
**self.fonts["labelframe2"])
self.weekly_progress_frame.grid(row=5, columnspan=4, padx=10, pady=5)
self.weekly_progress_table_widgets = {}
row_labels = ["Earned Value Cumulate ($)", "Planned Value Cumulate ($)", "Actual Cost Cumulate ($)", "Quantity Cumulate"]
col_labels = ["Previous Cumulative", "Current Week", "Current Cumulative"]
for t1_row, label in enumerate(row_labels):
    self.weekly_progress_table_widgets[label] = {}
    if label == "Actual Cost Cumulate ($)":
        row_label = tk.Label(self.weekly_progress_frame, text=f"{label.replace(' Cumulate', '')}:", fg="blue", font=("Didot", 9,
"underline"), cursor="hand2")
        row_label.bind("<Button-1>", lambda e: self.open_actual_cost_window())
    else:
        row_label = tk.Label(self.weekly_progress_frame, text=f"{label.replace(' Cumulate', '')}:", **self.fonts["label1"])
    row_label.grid(row=t1_row + 1, column=0, padx=10, pady=5, sticky="w")
    col_widget = tk.Label(self.weekly_progress_frame, text=f"{col_labels[0]}\n{past_week_ending}", font=("Didot", 9, "bold"))
    col_widget.grid(row=0, column=1, padx=10, pady=10)
    col_widget = tk.Label(self.weekly_progress_frame, text=col_labels[1], font=("Didot", 9, "bold"))
    col_widget.grid(row=0, column=2, padx=10, pady=10)
    for t1_col, bs in enumerate(col_labels):
        entry = tk.Entry(self.weekly_progress_frame, **self.fonts["entry1"])
        entry.grid(row=t1_row + 1, column=t1_col + 1, padx=10, pady=5, sticky="ew")
        self.weekly_progress_table_widgets[label][bs] = entry
    if t1_row == 0:
        if t1_col == 0:
            col_widget = tk.Label(self.weekly_progress_frame, text=f"{bs}\n{past_week_ending}", font=("Didot", 9, "bold"))
        elif t1_col == 1:
            col_widget = tk.Label(self.weekly_progress_frame, text=bs, font=("Didot", 9, "bold"))
            col_widget.grid(row=0, column=t1_col + 1, padx=10, pady=10)
        if t1_col != 1:

```

```

        entry.insert(0, self.last_weekly_progress_data.loc[0, label.replace(' ', '_').lower()])
        entry.config(state="disabled")
def update_all_sum_entries():
    for label in row_labels:
        entry1 = self.weekly_progress_table_widgets[label][col_labels[0]]
        entry2 = self.weekly_progress_table_widgets[label][col_labels[1]]
        entrysum = self.weekly_progress_table_widgets[label][col_labels[2]]
        update_sum_entries(entrysum, entry1, entry2)
col_widget_btn_sum = tk.Button(self.weekly_progress_frame, text=bs, command=update_all_sum_entries,
                                font=("Didot", 9, "bold"))
col_widget_btn_sum.grid(row=0, column=3, padx=10, pady=10)

# Adding Comments Label and Entry
comments_label = tk.Label(self.weekly_progress_frame, text="Comments:", **self.fonts["label1"])
comments_label.grid(row=5, column=0, padx=10, pady=5, sticky="w")
self.comments_text = tk.Text(self.weekly_progress_frame, width=50, height=2, **self.fonts["entry1"]) # Adjust width and
height as needed
self.comments_text.configure(bg="azure")
self.comments_text.grid(row=5, column=1, columnspan=4, padx=10, pady=5, sticky="ew")

self.progress_widgets["Comments"] = self.comments_text

# Adding Save Button
save_button = tk.Button(self.project_cost_status_frame, text="Save Progress", command=self.save_weekly_progress_data,
bg="#9370DB", fg="white", font=("Didot", 10, "bold"))
save_button.grid(row=6, column=3, padx=10, pady=10, sticky="e")
note_label_text = "NOTE: Project Manager or Project Controller only, have permissions to Submit this Form."
note_label = tk.Label(self.project_cost_status_frame, text=note_label_text, font=("Didot", 8, "italic"))
note_label.grid(row=7, column=0, columnspan=4, padx=10, pady=5, sticky="w")
self.actual_cost_details= {label: '' for label in self.actual_cost_details_labels}
def open_actual_cost_window(self):
    selected_project_name = self.project_name_combobox.get()
    if selected_project_name == "Select Project Name" or (not selected_project_name):
        messagebox.showwarning("Select Project", "Please select a project Name to view its information.")
        return
    selected_wp_name = self.wp_name_combobox.get()
    if selected_wp_name == "Select Work Package Name" or (not selected_wp_name):
        messagebox.showwarning("Select Project", "Please select a Work Package Name to view its information.")
        return
    self.wp_name = selected_wp_name
    conn = sqlite3.connect(self.db)
    wp_data = pd.read_sql(f'SELECT * FROM "Work_packages" WHERE "work_package_name"="{self.wp_name}" ORDER BY
"mod_date" DESC LIMIT 1', conn)
    self.wp_code = wp_data.loc[0, 'work_package_code']
    wp_ob_data = pd.read_sql(f'SELECT * FROM "Work_package_original_baselines" WHERE
"work_package_code"="{self.wp_code}" ORDER BY "mod_date" DESC LIMIT 1', conn)
    wp_cb_data = pd.read_sql(f'SELECT * FROM "Work_package_current_baselines" WHERE
"work_package_code"="{self.wp_code}" ORDER BY "mod_date" DESC LIMIT 1', conn)
    conn.close()

# Create a top-level window
actual_cost_window = tk.Toplevel(self.root)
actual_cost_window.resizable(False, False)
actual_cost_window.title("Actual Cost Details")

```

```

project_name_label = tk.Label(actual_cost_window, text=f"Project Name: {self.project_name}", **self.fonts["label2"])
project_name_label.grid(row=0, padx=10, pady=2, sticky="w")
wp_name_label = tk.Label(actual_cost_window, text=f"Work Package Name: {self.wp_name}", **self.fonts["label2"])
wp_name_label.grid(row=1, padx=10, pady=2, sticky="w")

# Define table columns and rows
columns = ["Original Baseline", "Current Baseline", "Actual Cumulative"]

# Create a frame for the table
table_frame = tk.LabelFrame(actual_cost_window, text='Actual Cost Details', **self.fonts["labelframe2"])
table_frame.grid(row=2, padx=10, pady=4, sticky='nsew')

# Creating the table
self.actual_cost_details_widgets = {}
for row_index, row in enumerate(self.actual_cost_details_labels):

    # Create a label for the row
    row_label = tk.Label(table_frame, text=row, **self.fonts["label1"])
    row_label.grid(row=row_index + 1, column=0, padx=10, pady=5, sticky="w")

    # Create entry widgets for each column in the row
    for col_index, (column, column_data) in enumerate(zip(columns, (wp_ob_data, wp_cb_data, self.actual_cost_details))):
        widget = tk.Entry(table_frame, **self.fonts["entry1"])
        widget.grid(row=row_index + 1, column=col_index + 1, padx=10, pady=5, sticky="ew")
        if col_index == 2:
            widget.insert(tk.END, column_data[row])
            self.actual_cost_details_widgets[row] = widget
        else:
            widget.insert(tk.END, column_data.loc[0, row.replace(' ', '_').lower()])
            widget.config(state="readonly")

    # Creating column headers
    for col_index, col in enumerate(columns):
        col_label = tk.Label(table_frame, text=col, font=("Didot", 9, "bold"))
        col_label.grid(row=0, column=col_index + 1, padx=10, pady=5, sticky="ew")

# Add a save button
save_button = tk.Button(actual_cost_window, text="Save Actual Cost", command=self.save_actual_cost_details,
bg="Mediumpurple3", fg="white", font=("Didot", 9, "bold"))
save_button.grid(row=3, columnspan=2, padx=10, pady=10)
def save_actual_cost_details(self):

    # Check for incomplete fields
    actual_cost_details = {label: self.actual_cost_details_widgets[label].get().strip() for label in self.actual_cost_details_labels}
    incomplete_fields = [label for label, value in actual_cost_details.items() if not value.strip()]
    if incomplete_fields:
        messagebox.showerror("Actual Cost Details Incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields))
        return
    self.actual_cost_details = actual_cost_details
    messagebox.showinfo("Data Saved", "Data has been saved in temporary memory.")

# Callback function to update Work Package combobox
def update_wp_combobox(self, event=None):

```

```

selected_project_name = self.project_name_combobox.get()
if selected_project_name == "Select Project Name" or (not selected_project_name):
    messagebox.showwarning("Select Project", "Please select a project Name to view its information.")
    return
self.project_name = selected_project_name
conn = sqlite3.connect(self.db)
self.project_id = pd.read_sql(f'SELECT "project_id" FROM "Projects" WHERE "project_name"="{selected_project_name}"
ORDER BY "mod_date" DESC LIMIT 1', conn).iloc[0, 0]
wp_names = pd.read_sql(f'SELECT DISTINCT "work_package_name" FROM "Work_packages" WHERE
"project_id"="{self.project_id}"', conn)["work_package_name"].to_list()
conn.close()
self.wp_name_combobox['values'] = wp_names
self.wp_name_combobox.config(state="readonly")
self.wp_name_combobox.set("Select Work Package Name")
def update_week_ending_calendar(self, event=None):
    self.wp_name = self.wp_name_combobox.get()
    conn = sqlite3.connect(self.db)
    wp_data = pd.read_sql(f'SELECT * FROM "Work_packages" WHERE "work_package_name"="{self.wp_name}" ORDER BY
"mod_date" DESC LIMIT 1', conn)
    self.wp_code = wp_data.loc[0, 'work_package_code']
    self.data_bd_0 = pd.read_sql(f'SELECT * FROM "Weekly_progress" WHERE "work_package_code"="{self.wp_code}" ORDER
BY "week_ending" ASC', conn)
    conn.close()

    self.week_ending_calendar.config(state="readonly")
    last_week_ending = pd.to_datetime(self.data_bd_0.iloc[-1, 0], format="%Y/%m/%d")
    self.week_ending_calendar.set_date(last_week_ending)
def save_weekly_progress_data(self):
    row_labels = ["Earned Value Cumulate ($) ", "Planned Value Cumulate ($) ", "Actual Cost Cumulate ($) ", "Quantity Cumulate"]
    col_labels = ["Previous Cumulative", "Current Week", "Current Cumulative"]
    progress_data = {}
    for row in row_labels:
        for col in col_labels:
            if col == "Current Week":
                progress_data[row.replace(" Cumulate", "").replace(" ", "_").lower()] =
self.weekly_progress_table_widgets[row][col].get()
            elif col == "Current Cumulative":
                progress_data[row.replace(" ", "_").lower()] = self.weekly_progress_table_widgets[row][col].get().strip()
    for label in self.progress_widgets:
        if label == "Week Ending (mm/dd/yyyy)":
            progress_data["week_ending"] = pd.to_datetime(self.progress_widgets[label].get(),
format="%m/%d/%Y").strftime("%Y/%m/%d")
        elif label == "Comments":
            progress_data["comments"] = self.progress_widgets["Comments"].get("1.0", 'end-1c').strip()
        else:
            progress_data[label.replace(" ", "_").lower()] = self.progress_widgets[label].get().strip()

    # Check for incomplete fields
    incomplete_fields = [label for label, value in progress_data.items() if not value.strip()]
    if incomplete_fields:
        messagebox.showerror("Weekly progress incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields))
    return

```

```

# Check for incomplete fields
incomplete_fields = [label for label, value in self.actual_cost_details.items() if not value.strip()]
if incomplete_fields:
    messagebox.showerror("Actual Cost incomplete Fields", "Please complete the following fields: " + ",
".join(incomplete_fields) + ". Click Actual Cost Details to complete them.")
    return
for label in self.actual_cost_details:
    label_format = label.replace(" ($)", " Cumulate ($)").replace(" ", "_").lower()
    progress_data[label_format] = round(float(self.last_weekly_progress_data.loc[0, label_format]) +
float(self.actual_cost_details[label]), 3)

progress_data["work_package_code"] = self.wp_code
conn = sqlite3.connect(self.db)
kwargs = {'con': conn, 'if_exists': 'append', 'index': False}
mod_date = datetime.now().strftime('%Y/%m/%d %H:%M:%S')
progress_data['mod_date'] = mod_date
progress_data_df = pd.DataFrame(progress_data, index=[0], columns=progress_data.keys())
progress_data_df.fillna("").to_sql("Weekly_progress", **kwargs)
conn.close()
messagebox.showinfo("Data Saved", "Data has been saved.")

```

3.0 COST FORECASTING

3.1 COST FORECASTING FRAME

```

def create_project_forecasting_frame(self):
    self.forecasting_frame = tk.LabelFrame(self.root, text="PROJECT FORECASTING", **self.fonts["labelframe1"])
    self.forecasting_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")

# Populate ComboBox with project names
conn = sqlite3.connect(self.db)
project_names = pd.read_sql('SELECT DISTINCT "project_name" FROM "Projects"', conn)['project_name'].to_list()
conn.close()
project_name_label = tk.Label(self.forecasting_frame, text="Project Name:", **self.fonts["label1"])
project_name_label.grid(row=0, column=0, padx=10, pady=3, sticky="nsw")
self.project_name_combobox = ttk.Combobox(self.forecasting_frame, state="readonly", **self.fonts["combobox1"])
self.project_name_combobox['values'] = project_names
self.project_name_combobox.set("Select Project Name")
self.project_name_combobox.grid(row=0, column=1, padx=10, pady=3, sticky="nsw")
self.project_name_combobox.bind("<<ComboboxSelected>>", self.update_wp_combobox)
wp_name_label = tk.Label(self.forecasting_frame, text="Work Package Name:", **self.fonts["label1"])
wp_name_label.grid(row=0, column=2, padx=10, pady=3, sticky="nsw")
self.wp_name_combobox = ttk.Combobox(self.forecasting_frame, state="readonly", **self.fonts["combobox1"])
self.wp_name_combobox.grid(row=0, column=3, padx=10, pady=3, sticky="nsw")
self.wp_name_combobox.config(state="disabled")
self.wp_name_combobox.bind("<<ComboboxSelected>>", self.update_week_ending_calendar)

# Week Ending ComboBox
week_ending_label = tk.Label(self.forecasting_frame, text="Week Ending (mm/dd/yyyy):", **self.fonts["label1"])
week_ending_label.grid(row=1, column=0, padx=10, pady=3, sticky="w")
self.week_ending_calendar = DateEntry(self.forecasting_frame, **self.fonts["dateentry1"])
self.week_ending_calendar.grid(row=1, column=1, padx=10, pady=3, sticky="nesw")
self.week_ending_calendar.config(state="disabled")
self.predict_button = tk.Button(self.forecasting_frame, text="Deep Learning Model", width=25, command=self.predict,
**self.fonts["button2"])

```

```

self.predict_button.grid(row=2, column=0, columnspan=4, padx=10, pady=3)

# Label to display prediction
'''self.predi_label = tk.Label(self.forecasting_frame, text="Deep learning model to forecast the EAC at the work package
level.", **self.fonts["label1"])
self.predi_label.grid(row=2, column=0, columnspan=2, padx=10, pady=3, sticky="nw")'''
return self.forecasting_frame
def predict(self):
    self.predict_eac()
    self.display_prediction()
def predict_eac(self, ):
    selected_project_name = self.project_name_combobox.get()
    if selected_project_name == "Select Project Name" or (not selected_project_name):
        messagebox.showwarning("Select Project", "Please select a project Name to view its information.")
        return
    selected_wp_name = self.wp_name_combobox.get()
    if selected_wp_name == "Select Work Package Name" or (not selected_wp_name):
        messagebox.showwarning("Select Work Package", "Please select a Work Package Name to view its information.")
        return
    selected_week_ending = self.week_ending_calendar.get()
    if not selected_week_ending:
        messagebox.showwarning("Select Week Ending", "Please insert a Week Ending to view cumulate information.")
        return
    else:
        try:
            week_ending = pd.to_datetime(selected_week_ending, format="%m/%d/%Y").strftime(format="%Y/%m/%d")
        except:
            messagebox.showwarning("Wrong Week Ending", "Please insert a week ending with 'mm/dd/yyyy' format to view
cumulate information.")
            return
    self.project_name = selected_project_name
    self.wp_name = selected_wp_name
    conn = sqlite3.connect(self.db)
    wp_data = pd.read_sql(f'SELECT * FROM "Work_packages" WHERE "work_package_name"="{self.wp_name}" ORDER BY
"mod_date" DESC LIMIT 1', conn)
    self.wp_code = wp_data.loc[0, 'work_package_code']
    data_cb = pd.read_sql(f'SELECT * FROM "Work_package_current_baselines" WHERE "work_package_code"="{self.wp_code}"
ORDER BY "mod_date" DESC LIMIT 1', conn)
    data_cb.to_csv(f'Data_{self.wp_name}_cb.csv', index=False)
    data_ob = pd.read_sql(f'SELECT * FROM "Work_package_original_baselines" WHERE "work_package_code"="{self.wp_code}"
ORDER BY "mod_date" DESC LIMIT 1', conn)
    data0 = pd.read_sql(f'SELECT * FROM "Weekly_progress" WHERE "work_package_code"="{self.wp_code}" AND
"week_ending" <= "{week_ending}" ORDER BY "week_ending" ASC', conn)
    self.data_bd = data0
    self.data_bd.to_csv(f'Data_{self.wp_name}_bd.csv', index=False)
    EV_cumulate = data0[data0['week_ending'] == week_ending]['earned_value_cumulate($)'].iloc[0]
    replaced_value = (data_cb.loc[0, 'budget_at_completion($)'] - EV_cumulate)
    cb_periods = round((data_cb.loc[0, 'duration_(days)'] / 7), 0).astype(np.int64)
    update_statement1 = """
UPDATE Weekly_progress
SET Estimate_to_completion = ?
WHERE week_ending = ? AND work_package_code = ?;
"""
    conn.execute(update_statement1, (replaced_value, week_ending, self.wp_code))

```

```

conn.commit()
data = pd.read_sql(f'SELECT * FROM "Weekly_progress" WHERE "work_package_code"="{self.wp_code}" AND
"week_ending" <= "{week_ending}" ORDER BY "week_ending" ASC', conn)
data['ETC'] = data['Estimate_to_completion']
data['EV'] = data['earned_value_cumulate_($)']
data['%DURATION'] = data['period_number'] * 7 / data.cb.loc[0, 'duration_(days)']
data['AC'] = data['actual_cost_cumulate_($)']
data['Labour cost'] = data['labour_cost_cumulate_($)']
data['Executed quantity'] = data['quantity_cumulate']
print("data\n", data)
"el resultado se le suma el actual cost cumulate (varía de semana en semana)"
self.dataset = data[['ETC', 'EV', '%DURATION', 'AC', 'Labour cost', 'Executed quantity']]
    if self.wp_name == "Concrete":
        self.model = load_model("model2GRU.h5")
    elif self.wp_name == "Piping (HDPE)":
        self.model = load_model("PIPING_GRU.h5")
    elif self.wp_name == "Barrow Fill":
        self.model = load_model("Fill_GRU.h5")
    else:
        self.model = load_model("model2lstm.h5")
self.predicts = []
for i in range(self.dataset.shape[0] - 3):
    dataset_red = self.dataset.iloc[i + 4].copy()
    print(dataset_red)
    a, b = self.prepare_data_for_prediction(dataset_red, 3, 6)
    etc_predict = self.make_prediction(a,b)[-1]
    self.predicts.append(etc_predict)
predic_value = self.predicts[-1]
predic_value = predic_value.astype(float)

#Updating predict value on the database
update_statement = """
UPDATE Weekly_progress
SET Estimate_to_completion = ?
WHERE week_ending = ? AND work_package_code = ?;
"""

conn.execute(update_statement, (predic_value, week_ending, self.wp_code))
conn.commit()
conn.close()
self.eac = self.predicts[-1] + self.data_bd.iloc[-1, 7]
print("type:", type(self.predicts[-1]))
self.data_bd.iloc[-1,-1] = float(self.predicts[-1])
self.data_bd['Estimate_to_completion']=self.data_bd['Estimate_to_completion'].astype('float64')

# Cost Status Graph
data_graphic = self.data_bd[['period_number', 'planned_value_cumulate_($)', 'earned_value_cumulate_($)',
'actual_cost_cumulate_($)']]
index_to_mark = len(data_graphic)
data_graphic['EAC'] = np.nan
data_graphic.loc[len(data_graphic)] = [cb_periods, np.nan, np.nan, np.nan, self.eac]
all_periods = pd.DataFrame({'period_number': range(1, cb_periods+1)})
all_periods['period_number'] = all_periods['period_number'].astype('int64')
data_graphic = pd.merge(all_periods, data_graphic, on='period_number', how='left')
"data_graphic.loc[:, 'period_number'] = data_graphic['period_number'].apply(lambda x: f'{str(x).zfill(2)}')

```

```

data_graphic.set_index('period_number', inplace=True)'''
self.fig, ax = self.graphic_budget_status(data_graphic, index_to_mark)

# Cost Status Table
data_summary_1 = self.data_bd[['period_number', 'week_ending', 'earned_value_cumulate_($)', 'actual_cost_cumulate_($)',
'Estimate_to_completion']]
data_summary_1['Original Baseline'] = data_ob.loc[0, 'budget_at_completion_($)']
data_summary_1['Current Baseline'] = data_cb.loc[0, 'budget_at_completion_($)']
data_summary_1['Estimate at Completion'] = data_summary_1['actual_cost_cumulate_($)'] +
data_summary_1['Estimate_to_completion']
print('Data Summary type: \n', data_summary_1.dtypes)
data_summary_1 = data_summary_1.drop('Estimate_to_completion', axis=1)
data_summary_1['Variance (%)'] = (((data_summary_1['Current Baseline'] - data_summary_1['Estimate at Completion']) /
(data_summary_1['Current Baseline'])*100).round(2)
columns_to_format = ['earned_value_cumulate_($)', 'actual_cost_cumulate_($)', 'Original Baseline', 'Current
Baseline', 'Estimate at Completion']
for column in columns_to_format:
    data_summary_1[column] = data_summary_1[column].map('{:.2f}'.format)
data_summary_1 = data_summary_1[['period_number', 'week_ending', 'Original Baseline', 'Current
Baseline', 'earned_value_cumulate_($)', 'actual_cost_cumulate_($)', 'Estimate at Completion', 'Variance (%)']]
data_summary_1.rename(columns={'period_number': 'Period Number', 'week_ending': 'Week Ending', 'Original Baseline':
'Original BAC ($)',
'Current Baseline': 'Current BAC ($)', 'earned_value_cumulate_($)': 'Earned Value Cumulate ($)',
'actual_cost_cumulate_($)': 'Actual Cost Cumulate ($)',
'Estimate at Completion': 'EAC Predicted ($)', 'Variance (%)': 'Variance (%)'}, inplace=True)
print("data_summary_1\n", data_summary_1)
tree = ttk.Treeview(self.forecasting_frame, columns=list(data_summary_1.columns), show='headings', height=6)
for column in data_summary_1.columns:
    tree.heading(column, text=column)
    tree.column(column, width=tk.font.Font().measure(column.title()))
scrollbar = ttk.Scrollbar(self.forecasting_frame, orient='vertical', command=tree.yview)
tree.configure(yscrollcommand=scrollbar.set)
style = ttk.Style()
style.configure("LastRow.TLabel", background="yellow")

# Inserting all rows and tagging the last row
total_rows = len(data_summary_1.index)
for i, row in data_summary_1.iterrows():
    if i == total_rows - 1:
        tree.insert("", 'end', values=list(row), tags=('lastrow',))
    else:
        tree.insert("", 'end', values=list(row))
tree.tag_configure('lastrow', background='violet')
tree.grid(row=7, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")
scrollbar.grid(row=7, column=5, sticky='ns')
self.note_label = tk.Label(self.forecasting_frame, text="Note: Variance (%) calculated as [Current BAC - EAC predicted] /
[Current BAC]",)
self.note_label.grid(row=8, column=0, columnspan=4, padx=10, pady=3, sticky="nw")
def series_to_supervised(self, data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()

# input sequence (t-n, ... t-1)

```



```

for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]

# forecast sequence (t, t+1, ... t+n)
for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]

# put it all together
agg = pd.concat(cols, axis=1)
agg.columns = names

# drop rows with NaN values
if dropnan:
    agg.dropna(inplace=True)
return agg

def prepare_data_for_prediction(self, data, n_past, n_features):

    # Ensure all data is float
    values = data.astype('float32')

    # Normalize features using MinMaxScaler
    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(values)
    reframed = self.series_to_supervised(scaled, n_past, 1)

    # Reshape input to be 3D [samples, timesteps, features]
    n_obs = n_past * n_features
    X = reframed.iloc[:, :n_obs]
    X = X.values.reshape((-1, n_past, n_features))
    return X, scaler

def make_prediction(self, prepared_data, scaler, actual_data=None):
    try:
        prediction = self.model.predict(prepared_data)
        # Adjusting the inversion of scaling
        inv_pred = np.concatenate((prediction, prepared_data[:, -1, -5:]), axis=1)
        inv_pred = scaler.inverse_transform(inv_pred)
        inv_pred = inv_pred[:, 0]
        return inv_pred
    except Exception as e:
        messagebox.showerror("Error", f"Error in making prediction: {e}")
        return None

def collect_prediction_data(self):

    # Specify the path to your CSV file
    csv_file_path = self.dataset.iloc[4]
    try:

        # Read the CSV file into a DataFrame
        data = pd.read_csv(csv_file_path)

```

```

        return data
    except Exception as e:
        messagebox.showerror("Error", f"Failed to load data: {e}")
        return None
def display_prediction(self):
    self.canvas = FigureCanvasTkAgg(self.fig, master=self.forecasting_frame)
    self.canvas.draw()
    self.canvas.get_tk_widget().grid(row=6, columnspan=4, padx=10, pady=3, sticky="nsew")
def graphic_budget_status(self, data_graphic, index_to_mark):
    plt.style.use('default')
    font_properties = {'family': 'Arial', 'size': 12}
    fig, ax = plt.subplots(figsize=(11, 4))
    ax.set_facecolor('lemonchiffon')
    print(data_graphic)
    ax.plot(data_graphic['period_number'], data_graphic['planned_value_cumulate_($)'], label='Planned', color='limegreen')
    ax.plot(data_graphic['period_number'], data_graphic['earned_value_cumulate_($)'], label='Earned', color='dodgerblue')
    ax.plot(data_graphic['period_number'], data_graphic['actual_cost_cumulate_($)'], label='Actual', color='deeppink')
    ax.plot(data_graphic['period_number'], data_graphic['EAC'], label='EAC predicted', marker='o', markersize=10,
markerfacecolor='violet', markeredgecolor='violet', markeredgewidth=2)
    ax.axvline(x=index_to_mark, color='r', linestyle='--', label='Status Date')
    ax.spines['bottom'].set_color('black')
    ax.spines['left'].set_color('black')
    x = data_graphic['period_number'].iloc[-1] # Last index of the DataFrame
    y = data_graphic['EAC'].iloc[-1] # Last value of the EAC column
    ax.annotate(f'EAC: {y:,.2f}', xy=(x, y), xytext=(x, 0.9*y),
        arrowprops=dict(facecolor='violet', arrowstyle="->"),
        fontsize=12, fontfamily='Arial', ha='right')
    ax.set_xlabel('Period Number', fontdict=font_properties)
    ax.set_ylabel('Cost ($)', fontdict=font_properties)
    ax.set_title(f'Cost Status\n{self.project_name} - {self.wp_name}', fontdict={'family': 'Arial', 'size': 14, 'color': 'black'})
    ax.set_xticks(data_graphic['period_number'])
    ax.tick_params(axis='x', labelsz=8, colors='black')
    ax.tick_params(axis='y', labelsz=12, colors='black')
    for label in ax.get_xticklabels() + ax.get_yticklabels():
        label.set_fontname('Arial')
    ax.grid(True, which='both', linestyle='-', linewidth=0.5, color='gainsboro')
    ax.legend(prop={'family': 'Arial', 'size': 10}, facecolor='white', edgecolor='black', fancybox=False)
    return fig, ax
def main():
    root = tk.Tk()
    app = ProjectCostManagementApp(root)
    root.mainloop()
if __name__ == "__main__":
    main()

```