SoDa-TAP v2: Social Data Analysis Made Simple

by

Hassnain Ali

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Social platforms are the mirror of our society's values, beliefs, and activities and have become the subject of study of many disciplines; researchers often study the themes and sentiments of social-platform discussions and attempt to understand how the various aspects of these discussions correlate with people's influence and the spread of ideas. This type of research requires substantial software engineering work. We developed SoDa-TAP$_{v2}$ (Social Data - Toolkit Analysis Platform, Version 2) to automate many useful tools and to make them available to scholars of all disciplines. SoDa-TAP$_{v2}$ integrates (i) a data-ingestion and analysis pipeline, and (ii) a visual query language through which to review data and evaluate hypotheses. The pipeline enhances datasets with lexical analyses, sentiment analysis, humor detection, and identification of personal values and Big Five personality traits from text and images. The visual query language features an intuitive drag-and-drop interface that enables users to filter and slice datasets to create distinct sample sets and save them for future use, perform aggregations, categorize data into buckets through classification, clustering, and natural breaks, and compare these buckets using statistical analyses and visualizations.

Acknowledgements

First of all, a huge thanks to Dr. Eleni Stroulia for her unwavering support and love during the entire process. I am forever grateful to you for all the guidance and growth.

I would also like to thank Dr. Lianne Lefsrud for her support during the research, and to Dr. Kenny Wong, and Dr. Paul Lu, for serving on my thesis committee.

A special thanks to everyone from the SSRG lab for being an awesome team.

I dedicate this thesis in memory of Abu who is always in our hearts, to Ami for always being an inspiration and the strongest person I know, to my brother and sisters for all the love that you give, and to my nieces and nephews for all the joy you bring.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Social platforms are today's key channels of communication, dissemination of ideas, propagation of opinions and beliefs, and, overall, cultural exchange. The broad participation of users from every socioeconomic background on social media makes these platforms an ideal "laboratory" where to take society's pulse.

In the recent decade, we have seen a wave of scholarly publications examining sociological phenomena on social platforms. These publications often refer to the same data sets, and sometimes use similar basic tools to process these datasets, but, more often than not, rely on software tools developed by the authors' teams. The system we report in this thesis has been developed to support this kind of scholarly work by researchers who do not necessarily possess software engineering skills. To that end, our system has been designed to meet two key requirements. The first requirement is to enable users to comfortably experiment with data and construct subsets of data of interest to them, based on their disciplinary expertise, by "slicing" it, i.e., selecting specific values or ranges within a dimension, and "dicing" it, i.e., by selecting combinations of values from multiple dimensions. The second important requirement that our system strives to meet is to automate the data-science tasks frequently involved in analyzing social-platform data.

Our system, called SoDa-TAP$_{v2}$, integrates (i) a data-ingestion and analysis pipeline, through which posts are annotated with a variety of lexical, sentiment, personality, and influence analysis, and (ii) a visual query language

through which users can explore their data, review their properties through a variety of visualizations and evaluate their hypotheses through a variety of statistical-significance tests.

1.1 The Research Problem

Today, the ability to effectively interpret massive amounts of data has become a necessity across various domains, from education and healthcare to energy and finance. Despite this prevalence of data, it is often underutilized due to technical barriers. These barriers include a lack of people proficient in data interpretation tools, technical experts who find traditional data analysis methods intimidating, researchers who can't afford the development time required to write scripts for analyzing a particular dataset, and domain experts in areas like healthcare who lack programming skills.

Challenges in Data Utilization

The challenges in leveraging data stem from the complexity of two tasks:

1. **Data Ingestion:** Convenient ingestion and processing of data is often hindered by underlying technical complexities [54]. While off-the-shelf technologies facilitate convenient ingestion and transformation [2], they regularly fall short in terms of extensibility. Instead, custom pipelines, when incorporated with Python scripts and its extensive range of NLP and ML libraries, can significantly enhance the insights extracted from data. However, setups like this require developers to do manual work, for example, in case of a Kafka-Spark setup [105], create Kafka topics, trigger Spark jobs, and write separate scripts for ingesting each dataset.

2. **Data Querying and Visualization:** Non-experts often lack knowledge of query languages and data visualization software. Even for those with programming skills, it can be arduous to dedicate focused effort to extract insights from a particular dataset using conventional libraries like Pandas. Generalized programming languages aim to achieve a balance in query-expressiveness [63] to cater to a wide range of domains.

However, this often leads to a trade-off where query succinctness (the ability to express queries concisely) is compromised. This trade-off is a common aspect of language design, where the goal is to make the language versatile enough for various applications, sometimes at the expense of the ability to express domain-specific queries as concisely as a domain-specific language (DSL) could. As such, our requirement for a DSL specifically for hypothesis testing, a common element of research work, was left unfulfilled.

1.2 Data Democratization with SoDa-TAP$_{\text{v2}}$

Data democratization is a recently coined term that refers to the aspiration of mitigating these barriers by making data more accessible to a wider range of users, irrespective of their technical expertise. Data democratization refers to the practice of empowering individuals, regardless of their technical expertise, to access and manipulate data resources in order to gain insights relevant to their field of study and work. A key requirement of data democratization is that data, which is one of the most valuable assets of an organization, should be available to the average end user, and not just siloed within IT departments or available only to data scientists. This component has been widely studied in the domains of data privacy and transparency [125]. However, data accessibility entails more than just availability. A very crucial tenet of data democratization pertaining to accessibility is the provision of tools that enable "self-service analytics" [62]. Such tools, owing to their easy-to-learn and easy-to-use user interfaces, enable individuals without expert knowledge to engage in data analysis without feeling overwhelmed.

With our focus on tooling, this thesis aims to advance data democratization efforts in the following ways.

- **Reduce communication complexity between human actors:** We expect that the ability to independently analyze data will streamline the communication process between team members, including those of our lab, minimizing misunderstandings that typically arise from relaying

data requests [12]. In small teams like ours, where a data engineer usually manages the data needs, relying solely on this individual for data handling can lead to miscommunications regarding data requirements, as team members may struggle to accurately convey their needs.

- **Cut-down on turnaround time:** As per the philosophy of lean management [124], teams must reduce delays by stripping out unnecessary communication. Preparing, conveying, and responding to data requests leads to inefficient time management as (i) the elicitation steps cannot be avoided, and (ii) everyone's schedule must align with the data engineer's availability.

- **Support data experimentation and minimize data waste:** This centralized dependency on another individual also discourages exploratory data analysis and experimentation, as individual team members might feel pressured to refine their requests to avoid imposing on the data engineer's time. A more accessible system gives individual actors more confidence to play around with data and also minimizes data waste [1].

- **Empower domain-knowledge experts:** Empowering individuals to conduct data analysis on their own gives people with domain knowledge more control, as they can make inferences from data-analysis results and make necessary adjustments to queries based on their knowledge.

- **Relieve data scientists of redundant tasks:** Routine analytical tasks can be distributed across the team, allowing data scientists to focus on complex, value-adding activities. This approach optimizes the use of skilled data scientists' time, for example, by freeing them to develop advanced predictive models rather than generating reports.

1.3 SoDa-TAP$_{v2}$ Features

The first version of SoDa-TAP [46] focused on incorporating functionality for advanced social data analysis. However, the use of the tool was strictly suited

to data engineers or those experienced with data analysis tools. SoDa-TAP$_{v2}$ focused on the premise of making this system accessible to those with little to no programming experience, such that they could interact with the tool using a simplified interface that does not require expert knowledge. SoDa-TAP$_{v2}$ plays an important role in overcoming traditional data analysis challenges through innovative functionalities that cater to the needs of non-technical users, in the following ways:

Simplified Data Ingestion

- **Automated Data Import:** SoDa-TAP$_{v2}$ offers a highly simplified ingestion process, allowing end users to initiate data ingestion directly from the user interface by providing a simple, publicly accessible URL to the dataset, hosted on platforms like Google Drive.
- **Customizable Data-Analysis Pipeline**: The end user can toggle the specific analyses they want the pipeline to conduct. SoDa-TAP$_{v2}$ internally relays this configuration to the pipeline, ensuring that the analysis is tailored to user needs.

Intuitive Querying and Visualization

- **Visual Query Language (VQL):** SoDa-TAP$_{v2}$ introduces a domain-specific visual query language that allows users to construct queries through a drag-and-drop interface. This method drastically lowers the learning curve associated with traditional text-based query languages.
- **Flexible Data Manipulation:** Users can create, compare, and analyze data subsets using predefined blocks in VQL, such as filtering for specific attributes or comparing different datasets. This flexibility is vital for non-technical users who need to conduct complex data operations without programming.
- **Dynamic Visual and Statistical Comparisons:** Users can visualize their results directly within SoDa-TAP$_{v2}$ using a variety of charts, and conduct statistical tests on these results.

Ease-of-Use Features

- **Data Preview:** SoDa-TAP$_{v2}$ features a comprehensive data preview tab, with a table view and visualizations to help users understand the structure of their data before proceeding with deeper analysis.
- **Saving/Loading Workspaces:** The users can save a query workspace that they have created and return to it later.
- **Tooltips:** Each VQL block displays a tooltip upon hovering, explaining the usage of that block.
- **Help Pages:** Each tab on the user interface includes a help page that offers users an overview of how to use that particular tab.

1.4 Thesis Outline

The rest of the thesis is structured as follows: Chapter 2 discusses related research and existing tools in social media data analysis space, acting as a primer to the functionality of SoDa-TAP$_{v2}$. Chapter 3 delves into the functionality and design of SoDa-TAP$_{v2}$, explaining each feature and architectural component in detail. Chapter 4 provides an evaluation of the system's utility, generality, and performance, using various experiments. Finally, Chapter 5 summarizes the key aspects of the system and details the planned future modifications to the system.

Chapter 2

Background and Related Research

In this section, we recap relevant commercial tools that support social data analyses. Additionally, we detail research studies utilizing such tools as a foreground to understanding the utility of SoDa-TAP$_{v2}$.

2.1 Commercial Tools

There is a variety of commercially available tools that employ techniques similar to ours to study social media data.

Audiense [7] focuses on understanding groups within Twitter data, by segregating audiences based on their "psychographics, demographics, content they like, and sources of influence" [53]. This is based on social network analysis, that allows the user to assess the connections that exist within their audience, based on who engages with what content, and how the data flows across from one point to another, or from one audience subset to another. Once these distributed subsets with distinguished attributes are created, you can make targeted marketing efforts based on the characteristics of each subgroup. Similar to many other social media analysis tools, you can also monitor your competitors, as well as conversations about your brand. Although Audiense focuses primarily on Twitter, it can also be integrated into other platforms.

CrowdTangle [26] is a tool provided by Meta, that can be used to analyze content on Facebook, Instagram, and Reddit. It allows you to "follow" content

(e.g. tracking posts with specific keywords, topics, links, or trending content) across the entire platform, tracking as many accounts as you want (real-time monitoring). Not only does it allow analyzing content 'within' a platform, but also 'across' platforms, such that you can analyze how your content performed on different social media platforms, and how it resonated with the audience on each. Custom real-time dashboards can be created by the user as per their preferences, to assess multiple streams of data on a single dashboard, which includes creating reports and visualization. Its analysis capabilities are mainly focused on monitoring engagement rates and associated functionality like creating leaderboards and comparing performance metrics across different forms of content based on engagement. Additionally, the tool allows you to set up 'custom alerts'. For example, you might want to be alerted on your Slack channel once some content crosses a specific threshold for engagement rate. Its API is also accessible so that it can be integrated into other extended tools, leveraging its capabilities.

Keyhole [58] is a tool that allows you to monitor trends, hashtags, influencers, keywords, etc. It offers you specific functionality for each category; for example, for influencers, it can help you assess their return on investment (marketing efforts compared with engagement), and for content, it can conduct real-time sentiment analysis such that you can monitor the changing sentiment of your audience towards a series of posts you are making. As such, it can be particularly useful for brand monitoring, to assess how the consumer perspective towards your brand is changing, and what measures might need to be put in place to influence this change.

Brand24 [15] is a 'social listening tool' that allows businesses to monitor how they are being talked about across social media platforms. This includes content that mentions your brand or a product made by your brand, the users mentioning your brand, or any hashtags or links you might want to associate with your brand. The 'user detection' capabilities allow you to identify which major companies, competitors, or influencers might be mentioning your company or products so that you can engage with them accordingly. Perhaps an influencer bought one of your products from Amazon and gave it a negative

review on their channel. You can be notified of it as soon as possible, using custom alerts, and reach out to them to make amends. This is also supplemented by sentiment analysis capabilities, to monitor the general perception of your company and products.

2.2 Social and Behavioral Research using Analytical tools

Many research studies have used tools that analyze social data to gain insights into topics ranging from health and public opinion to behavioral trends and the spread of misinformation.

Social media analysis tools have been extensively used to explore topics in healthcare. Santarossa et al. [93] used the Netlytic [74] software to assess the prevalence of orthorexia nervosa on Instagram. The study examined the impact of social media on dietary behaviors, with a focus on how discussions tagged with #orthorexia reflect the broader issues of dietary perfectionism and its potential harms. Netlytic's capabilities to process large datasets allowed for detailed analysis of user interactions and content themes, providing insights into the social and psychological impact of online conversations about health on individuals, and the dominant role of social media in promoting health-related disorders among users. In Malhotra et al.'s research [65], tools such as Sprout Social [101], Symplur [113], and SocioViz [99] were used to study the impact of World Hypertension Day. The research provided insights into the regional interest in hypertension awareness from 2014 to 2022. Sprout Social and Symplur allowed the extraction of tweets and impressions, and SocioViz aided in conducting network analysis to understand connections between various hashtags and topics. The study found that World Hypertension Day had a huge influence on awareness regarding the issue, even noticing a spike of over 800% increase in 2021. Moens et al.'s study [73] aimed to evaluate the type, quality, and content of web-based information on spinal cord stimulation (SCS) for chronic pain. It used keywords like "pain" and neuromodulation" to identify relevant conversations about the topic on Facebook, Twitter, Youtube,

Instagram, and blogs, etc, using the Awario [10] tool.

Research during the COVID-19 outbreak further concretized the role of such tools in public health communication, particularly in pandemic management and emergency response. Obiała et al. [76] used BuzzSumo [18] to assess the accuracy of articles that were most frequently shared on social media platforms regarding COVID-19 prevention. The study analyzed thirty articles and classified them as accurate, misleading, or inaccurate in accordance with the health guidelines issued by authorities such as WHO and the CDC. It found that while most of the articles (80%) were accurate, they only accounted for 64% of the shares. BuzzSumo was also used in another study [69] by Mirone et al. to assess telemedicine information on social media during the pandemic. The paper suggested that online content by different medical institutions should be standardized to combat misinformation. Rovetta's study [90] used Google Trends [44] and Talkwalker [107] to examine concerns on the web about COVID-19 vaccines in Italy. It used the keyword search and filtering functionalities of these platforms to identify relevant conversations. The study indicated a general sense of doubt amongst the public regarding vaccines.

Besides healthcare and pandemic research, these tools have also been used to investigate the manner in which people communicate on social media in general. Gruzd et al. [45] used the Communalytic [22] tool to study anti-social behavior within online communities, especially on platforms like Reddit. The study examined issues like trolling, hate speech, and inflammatory messages made to disrupt conversations. Rohlinger et al. [89] used DiscoverText [28] to examine how Twitter's suspension of a few accounts during the 2020 presidential election audits affected open discourse on the topic. The study discovered that such suspensions had little to no effect on the quality of information shared, or the people who posted.

Each of these tools and their assisted research reflects the core goals of our system, allowing users to conduct social data analysis conveniently. The first version of SoDa-TAP implemented a comprehensive range of lexical, impact, and semantic analyses within a data pipeline. However, this setup required

manual configuration, including the setup of Kafka topics, connectors, triggering Spark jobs, etc. Additionally, the platform did not provide a user interface to make it convenient for end users to interact with the system. This rigidity necessitated the constant presence of a developer to customize ingestion scripts and to develop scripts for data analysis after storage. Moreover, the platform's filtering capabilities were quite limited, supporting only a set of static visualizations, offering little flexibility for end users to interact with and explore their stored data dynamically. SoDa-TAP$_{v2}$ addresses these limitations by handing non-expert users more control, to customize and trigger the pipeline conveniently, conduct additional analyses, store data, and play around with it using an easy-to-use Visual Query Language (VQL) and an intuitive user interface. Each of these improvements over the Soda-TAP platform will be detailed in length in the next section.

Chapter 3

Software Architecture and Implementation

SoDa-TAP$_{v2}$ combines a number of tools, as shown in the diagram of Figure 3.1, accessible over a browser-based user interface. The user interface contains four tabs: 'Data Ingestion', 'VQL Interface', 'Dataset Preview', and 'Plotting', each with a help page detailing its functionality. A user can log in to the system using the Auth0 [8] authentication service.

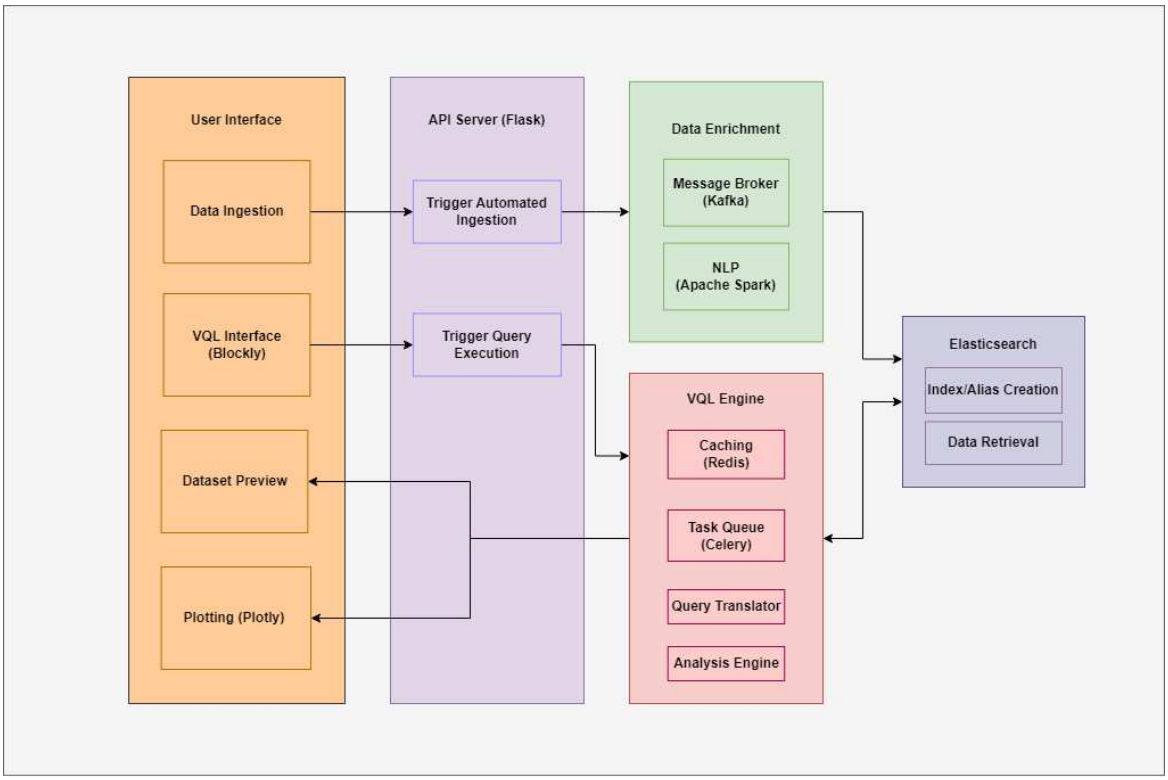


Figure 3.1: SoDa-TAP$_{v2}$ Software Architecture

3.1 Data Ingestion

SoDa-TAP$_{v2}$ relies on Kafka [3] for data transfer, Apache Spark [5] for processing, and Elasticsearch [31] for storing the analysis results (instead of CrateDB [25] which was part of the first version of the system). The system continues to use Docker [29] containers to simplify deployment. SoDa-TAP$_{v2}$ enhances the previous architecture with more automation, handing more control to a non-expert using the system. Section 3.1.1 describes the journey of a CSV file as it flows through the different tools of the data pipeline and lands in the data store. Section 3.1.2 details the types of analyses conducted on the data as it flows through the pipeline.

3.1.1 Automated Data Ingestion in Action

SoDa-TAP$_{v2}$ automates the data-ingestion workflow in four steps. (i) The file is downloaded to a designated "unprocessed" folder, which triggers the rest of the pipeline. (ii) Kafka is the mechanism for data transfer; a topic is created, and a connector reads the file from this directory into the topic. (iii) A Spark job is triggered to consume data from the Kafka topic, infer the schema, and perform analyses. (iv) Finally, the processed data is written to Elasticsearch for efficient querying.

Triggering the Data pipeline

The journey of the file through the pipeline starts with the user providing two inputs, to **trigger the pipeline**:

- a link to the dataset, typically a public Google Drive URL
- the name of the dataset

Upon clicking the "Begin Import" button, the size (up to 500MB) and format (CSV) of the file are checked and its column names are extracted. The user then chooses the analyses they want to be conducted on the dataset. SoDa-TAP$_{v2}$ supports a variety of analyses, including topic modeling, lexical analysis, impact analysis, high-order semantic analysis, and image analysis,

described in detail in Section 3.1.2. The user can choose to apply any (or all) of these analyses by toggling checkboxes on the UI. Once they click the "confirm" button, the pipeline is triggered, requiring no further user interaction.

Next, the Python "requests" [87] library downloads the data file from the specified URL and stores it in a designated "unprocessed" folder on our file system, ready for Kafka to pick it up.

Data Transfer with Kafka

Kafka remains the core of the data transfer mechanism in SoDa-TAP$_{v2}$. It has the following components:

- **Topic:** Acts as a message buffer for transporting data bytes.
- **Producer:** Sends data to a topic.
- **Consumer:** Reads data from a topic.
- **Connector:** Links Kafka with external services e.g. a database.

In the first version of the architecture, a topic and a CSV source connector had to be manually created each time a dataset had to be ingested. In SoDa-TAP$_{v2}$, these are automatically created using the "confluent_kafka.admin.AdminClient" [23] and the "Kafka Connect REST Interface" [57] respectively. The CSV connector reads the file in the "unprocessed" folder into the Kafka topic, ready to be consumed by Spark. To ensure smooth data movement, the following additional services are used:

- **Zookeeper [6]:** Provides state management to the Kafka cluster.
- **Schema Registry [94]:** Stores the data schemas for the data being transported by Kafka.

Data Analysis with Apache Spark

Apache Spark is an analytics engine for large-scale data processing, offering high-level APIs for distributed data processing and machine learning. In the first version of the system, Spark was deployed in local mode, which doesn't

allow distributed processing over multiple machines. In SoDa-TAP$_{v2}$, a Spark cluster is set up using Docker for future expansion to multiple machines, allowing for better resource isolation, fault tolerance, and scalability. The cluster setup includes the following components.

- The **master node** manages the cluster's resources and schedules tasks by distributing them among the worker nodes.
- The **worker nodes** execute the tasks assigned by the master node. Four worker nodes have been deployed, with 5G of memory and 3 cores each.
- Spark **Standalone Cluster mode [100]** is used, which is the default cluster manager that comes bundled with Apache Spark, removing the need to set up a separate software like YARN [92] or Mesos [91]. It provides robust scheduling and resource management capabilities across multiple worker nodes out-of-the-box, as well as a web-based user interface that we use to monitor the cluster.

SoDa-TAP$_{v2}$ triggers a Spark job using the spark-submit [106] utility, which provides the topic name and the list of required analyses to the cluster. Spark reads from the Kafka topic, infers the schema for the dataset, and performs the analyses on it. The results are written to Elasticsearch using the "org.elasticsearch.spark.sql" package and the "Elasticsearch-Hadoop connector" [49].

Data Storage with Elasticsearch

We have changed the system's data repository from CrateDB to Elasticsearch in SoDa-TAP$_{v2}$. Elasticsearch is a powerful, open-source search engine built on Apache Lucene [4], widely recognized for its speed, scalability, and robust querying capabilities. It is designed to handle large volumes of data querying in near real-time, making it an ideal choice for our data-intensive query language detailed in section 3.2. Elasticsearch uses an inverted indexing system, allowing for quick text searches, and supports complex search queries with its DSL (Domain-Specific Language). Its distributed nature allows for the data and search load to be spread across multiple servers or nodes. Our Elasticsearch setup has the following components:

- **Nodes:** We have deployed Elasticsearch using Docker, with a cluster of three nodes, aiding efficient querying. All nodes currently run on a single machine. However, our deployment is suitable for a multi-machine setup to allow data redundancy, fault tolerance, and high availability.

- **Document structure:** For the functioning of our VQL, we maintain a flat document structure within our indices to simplify querying. As such, each document contains a denormalized set of fields with no parent-child relationships or nested fields. This simplicity keeps the VQL implementation straightforward and ensures that non-expert users do not have to deal with complex queries involving nested structures.

- **Kibana [59] Integration:** We used Kibana for its Developer Tools to create and explore indices during development, and to upload small files during VQL testing.

3.1.2 Data Processing and Analyses

SoDa-TAP$_{v2}$ features a comprehensive set of analyses, conducted primarily within Apache Spark, to enhance input data before it is written into Elasticsearch. SoDa-TAP$_{v2}$ includes the following text-analysis functionalities, originally available in the first system version.

Preprocessing: This stage cleans the text by eliminating non-essential elements like stop-words and standardizes it to lowercase.

Dictionary look-up: A number of dictionaries are used to identify words tied to personal values, sentiment, humor, big-five personality traits, and emotions.

1. Personal Values: Words categorizing personal values into ten categories: self-direction, stimulation, hedonism, achievement, power, security, conformity, tradition, benevolence, and universalism, based on the framework proposed by Ponizovskiy et al [81].

2. Sentiment: A binary classification of sentiment (positive and negative) as proposed by Hu and Liu [50].

3. Humour: Words categorized into four levels of humor intensity, as suggested by Engelthaler and Hills [35].

4. Big-Five Personality Traits: The Big 5 personality traits represent five dimensions of personality. This dictionary categorizes words falling into each of these five dimensions, namely "extraversion", "agreeableness", "openness", "conscientiousness" and "neuroticism".

5. Emotions: The dictionary categorizes words into a set of eight emotions, namely "anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", and "trust".

Sentiment Analysis: Captures the sentiment associated with a text using VADER [116].

Element Extraction: This functionality extracts commonly occurring elements like hashtags, emojis, and URLs from tweets, and does frequency analysis on them.

Engagement Calculation: The engagement rate, extended reach, and potential impressions of a tweet are calculated using tweet interactions and the author's following/follower counts.

Image Analysis: Version 1 provides a custom-built console application, not incorporated into the pipeline, that conducts color scheme analysis, object detection, image classification, sentiment analysis, OCR, and face recognition.

Data Analyses New in SoDa-TAP$_{v2}$

In addition to the above, SoDa-TAP$_{v2}$ features a number of additional analyses, described below.

LDA Topic Modelling: Topic modeling is used in NLP to discover abstract topics within a collection of documents. It clusters co-occurring words into topics, revealing latent thematic structures in the text. SoDa-TAP$_{v2}$ uses the popular Latent Dirichlet Allocation (LDA) algorithm [13] provided by "pyspark" [61] to categorize each document as a mixture of topics and to determine which words make up a topic. The topic-modeling process involves the following three steps.

- Vectorization: A tokenizer splits the preprocessed text for each document into words. A "CountVectorizer" [24] then converts these words into a sparse vector representation, resulting in a matrix of word counts for each document.
- Topic Discovery: With the text data vectorized, a Latent Dirichlet Allocation (LDA) model is used to discover a distinct number of topics within the corpus. The model iterates multiple times to optimize the topic distributions.
- Topic Assignment: Once the LDA model is fitted, the topic distribution for each document is retrieved, and the most dominant topic for each document is also determined.

Fact vs Emotion Image Classification: Social-platform users often include images with their posts, including personal images, memes, infographics, etc. These images play an important role in how users engage with each other and researchers are often interested in exploring the nature of the images and their impact. This is why SoDa-TAP$_{v2}$ integrates a console application to detect if an image depicts facts, emotions, or both. It uses Google's Cloud Vision API [21], specifically its face detection, text detection, and labeling features to label tweet images. If people's faces are the dominant element of the image, whether the image contains one big face or many smaller faces, the emotions detected by the API are examined: if any face is rated "likely" or "very likely" to express an emotion like joy, anger, surprise, or sorrow, the image is labeled as "emotional." To assess whether faces are a dominant image

element, we calculate the percentage of the image covered by the bounding boxes of the detected faces. If this percentage is above a threshold, facial information is determined to be a dominant feature of the image. We typically set the threshold to 1%. Although this percentage may appear to be low, it pertains to the bounding box of faces only and not the whole person. If faces are not a dominant feature or the faces do not show clear emotions, we examine whether the image has a text element. If the bounding boxes of all text segments cover more than 20% of the image, the text feature is considered "dominant." If they cover between 10% and 20% of the image, the text feature is deemed "relevant." The detected text is passed to the OpenAI API for classification for both relevant and dominant texts. If the text is dominant, this classification is applied to the whole image. If the text doesn't dominate the image, we consider the array of objects and notable attributes detected by the Google Cloud Vision API. The array elements are classified as "positive," "negative," or "neutral" using the "finiteautomata/bertweet-base-sentiment-analysis" HuggingFace model [38]. If any element is tagged as "emotional," whether "positive" or "negative," the image is classified as emotional. Our rationale is that if the earlier steps (face detection and text detection) have not confirmed a label, any trace of emotion in the image is prioritized. This decision criterion is often validated by the label detection feature, which identifies sentiments like "happiness" or "sadness" rooted in the broader image context, which in turn is immediately labeled as "emotional" by the HuggingFace model. Finally, if up to this point, all our efforts have been unsuccessful, we are left with a list of labels. Each label is identified either as "factual," "emotional," or "both." The label marked as "both" can only have originated from the previous "relevant" text detection, while the "factual" labels are the result of the HuggingFace classifier. At this point, the image is usually labeled as "factual" because the "factual" tags are more common than the "both" tag, which could appear once (at most). In the rare scenario where there is a single "factual" label and a "both" label, the image is categorized as "both."

GPT 3.5 Turbo Instruct API integration: SoDa-TAP$_{v2}$ also includes a console application to classify text based on an OpenAI model's response. This console application was implemented to classify text as "emotional", "factual", or "both" for a research study done in parallel to the system's implementation (detailed in section 4.1) but is truly versatile and can be used for any other classification as well. The console application used OpenAI's GPT-3.5 Turbo Instruct API [71], optimized for task-oriented use cases with well-defined instructions, distinguishing it from the chat models. The text of each tweet was submitted to the model with the following concise prompt: "Does the following tweet seem like it was written based on facts, emotions, or both? Please output just one word: 'facts', 'emotions', or 'both'. Tweet: {tweet-text}." This prompt accompanied each API request to ensure the model retained the context of providing a one-word response and prevented the model from offering extended explanations that would require the need for post-processing. This method was favored over the alternative of giving a "system" message to the API to establish its role. After a brief period of trial-and-error with both techniques, sending the context with each request proved to be more precise for our specific use case. Consequently, the OpenAI API provided the label for each text. Post-processing was needed only in the extremely rare instances (in four cases) when the API, contrary to instructions, chose to provide explanations.

3.2 VQL Interface

Unlike traditional text-based programming languages, Visual Programming allows users to implement computing logic using graphics and images. Visual programming languages and interfaces [104][34][60][30] come in all shapes and forms, from blocks that mimic real-life objects to those that represent individual operators within a language. This visual approach benefits from the widespread familiarity with drag-and-drop interfaces, thanks to their use in introductory programming courses and coding education for children, such as with Scratch [97][96]. To enable users to explore and test their hypotheses visually, we designed a domain-specific visual query language (VQL) for

hypothesis testing. Through an intuitive drag-and-drop interface, the SoDa-TAP$_{v2}$ VQL guides users through a sequence of steps, each one supported by a special-purpose block, toward a complete and correct specification of a hypothesis.

3.2.1 Query Builder

The VQL is implemented using Blockly [14], a client-side library that enables the creation of block-based visual programming languages. Blockly's interlocking blocks visually represent the flow of execution with statements and variables. Figure 3.2 shows our Blockly query builder interface with one of the tabs open. The dotted grid (labeled A) in the image is the query grid, where blocks can be dragged in to form a query. The block labeled "Workflow" (labeled B) on the grid is fixed and represents the starting point for a query. The query needs to be created within this root block. Any blocks not directly or indirectly connected to it will be disabled, and ignored upon query execution. On the right, there is a panel (labeled C) containing blocks separated into categories. Each category has a specific color allowing for easy identification of its blocks. As can be seen in the figure, the third workflow block (labeled D) already has its inputs pre-connected. This is a Blockly feature that allows pre-plugging common inputs for a block to aid the query setup process, reducing development time. Blockly by default provides additional features like duplicating, deleting, expanding, and collapsing blocks, and even retrieving them from a trashcan (labeled E) after deletion. We have added tooltips for each block that appear upon hovering, aiding with the learning curve. The two buttons labeled F and G in Figure 3.2 can be used to save and load a query workspace, respectively, in case the workspace is needed for future use, or for saving frequent queries.

Query Workflow Construction

The learning curve of a query language can be affected by the fact that there is a multitude of potential queries that the user can come up with that may not always correspond to a valid query within the language. This is due to their

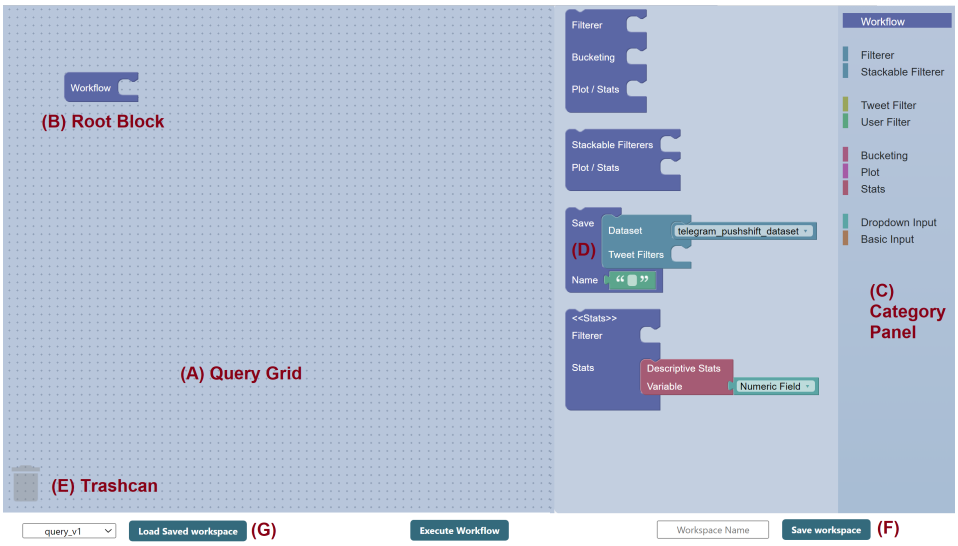


Figure 3.2: Query Builder Tab

often text-based and non-linear input structure. Structured query languages, such as SQL, mitigate this problem by imposing a fixed order in how queries are constructed (e.g. SELECT-FROM-WHERE structure), guiding the user through a logical sequence of operations that result in a valid query. This structured approach is indicative of declarative languages [121], where the focus is on the "what" of the data retrieval rather than the "how", making it easier to retrieve and transform the required data. Similar to SQL's logical sequencing, our VQL, designed with hypothesis testing in mind, introduces four default workflows. These workflows act as the starting point for the end-user, guiding them through a structured series of steps needed to conduct an analysis, significantly reducing the risk of errors or incomplete queries. The workflow blocks are tailored to cover the most common types of operations typically involved in data slicing and dicing for hypothesis testing. These workflows will be detailed in section 3.2.2.

Query Validity Evaluation

Ensuring query validity upfront, a practice usually referred to as "compile-time validation" as opposed to "run-time validation", is crucial for maintaining the integrity and efficiency of database operations. Compile-time validation checks the correctness of inputs during the query-building phase, before execution.

This preemptive validation approach helps avoid the execution of erroneous queries that can lead to system crashes or unexpected results.

Blockly provides built-in `setCheck`, `setPreviousStatement`, and `setNextStatement` features that are used to ensure query validity within the VQL. The `setCheck` feature allows developers to specify the type of data that should be accepted by a block's input connector. This is accomplished by assigning an array of acceptable data types to each input connector, ensuring that only compatible data types can be connected. For example, if a block is designed to perform operations on numerical data, its `setCheck` property can be set to accept only 'Number' data types. Attempting to connect a block that outputs a 'String' type would be disallowed, visually indicated by an inability to connect the blocks. The `setPreviousStatement` and `setNextStatement` features manage how blocks interlock with each other, defining which blocks can be sequentially connected. For example, for a date filter block, it can be ensured that the block connected to it is a date-selector block. These features help prevent runtime errors from type mismatches and guide users in constructing queries correctly by enforcing data type constraints.

Query Execution

When a user constructs a query in Blockly and hits the execute button on the front-end, a JSON representation of their query is created. Each line in this JSON corresponds to a specific statement block and its variables. The front-end then performs validation ensuring no required connections are missing, and no dropdown blocks are unselected. If the validation fails, the user is immediately alerted of the missing inputs. Otherwise, the JSON is sent to the backend for processing using jQuery

3.2.2 VQL Syntax and Functionality

The BNF grammar for our VQL is provided in Appendix A. The Backus-Naur Form (BNF) [120] is a notation technique used to describe the syntax of programming and data description languages. Each rule in BNF notation is written with a non-terminal symbol on the left-hand side and the derivation

it represents on the right-hand side, separated by the "::=" special symbol. The right-hand side can contain a combination of terminals and non-terminals. Non-terminals are typically enclosed by the <> symbol and terminals are enclosed by quotation marks. We occasionally add text within parenthesis to further explain a symbol in our BNF for readability. The simplicity and clarity of BNF allow language designers to unambiguously convey the structure of language constructs.

The VQL allows operations on two levels of granularity, the tweet level, and the author level. A tweet is equivalent to a single document in the Elasticsearch index. An author is equivalent to an aggregation of the author's tweets based on the author_id field. The words "tweet" and "author" are only used to simplify explanations and to keep the VQL in line with the social data analysis pipeline that precedes it. Otherwise, its implementation is not restricted to social media data. For example, the VQL would work just as well for a library analysis system, where a "book" would correspond to a single document instead of a tweet, and a "book genre" would correspond to an aggregation, instead of an author.

The user starts building their hypothesis by selecting one among four top-level **workflow blocks**:

1. The **data-subset-creation workflow** enables the user to flexibly slice and dice a dataset to produce a new dataset, which can be named and saved for further processing.

2. The **intra-dataset comparison workflow** guides the user to create distinct data buckets within their dataset, and compare them against each other.

3. The **inter-dataset comparison workflow** is similar to the one above, but instead of comparing data buckets from the same dataset, the user compares two different datasets against each other.

4. Finally, the **statistics workflow** allows the user to conduct statistical analyses on fields of a dataset, specifying the relevant field for each

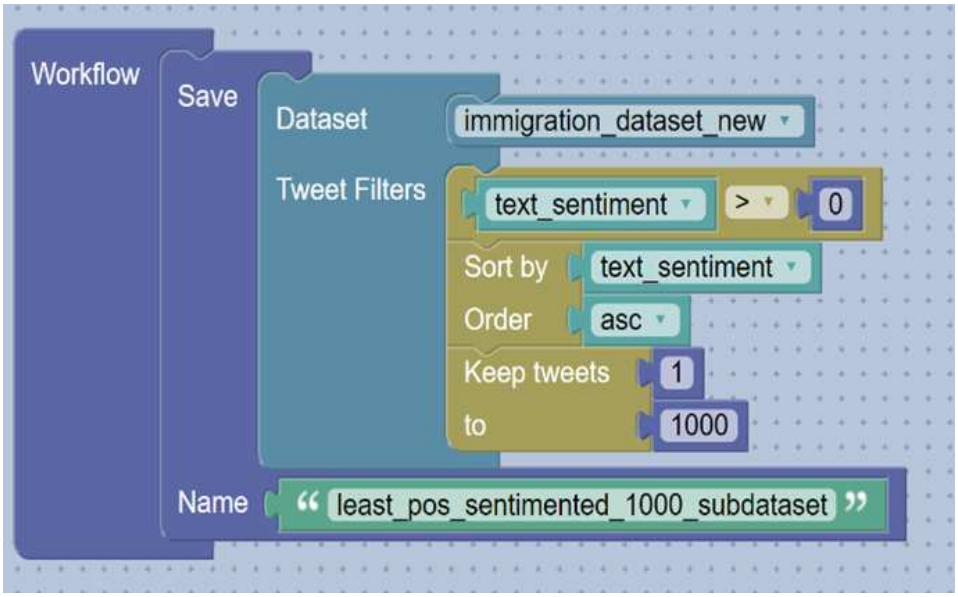


Figure 3.3: Data-Subset Construction

required input variable.

1. Example: data-subset-creation workflow (Filtering) Let us now review the first workflow, shown in Figure 3.3. Note that the workflow has two main inputs: the original dataset and the name of the data-subset to be constructed. In this example, the user chooses to filter the "immigration_dataset_new" dataset, based on which to construct the new data-subset. The workflow works almost instantaneously, using Elasticsearch's filtered aliases as its backbone. The user can choose to apply a "stack" of filters. Each **filter block** requires the name of the field on which it should be applied. Each filter in the sequence applies to the output of the filter preceding it; in effect, the filters are all applied in conjunction with the original dataset. Note that the field names are dynamically retrieved from the repository.

The SoDa-TAP$_{v2}$ VQL has seven **primary field name blocks**: "All Fields block", "Numeric field block", "Boolean Field block", "String Field block", "Data Field block", "Array Field block", and "Dictionary Field block". Each of these is constructed using a Blockly "dynamic dropdown field". The system dynamically populates these blocks with field names using field types inferred from the Elasticsearch index mapping. This ensures that the dropdowns are

always up-to-date and aligned with the current data schema. The All Fields block contains the names of all fields found in the index. The numeric, Boolean, and date blocks contain the fields with these respective data types in Elasticsearch. String, array, and dictionary types are not inherently distinguished in our indices, as they are all indexed under "keyword" or "text" types. To classify these accurately, the first document from the index is analyzed: if a field is a list instance, it is classified as an array; if the first character of the field is '{', it is identified as a dictionary; otherwise, it is treated as a string. Dynamically populating fields based on these types allows us to impose type restrictions for tailored operations specific to each data type, easing user interaction by minimizing manual input and reducing the likelihood of errors. There are four different types of **tweet-filtering blocks:**

1. The **range filter** is used to filter tweets based on a range of a numeric or a date field.

2. The **comparison filter** is used to filter tweets based on various operators ($=$, $>$, $<$, $\geq$, $\leq$, contains, does-not-contain), and can be used with all types of fields (the above operators are overloaded).

3. The **sorting block** applies an ascending or descending order to the chosen field and is typically used as a pre-step for the position filter.

4. The **position filter** selects a range of tweets, between two sentinels e.g. from position 15 to 200.

In our example, the user applies three filters on the input dataset. First, a range filter on the text_sentiment field of the tweets. This field is numeric and the user selects the tweets that have a positive text_sentiment value. Next, the user applies a sorting filter, ordering the tweets with positive sentiment in ascending sentiment order. Finally, the user selects the first 1000 tweets, with a position filter. In effect, this block has selected the 1000 tweets from the original dataset that express the least positive sentiment. In addition to filtering tweets, the SoDa-TAP$_{v2}$ VQL supports three types of **author filtering blocks**.

1. The **tweet count filter** selects authors whose number of tweets is within the specified numeric range. For example, select authors who have between 30 and 50 tweets.

2. The **author metric filter** selects authors based on an aggregation metric, i.e., average, median, minimum, maximum, sum, or median-absolute-deviation, of a numeric variable of all of the tweets for that author. For example, one may need to examine "generally happy authors" and they would do that by selecting authors for whom the average text_sentiment of their tweets is above 0.

3. The **count comparison filter** selects authors based on the comparison of two distinct range queries. This approach is particularly useful when outliers might skew the measures of central tendency used in the author metric filter. For instance, to identify generally happy authors, one might typically use an author metric filter to select authors whose average text_sentiment exceeds 0. However, this average can be influenced by extreme values. To address this, the count comparison filter can be applied to ensure a more robust selection by specifying that (the number of tweets with a text_sentiment greater than 0) (is greater than) (the number of tweets with a text_sentiment less than 0).

Lastly, the VQL supports saving a specific bucket using a **bucket-picker** block. Bucketing is explained in the next example (intra-dataset comparison). Essentially, this block allows you to create buckets within a dataset using a technique like clustering, and then indicate which specific bucket/cluster you want to retain/save as a data-subset.

2. Example: intra-dataset comparison (using Bucketing) Let us now move on to a second more complex example, to illustrate the intra-dataset comparison workflow. As discussed previously, the intra-dataset comparison workflow guides the user to create distinct data buckets within their dataset and compare them against each other. Examples of its usage are shown in chapter 4 (Evaluation).

Instrumental to the intra-dataset comparison workflow are the **bucketing blocks** that allow users to create data buckets within a dataset. These buckets can then be compared against each other using a plot or a statistical test. There are four types of bucketing blocks:

1. **Clustering** based on the tweet's text using two different clustering algorithms;

2. **Classification** of the tweets according to an input model;

3. **Continuous bucketing** based on a numerical field and one of a set of alternative methods for identifying breaks in the continuous range of values of this field; and

4. **Discrete bucketing** based on a categorical field of the tweets.

The **clustering block** requires that the tweets' texts be vectorized. As the vectorization model, we currently use Google Code's Word2Vec model [43], trained and fine-tuned on the Google News Dataset (100 billion words), named "GoogleNewsvectors-negative300". The vectorization process transforms every word in each tweet into a 300-dimensional vector. These vectors are then averaged to produce a 300-dimensional vector representation of the tweet. For author-level analysis i.e. if clusters of authors are required rather than clusters of tweets, the mean of each author's vectorized tweets is taken to get a 300-dimensional vector representation of each author. The vectors can then be fed to a clustering algorithm.

SoDa-TAP$_{v2}$ supports two clustering algorithms:

- **Flat centroid-based clustering (using k-means)** This can be run in two ways. Either the user can manually specify the number of clusters to be created, as is common with k-means, or use the automated k-means block. With the automated block, the k-means algorithm is run multiple times, using a different number of clusters each time, and the best number of clusters is automatically determined using the silhouette score.

- **Hierarchical Density-Based Clustering (using HDBSCAN)** HDBSCAN allows hierarchical clustering without having to manually identify the optimal number of clusters. Additionally, contrary to the centroid-based K-means clustering algorithm that assumes the clusters to exist within a Gaussian sphere, HDBSCAN can adapt to obscure shapes within the dataset, and also identify the data points that are noise [67][48].

The **classification block** takes advantage of the recent advances in LLMs (Large Language Models) to classify text using a variety of HuggingFace [70] models. For author-level analysis, this text is a concatenation of all of that author's tweet's texts. Although this approach works flawlessly for tweet-level classification, the author-level classification is currently problematic due to the input token limits of HuggingFace models. A better approach is required specifically for user-level classification. The models available in SoDa-TAP$_{v2}$ are listed below.

1. cardiffnlp/tweet-topic-21-single for tweet topic classification
2. yiyanghkust/finbert-esg-9-categories for finance topic classification
3. SamLowe/roberta-base-go_emotions for emotion classification
4. bucketresearch/politicalBiasBERT for political bias classification
5. martin-ha/toxic-comment-model for toxicity classification
6. papluca/xlm-roberta-base-language-detection for language classification
7. nlptown/bert-base-multilingualuncased-sentiment
8. cardiffnlp/twitter-roberta-basesentiment-latest for sentiment classification
9. mohameddhiab/humor-no-humor for humor classification
10. snlp/roberta-base-formality-ranker for formal/informal classification

11. cardiffnlp/twitter-roberta-base-irony for irony/non-irony classification

We also provide a list of distilled versions of these classification models, if available. Distilled models are optimized for faster performance, representing a compact "student" model to replicate the behavior of a larger "teacher" model, based on the same foundational architecture. This process of creating a student model reduces computational demands while maintaining similar, though not as great accuracy.

The **continuous bucketing block** allows the creation of buckets based on a continuous numeric variable. The user provides the numeric field based on which they want to create buckets. For author-level analysis, bucketing would be done based on the average of that numeric field across all the author's tweets. The user selects one of the following methods to create the buckets.

1. **Jenks Natural Breaks** [**122**] minimize the variance within clusters and maximize the variance between them. It is particularly useful for identifying natural groupings in the data that aren't obvious just by looking at the numbers.

2. **Uniform Breaks** divide the data into evenly spaced intervals from the minimum to the maximum value.

3. **Quartile Breaks** organize data into four equal parts based on quantiles. This approach ensures that each bucket contains a roughly equal number of data points but may vary widely in the range of the data values. This is particularly useful in scenarios when Jenks Natural breaks or Uniform breaks might produce buckets with disproportionate distributions, where some buckets might be overly dense while others are sparse.

The **unique bucketing block** categorizes data based on discrete variables. This can include categorical data such as author IDs, the number of modalities of a tweet (likes, comments, hashtags, etc.), or a non-numeric attribute that distinguishes one piece of data from another.

3. Example: Plotting and Statistical Analyses Once the user has constructed the subsets of interest, either using a data-subset-creation workflow or through buckets created using any of the previously mentioned methods (clustering, classification, continuous breaks, unique breaks), these buckets can now be compared against each other based on any user-selected variable, using one of the **plotting blocks** available: pie-charts, single-variable plots like boxplots and violin plots, plots with two variables like scatter plots, faceted box plots, faceted violin plots, line plots (with a date type being one of the variables), and faceted scatter plots. The plots with two variables aid in assessing the effect of a moderating variable, which in this case would be the variable based on which the bucketing was done. Similarly, after the buckets have been created, they can be compared using statistical tests. Currently, the system has blocks for the following statistical analyses on variables from the buckets:

- **Covariance:** Measures the directional relationship between two variables, indicating how much the variables change together.
- **Correlation:** Provides a scaled version of covariance that measures both the strength and direction of the linear relationship between two variables.
- **T-test:** Assesses whether the means of two groups are statistically different from each other. This is useful for hypothesis testing between two conditions or treatments.
- **ANOVA (Analysis of Variance):** Tests differences in means across three or more groups, helping to determine if any of the group means are significantly different from each other.
- **ANCOVA (Analysis of Covariance):** Similar to ANOVA, but it includes one or more covariate variables that you control for, improving the accuracy of the analysis.
- **MANOVA (Multivariate Analysis of Variance):** An extension of ANOVA that can analyze multiple dependent variables simultaneously,

determining whether the vector of means of several variables is different across groups.

- **MANCOVA (Multivariate Analysis of Covariance):** Combines the principles of MANOVA and ANCOVA, analyzing multiple dependent variables while controlling for one or more covariates.

3.3 The VQL Engine

The SoDa-TAP$_{v2}$ backend APIs are powered using Flask [39], a lightweight WSGI [119] (Web Server Gateway Interface) web application framework. Flask blueprints [72] are used for the backend design. A blueprint is a Flask construct that allows the separation of a Flask application into separate related components. This ensures that our API implementation is highly modular and easily extendable.

The query executed by the user on the front-end is handled by a Flask endpoint upon reception. This endpoint checks if the user's query has already been executed against a Redis [86] cache. After determining whether a fresh query execution is necessary or not, it either sends back the cached results or relays the new query to the Celery Task Queue.

SoDa-TAP$_{v2}$ uses Redis for caching query responses. Redis is an in-memory key-value database, cache, and message broker. The JSON from the user's query is hashed using MD5 to create a unique and consistent key for caching. The key is used to check if a response for this particular query already exists. If there is a cache hit, the response is immediately returned to the front-end, bypassing the need for re-execution. Caching is also implemented at a fine-grained level, such that portions of a query are also cached. If a portion of a query containing a series of commands has been executed before, the result for that specific portion of commands will be returned directly from the cache for a future query that has the same series of commands as a subset, for the execution of subsequent commands in the query.

3.3.1 Task Queueing

SoDa-TAP$_{v2}$ uses Celery [20], an asynchronous task queue based on distributed message passing, for queueing and managing query requests. Celery is particularly suited for tasks that require execution in the background as it does not block the main thread of an application while the task is being processed. Flower [40], a Celery monitoring application, is used to monitor the status of the Celery workers and tasks.

Once a query is submitted and there is a cache miss, a Celery worker is assigned the task for execution. At this point, the front-end is informed that the task has been successfully accepted. The front-end then begins pinging another Flask endpoint at half-second intervals to retrieve the state of the Celery task. This endpoint informs the front-end whether the task is:

- Completed: In this case, the result of the query is returned.
- Pending: If the task is pending, the user has the option to terminate it using another endpoint, which is particularly useful in cases where processing a huge dataset takes excessive time.
- Failed: If the task fails due to an error, the front-end is informed so the user can be alerted.

The incorporation of this task-queueing component with the help of Celery allows the VQL engine to handle multiple simultaneous queries, across different browser tabs or sessions.

3.3.2 Query Translator

Each line in the JSON representation of the user query sent from the front-end corresponds to a distinct command. The Query Translator is a function within the VQL Engine that maps these commands to distinct functions in the Analysis Engine. Each command is processed sequentially, from the first JSON line to the last. First, parameters are extracted from the command, and then, a function corresponding to the command within the Analysis Engine is called, passing in the extracted parameters.

3.3.3 Analysis Engine

Data-subset Creation

A data-subset (or data slice) is one of the primary concepts within our VQL. In a typical data analysis script, data transformation often yields intermediate data representations, on a particular line of code. Traditionally, these intermediate representations are transient, recalculated with each execution of the script, and restricted within a particular script. This inability to easily reuse these slices across different analyses or scripts results in redundant processing and a lack of fluidity in data exploration. The ability to do so would be particularly useful within our domain of hypothesis testing, where the aim is often to compare multiple groups/slices against each other.

The backbone of this feature within the VQL is an Elasticsearch "filtered alias" [33]. A plain Elasticsearch alias is a reference or pointer to one or more indices and can be thought of as a soft link to the actual resource, rather than a duplicate. A filtered alias is a type of alias that has a filter applied in its definition, such that every time the alias is called upon, it first applies the specified filter in its definition to the index and then returns that specific subset. Alias creation in Elasticsearch is notably swift, providing near-instantaneous referencing of the defined subset.

Once the user has selected specific filters to define a data slice using the VQL's "data-subset-creation" workflow, these filters are embedded into the alias definition. This filtered alias is then presented to the user as if it were any ordinary dataset, allowing for analysis or comparison with other data slices. When the user performs an operation on this filtered alias, Elasticsearch applies the filters within the alias definition behind the scenes to acquire that exact data slice from the original index before further processing. Elasticsearch's caching mechanism plays a crucial role, ensuring that subsequent accesses to the same alias are expedited, avoiding the need to reapply filters or reprocess the entire initial index every time.

Querying the Elasticsearch Repository

Our decision to opt for Elasticsearch as the data repository in SoDa-TAP$_{v2}$, replacing CrateDB, was influenced by the availability of the "Elasticsearch-dsl-py" library [32], a high-level library providing a Pythonic way of running queries directly within Elasticsearch. The Elasticsearch DSL offers an ever-increasing variety of filtering and aggregation features that were deemed necessary for current and future releases of the system.

Pushdown processing with Elasticsearch-dsl-py: SoDa-TAP$_{v2}$ is intended to support the analysis of large datasets; therefore, it was crucial that the processing be done closest to the data source, aligning with the concept of *pushdown processing* [83][82]. This technique involves applying possible data transformations directly at the database level, reducing delays caused by the need to fetch huge amounts of data to be operated on elsewhere, often losing out on the processing optimizations offered by modern search engines. By implementing pushdown processing, user-built queries are transformed into queries understandable to the data store and sent to it for optimized data retrieval [118]. The analysis engine, instead of fetching the data and then transforming it inside Python, uses elasticsearch-dsl-py to carry out the first few steps of data processing directly within Elasticsearch, i.e. operations like filtering, sorting, aggregating, applying aggregation filters, or molding data into a particular format necessary for further processing. As a result, the size of the returned data is much smaller and often closer to the final result than otherwise. These benefits are augmented by Elasticsearch's out-of-the-box query caching and distributed data handling capabilities.

Query Optimization with Elasticsearch-dsl-py: In traditional data processing scripts, operations are often executed sequentially. Each line of code, such as a single filter triggers an immediate operation for its result to be utilized in subsequent operations. This approach limits the abilities of modern libraries and search engines to optimize query execution e.g. executing multiple filters in a single dataset pass. Query Optimization involves the evaluation

of multiple potential execution plans to select the one that minimizes the cost in terms of processing time and resource consumption.

Utilizing Elasticsearch-dsl-py, SoDa-TAP$_{v2}$ embraces a lazy evaluation approach to execute search queries, creating a "search object" to encapsulate multiple operations by chaining them within a single request. The VQL engine gathers all user-placed filters, splits, aggregations, aggregation filters, and sorting functions, as well as operations needed to mold a request-response into a specific input format for subsequent operations, into a single search object in the back-end. When a subsequent operation like bucketing requires the result of this combined search object, a single API request is made to Elasticsearch at that specific moment, rather than multiple, isolated requests at a time when the result might not be required. Elasticsearch intelligently optimizes the execution of the search object by reordering operations internally to minimize processing time and resource consumption. Also, since the entire query is sent to Elasticsearch as a single unified request, there's a reduction in latency, and the need to wait for one operation's completion before starting another is eliminated.

The Elasticsearch querying capabilities of the analysis engine make use of the following query and aggregation constructs:

- Range Queries [85]
- Term-level queries [109] like term [108], wildcard [123], and terms [111]
- Bucket aggregations [16] like variable_width_histogram [117], filter [37], and terms [110]
- Pipeline aggregations [78] like bucket selector [17] and extended_stats_bucket [36]
- Metric aggregations [68] like avg [9], cardinality [19], and top_hits [114]
- Scripted Metric Aggregations [98] in multiple phases (init, map, combine, reduce)

Bucketing

The bucketing functionality in the Analyzer engine, used for creating buckets within a dataset to compare them against each other, or save a particular bucket for future use, utilizes a variety of Python modules and libraries. The primary ones used are shown in Table 3.1.

Table 3.1: Libraries used for the bucketing functionality

Module/Library	Used for
Pandas [77], Numpy [75]	Data Manipulation
jenkspy [56]	Jenks Natural Breaks
gensim.models [41]	Loading word2vec model
transformers [51]	Classification
sklearn.cluster [95]	K-Means
hdbscan [112]	HDBSCAN

Statistical Analyses

SoDa-TAP$_{v2}$ includes the statistics modules listed in Table 3.2 below. The purpose of each measure/test is explained in Example 3, in section 3.2.2

Table 3.2: Libraries used for the statistics functionality

Module/Library	Measure/Test
elasticsearch-py [42]	Descriptive Statistics
elasticsearch-py [42]	Correlation
elasticsearch-py [42]	Covariance
sklearn.cluster [95]	T-test
statsmodel.api.stats [103]	AN(C)OVA
statsmodels.multivariate.manova [103]	MAN(C)OVA

3.4 Interactive Dataset Exploration

The Dataset Preview tab provides the user with high-level insights into the available datasets, including pie charts, treemaps, and a table view. A drop-down in the toolbar lists available datasets fetched from Elasticsearch. A refresh button is provided to fetch the newly created sub-datasets and update the drop-down with them without needing a page-reload. The toolbar consists

of analysis icons representing the type of analysis that will be done on the entirety of the dataset, along with the type of chart that will be generated, indicated by a mini-icon on the bottom right of the image. The name of the analysis is also displayed upon hovering over the icon. The toolbar can be seen in Figure 3.4, consisting of the following buttons, from left to right:

- dataset selection dropdown and dropdown refresh button below it.
- table view of the entire dataset.
- pie-charts of the distribution of hashtags, handles, and emojis, respectively.
- treemaps representing distributions of emotions, big-five personality traits, personal values, and humor respectively.
- sunburst charts representing distributions of emotions, big-five personality traits, personal values, and humor respectively.

Figure 3.4: Dataset Preview toolbar

Table View Figure 3.5 shows the table view. The table view is created using DataTables [27], a jQuery plugin that transforms standard HTML tables into interactive tables capable of handling large datasets with features like pagination. We use its server-side processing feature to dynamically populate the column names using the Elasticsearch index mapping. The rows are filled by fetching a sample of documents from the index, as specified by the "show n entries" drop-down and the page number selected, on-demand, for efficiency.

Visualizations Figures 3.6 and 3.7 demonstrate an example of a treemap and a sunburst chart respectively, retrieved for a dataset regarding immigration. The plots are interactive and can be zoomed in and out, and a category can be clicked to expand it for a closer look.

Show 10 entries

author_followers_count	author_following_count	author_id	author_name	author_verified	created_at	id	lang
734	2039	177134474	Terrance Wishart	FALSE	2020-11-21T12:54:15....	1330132418687332353	en
64	729	933763559359594496	Ed Hale	FALSE	2020-10-22T22:06:02....	1319399643373998081	en
33	35	3100039587	Peter Hopley	FALSE	2020-11-24T12:55:20....	1331219852690853891	en
5996	1186	141009810	Earth Soldier Lifestyle	FALSE	2020-10-08T14:38:37....	1314213616543911939	en
17	101	1296946097231814656	Hans Peter Meyer (aka...	FALSE	2020-11-21T06:27:58....	1330035206103953410	en
65	89	972478664	Denise Wanchulak	FALSE	2020-12-16T17:13:32....	1339257363476312064	en
5996	1186	141009810	Earth Soldier Lifestyle	FALSE	2020-12-15T17:27:40....	1338898533869563904	en
74	426	1176538423303319552	Virtual Nonsense	FALSE	2020-10-08T14:18:35....	1314208573165981703	en
527	771	3550412360	kara	FALSE	2020-11-20T18:46:43....	1329858728477171713	en
3502	426	1044738781171961856	Greg McLean	TRUE	2020-11-13T19:37:34....	1327334812567302144	en

Showing 1 to 10 of 298 entries

Previous 1 2 3 4 5 ... 30 Next

Figure 3.5: Table View

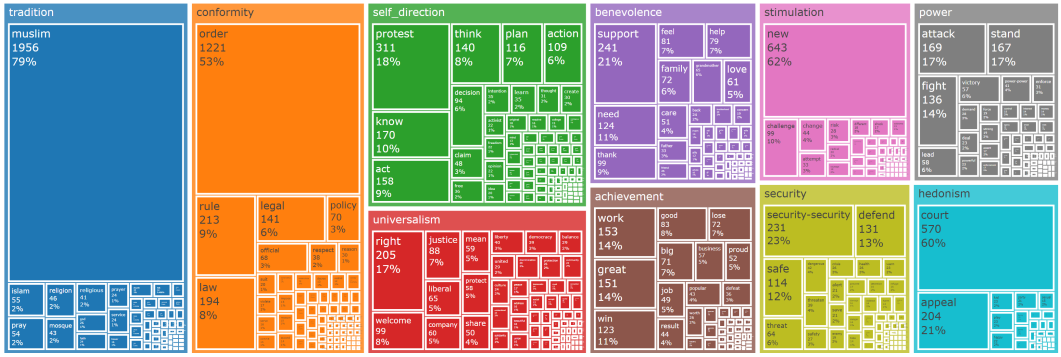


Figure 3.6: Sample Treemap for Personal Values

The plotting in SoDa-TAP$_{v2}$ is done using "Plotly.js" [80], a charting library that enables the creation of highly interactive visualizations. The current system iteration renders the plots directly on the client-side, allowing for the convenient interactivity features that "Plotly.js" offers out-of-the-box. This includes features like zooming, panning, saving plots, hiding/showing portions of the plot, and hovering for detailed insights. However, this interactivity comes at a cost, since client-side rendering can occasionally put an unnecessary burden on the client for massive datasets. To minimize this burden, we ensure most of the data manipulation is done on the server-side before the data is sent to the client, only requiring the client to do minimal data manipulation to mold the data into a format necessary to generate the plot. Additionally, to overcome this problem, we have implemented two alternative server-side rendering approaches, one available in the current iteration and the other as our preferred approach for the next SoDa-TAP$_{v2}$ iteration. In our first al-

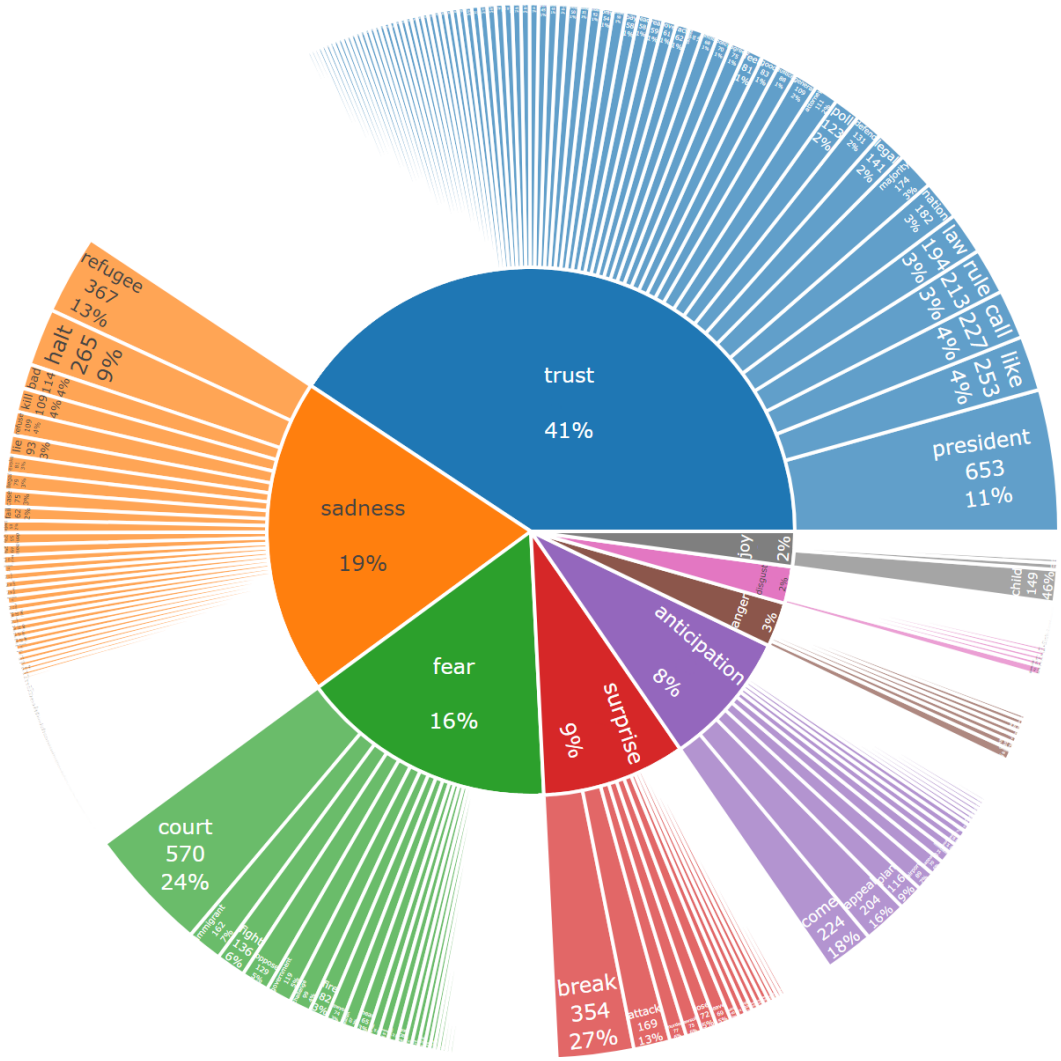


Figure 3.7: Sample Sunburst Chart for Emotions

ternative approach, we use "Pyppeteer" [84] to perform server-side rendering with a headless browser that renders the plot and saves its HTML content. This HTML content is then sent to the client to be displayed. This approach currently works with all our plots but unfortunately causes them to lose out on their interactivity, only displaying a static image. The second alternative, which will be our preferred approach in the next iteration, is currently implemented for some of the plots. It generates the plots on the server-side in Python, using the "plotly.express" [79] package, which allows the creation of HTML that includes the interactivity component. The client-side can then access this interactive plot generated on the server-side. Additional exam-

ples of the generated plots available in SoDa-TAP$_{v2}$ can be seen in Chapter 4 (Evaluation).

Chapter 4

Evaluation

We used our system to analyze two datasets, listed in Table 4.1, with three separate studies. Our work with the first dataset was formative for our system: the study (Energy Conversations) was conducted in parallel with the system development and provided requirements for its development. The other two studies (Immigration Conversations, and Replication Study) were evaluative and were conducted after the system development was completed. The "Immigration Conversations" study used the "Immigration-Sample" dataset, and the "Thesis Replication" study used the "Energy-East" dataset from the first study.

In total, we conducted four experiments: (i) a study of the opinions and sentiments of Twitter users discussing energy projects; (ii) a brief comparative analysis of influential and non-influential users discussing the Immigration and Travel Ban announced by the Trump administration; (iii) the replication of an undergraduate thesis-level Twitter study; (iv) a systematic performance evaluation.

4.1 Energy Conversations

This study was conducted in parallel with the development of the system, and, to a degree, informed the design of its functionalities. It is currently under review in an RSO (Research in the Sociology of Organizations) [88] volume titled "Moving beyond the microfoundations and macrofoundations of institutions".

Table 4.1: Datasets

Dataset	from (DD-MM-YYYY)	to (DD-MM-YYYY)	#Tweets	#Authors	Description
Energy-East	07-06-2013	16-06-2021	144,561	34,864	The dataset (retrieved using twarc2) contains tweets regarding the controversial energy-east pipeline project in Canada.
Immigration-Sample	30-01-2017	20-04-2017	1,000,000	526,216	The dataset is a sample of a Harvard Dataverse dataset [64] containing tweets related to the Trump Administration's immigration and travel ban executive order.

In this study, we explored what people had to say about the Energy East Pipeline, as part of a broader project on energy system transformation and contestation [47]. TransCanada Pipelines announced the 4,600 km project on August 1, 2013, to carry 1.1 million barrels per day of diluted bitumen from western Canada (primarily Alberta's oilsands) and North Dakota US to refineries in central Canada (primarily Quebec) and New Brunswick on the east coast. Other highly controversial topics being contemporaneously debated were the oilsands' greenhouse gas emissions and tailings ponds, expansion of the Keystone XL pipeline in the US, pipelines leaking into the Great lakes, and Indigenous sovereignty over their lands. This project was emblematic of these

broader issues and, thus, became heavily debated among multiple groups on economic, political, environmental, and moral grounds. Although the project ultimately was cancelled by TransCanada on October 5, 2017, #energyeast continued to feed pipeline debates for years afterward.

Data Collection

We utilized the Twitter API's full-archive search endpoint to identify all tweets with the "#energyeast" hashtag. Using the tweet IDs and twarc2 [115], we rehydrated these tweets and their attached images (if any). The resulting data set contained a total of 144,561 tweets published between June 7, 2013, and June 16, 2021. Included in this corpus were 28,319 tweets, 3,695 tweets with an embedded quote, 107,867 retweets, and 4,680 replies.

Tweet metadata

We parsed and formatted the resulting set of tweet objects and then extracted and computed: (i) the tweet's text; (ii) its follow-up metrics: retweet_count, reply_count, likes_count, quote_count; (iii) created_at timestamp; (iv) the URL(s) included in the tweet and their count (if any); (v) the hashtag(s) included in the tweet and their count (if any); (viii) mention(s) of other users included in the tweet and their count (if any). For each tweet, we computed its character count with all the elements present. If a URL(s) was seen during this process, we counted it with a fixed value of 22, because of Twitter's URL shortener. We also calculated three influence metrics: engagement rate considers retweets and likes (i), or retweets, likes, replies and quotes (ii), divided by the number of followers of the tweet's author multiplied by 100. The extended reach metric is based on the number of retweets divided by the total number of tweets by the author, multiplied by 100. The user's potential impressions metric is based on the number of the user's followers multiplied by the total number of the user's tweet count [46][102]. We also gathered descriptive information about the user who posted the tweet, including (i) their public profile metrics: followers_count, following_count, tweet_count, listed_count; (ii) whether the user was verified; and (iii) their screen_name. Based on these variables,

we examined the tweets' multimodal combinations of text and images. Malik et al. [66] analyzed combinations of three modes: text, URL links, and photos. We expanded on this to create a modality index ranging from 1 (for a tweet that has only text including the hashtag from the initial search "#energyeast") to 6 (for a tweet that includes text, URLs, hashtags (different than "#energyeast"), user handles, emojis, and photos).

Sampling tweets with text and image

Focusing on original tweets only (to eliminate duplicate images in retweets), a total of 5,696 images were identified. To simplify the analysis and preclude the introduction of confounding variables implicit in the influence of multiple images within a tweet, we ignored tweets with links to multiple images, which left us with 5,011 images. Finally, we sorted the corresponding tweets chronologically based on the tweet's creation timestamp, and we sampled every fifth tweet to collect the dataset of 1,000 tweets, along with their accompanying text, images, and the influence metrics defined above, i.e., engagement, extended reach, and possible impressions.

Labeling text and images

We used OpenAI's GPT-3.5 Turbo Instruct API to label tweet text, and Google Cloud Vision API along with a HuggingFace model to classify images. The procedure for text labeling and image labeling are detailed in 3.1.2, under the headings "GPT 3.5 Turbo Instruct API integration" and "Fact vs Emotion Image Classification" respectively.

1) What types of users frequently tweeted about #EnergyEast, and what did they talk about? We used the author tweet count filter to create a data slice containing tweets by users who posted more than 100 times, as shown in Figure 4.1. We then used the table view on our preview tab to observe the types of authors and what they were talking about.

We noticed that although there was a total of 144,561 tweets in our corpus, the conversation was highly concentrated among a few accounts. Only 141

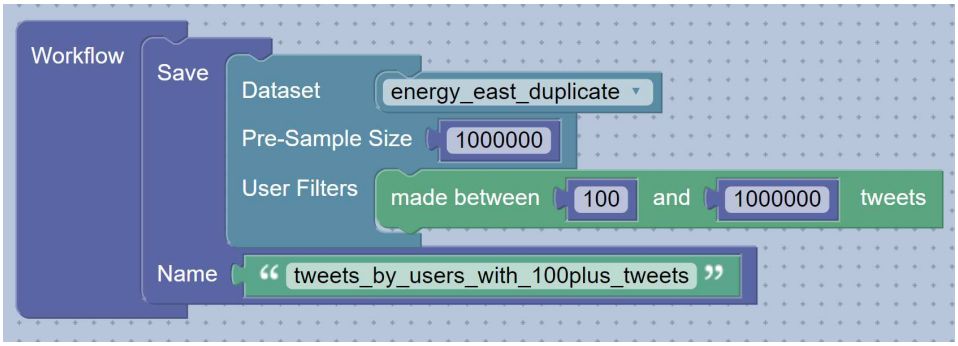


Figure 4.1: Create data slice of tweets by users with 100-plus tweets

users (re)tweeted about #energyeast more than 100 times during our study period. Of these, 103 were individual people, 35 were organizations, and three were suspended or deleted accounts. This concentrated group of users also tended to take political positions towards the pipeline: 45 (32%) were environmental activists and 50 actively criticized all politicians (28%), conservative politicians (5%), or Prime Minister Trudeau specifically (10%). There were fewer moderate voices: eight supported Indigenous rights (6%), seven were media outlets (5%), five supported sustainable development (4%), five were energy activists (4%), one was a government agency (Canadian Energy Regulator), and one was a politician.

2) How does tweet length affect viewer engagement on #EnergyEast Tweets? We used our "inter-dataset comparison workflow", along with the "quartile breaks" block to see the impact of character count on four fields; number of likes, number of retweets, number of quotes, and number of replies. The query for observing the number of likes within quartiles is shown in Figure 4.1. The variable in the boxplot can be changed for the other fields.

Figure 4.3 shows the results of the queries for each variable. We found that tweet length exhibited an inverted U relationship with viewers' responses. In our preliminary analysis, these boxplots are largely overlapping, and the differences are not statistically significant. Follow-on multivariate analysis will include other user and tweet-level variables to sharpen our analysis. It appears that relatively shorter original tweets, albeit not too short (i.e., tweets in the

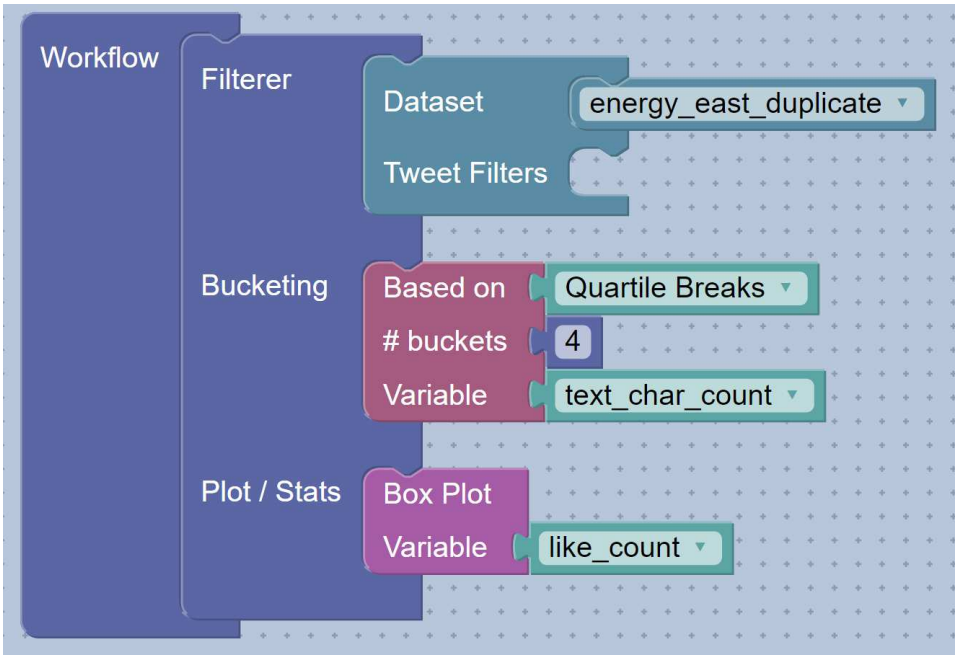
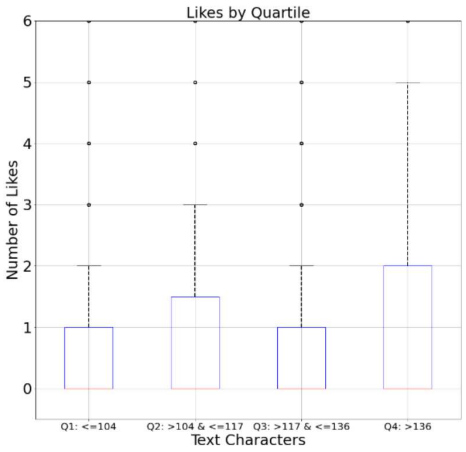


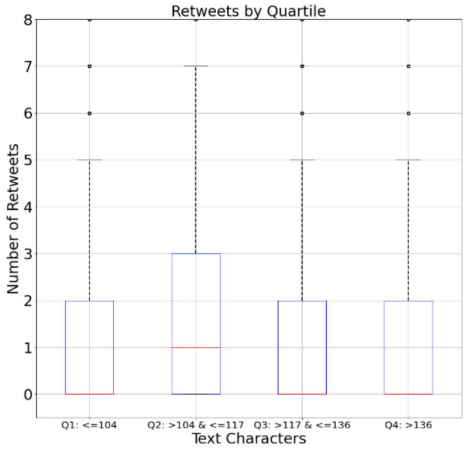
Figure 4.2: Compare number of likes in quartiles, based on tweet character count

second quartile in terms of length), and tweets with a high character count (i.e., tweets in the fourth quartile in terms of length), both tend to garner larger response rates. There are potentially two reasons why tweets that were neither too short nor too long were seen as more engaging: (i) they were text-only messages, which made them simple to digest; and (ii) they contained text plus several URLs, which artificially increased their character count without complicating their message. This may be because they thought shorter texts would be easier to read, parse, and understand. Hence their high follow-up interaction particularly with semi-short messages could allow the user to have a faster reaction time while not spending too much time trying to understand a long text.

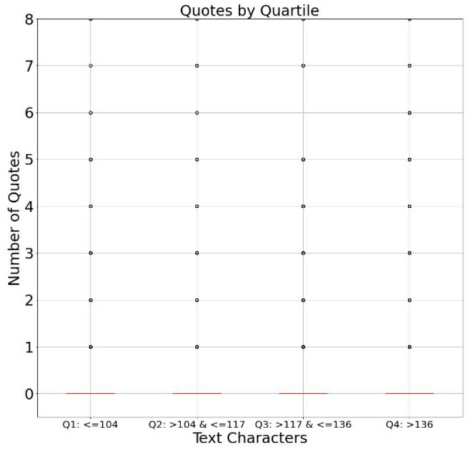
3) How do different multimodal elements in tweets impact viewer engagement and response rates? The multimodality-index is calculated during the ingestion phase. There are multiple ways to get a score from 2 to 6. The index adds up for any element present in the tweet [text, URL, hashtag (different than "#energyeast"), handle, emoji and photo]. Besides



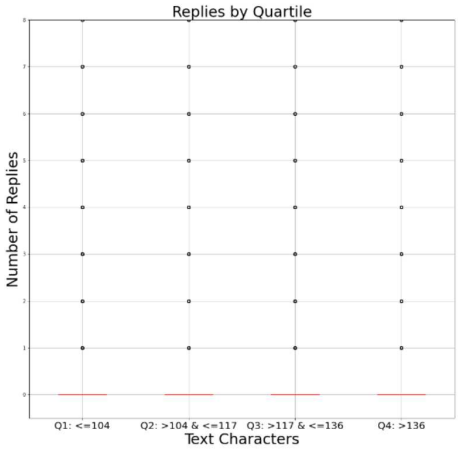
(a) Likes by Quartile



(b) Retweets by Quartile



(c) Quotes by Quartile



(d) Replies by Quartile

Figure 4.3: Tweet length versus viewer response

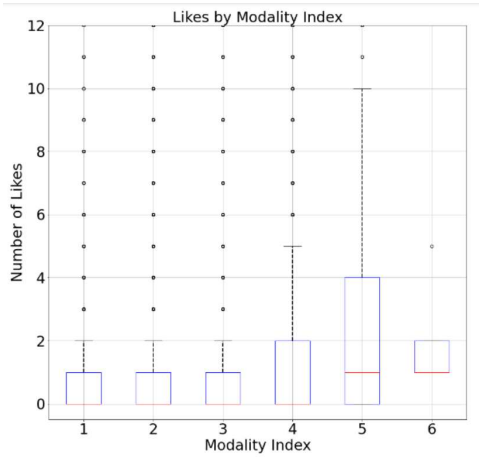
the count of multimodal elements, we also measure the presence of images — which is the single most influential multimodal element for engagement rate. To measure the impact of the multi-modality index on viewer response rates, we used a query similar to , simply replacing the "text_char_count" variable with the "multi-modality index" variable.

Figure 4.4 reports the relationships between tweet multimodality and viewers' response. As can be seen in the four boxplots, more multimodal tweets have distinctly higher responses (retweets, quotes, replies, likes) than text-only tweets. There were three predominant indexes: 2, 3, and 4, where 3 contained the highest number of tweets. This means that the presence of at least three elements was mostly used by tweeters. By inspecting the interactions, we observed that 'likes' and 'retweets' were the most predominant ones, likely because these responses are most easily accessible to users. This analysis supports our hypothesis that tweets with multimodal elements are more likely to capture viewers' attention, engagement, and response.

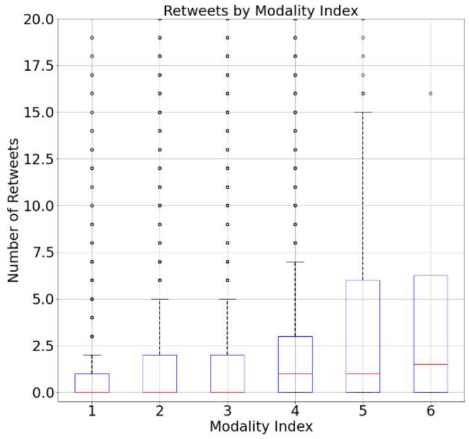
4) How do factual and emotional appeals in text and images influence tweet engagement? As we analyzed the tweets, instead of offering purely emotional or factual claims, we found that users frequently invoked claims that were factual and emotional, and utilized both text and images, resulting in multimodal hybridized claims. Thus, rather than create a 2 x 2 for factual x emotional and text x image, we have created a 3 x 3 to include factual + emotional texts and images. Figure 4.5 shows examples of our auto-labeling of tweets' factual versus emotional appeals in text in images.

In the top left corner of Figure 4.5 is a sample tweet with factual text and factual image. Fact-based text presented evidence as statistics and references to experts others and fact-based value systems (economics, science, law, energy security). Fact-based images gave complementary information such as maps, figures, and infographics.

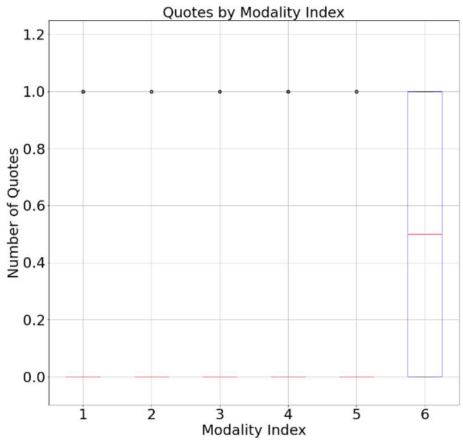
In the bottom right corner of Figure 4.5 is a sample tweet with emotional text and emotional image. Emotional text invoked group-level emotions like shame and love; individual-level emotions like frustration and anger; and val-



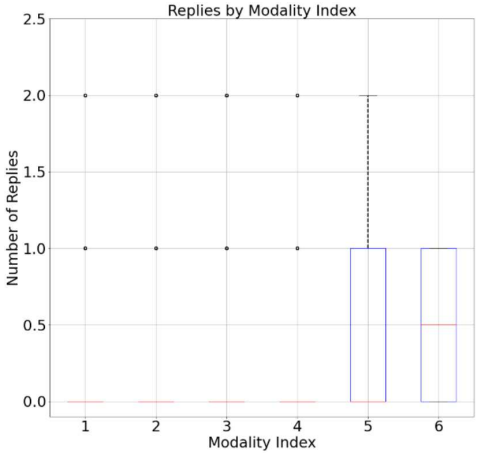
(a) Likes by Multimodality Index



(b) Retweets by Multimodality Index



(c) Quotes by Multimodality Index



(d) Replies by Multimodality Index

Figure 4.4: Tweet Multimodality versus viewer response

(a) Factual Text & Factual Image

(b) Factual+Emotional Text & Factual Image

(c) Emotional Text & Factual Image

(d) Factual Text & Factual+Emotional Image

(e) Factual+Emotional Text & Factual+Emotional Image

(f) Emotional Text & Factual+Emotional Image

(g) Factual Text & Emotional Image

(h) Factual+Emotional Text & Emotional Image

(i) Emotional Text & Emotional Image

Figure 4.5: Sample tweets for Tweets Text and Images (Factual, Emotion or Both)

ues of self-transcendence, benevolence, universalism, and hedonism. The emotional appeals were positive high-arousal emotions (excitement, attraction), negative high-arousal emotions (anger), and negative low-arousal emotions (frustration, fear). The emotional images focused on peoples' faces to reinforce the emotional appeals, and create emotional transference, and connection to moral domains. Images were funny, cute, and often brightly colored cartoons.

In the center of Figure 4.5 is a sample tweet with factual + emotional text and factual + emotional image. These hybridized claims supported irony (saying one thing while meaning another), hyperbole (over-exaggeration), and sarcasm (use of irony to mock) to create negatively-based humor. The focus of actors' ire was the hypocrisy of politicians, energy development policies, and regulatory review processes.

The remaining cells of Figure 4.5 give sample tweets that have varying degrees of hybridized claims. Tweets with factual text and emotional images gave numbers, statistics, and economic arguments in the text, supplemented with the faces of farmers, workers, and politicians. This script is flipped for tweets with emotional texts and factual images. The text was often a sarcastic criticism of factual imagery like corporate logos, maps, and machinery. Factual texts and factual + emotional images used a hybridized version of facts and statistics with imagery that emphasized politicians' hypocrisy, broken maps with spilling oil, and poster-like appeals to activists. Emotional text and factual + emotional images have emotional appeals and images that emphasize the hypocrisy of politicians, activists, and energy policies.

4.2 Immigration Conversations

This study conducted a comparative analysis of users with a different degree of popularity in a sample of the Harvard Dataverse Immigration and Travel Ban Tweets dataset [64]. The dataset contains tweets related to the "immigration and travel ban executive order announced by the Trump administration in January 2017" [52]. We used SoDa-TAP$_{v2}$ to investigate two questions.

1) Do tweets made by users with varying popularity exhibit different sentiments? We use the bucketing workflow to assess this difference. To ensure we are working with very dense buckets, we conduct this particular analysis on tweets made by users with less than 10,000 followers. We first apply a tweet filter to only keep the tweets where user_followers_count is less than 10,000. Then, we use the "uniform breaks" bucketing to create 8 different buckets with varying follower count ranges and use a boxplot to check the distribution of tweets within these buckets. Figure 4.6 shows the relevant query. 4.7 shows the resultant plot. As expected, the tweets for all buckets generally have a negative sentiment since the dataset is about a controversial topic. A preliminary observation from the plot might be that except for the 7th bucket, the median negative sentiment tends to get more negative as the user follower count increases. We can use an ANOVA to get further insights into these buckets' differences. To do this, we simply replace the Box Plot block in the query with an ANOVA block. Doing so, we get the following results: "F-Statistic = 3.34, p-Value = 0.00147". The results hint that there is a statistically significant difference between the means of the buckets, and the variation between the buckets is not due to random chance. Further analyses might be needed to confirm this.

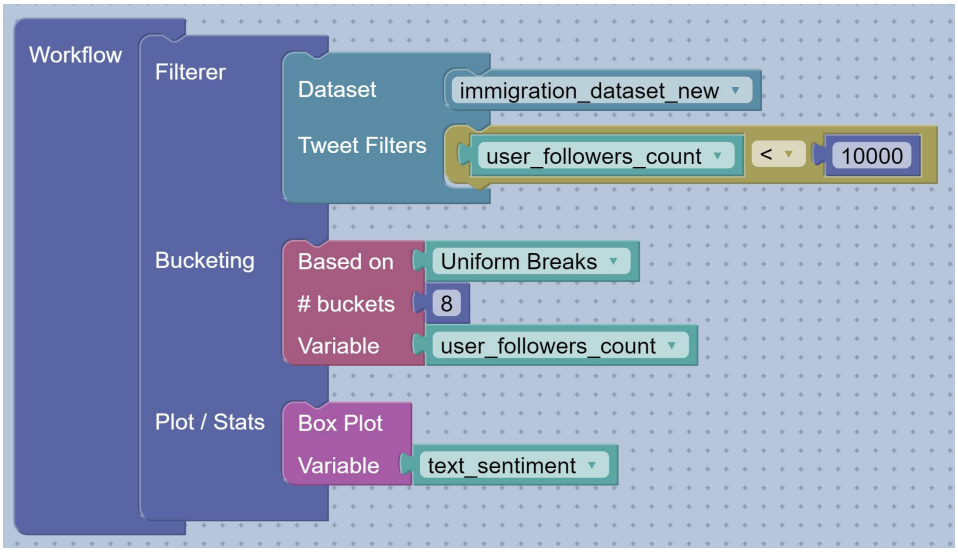


Figure 4.6: Query - Sentiment of Tweets made by Users with varying popularity

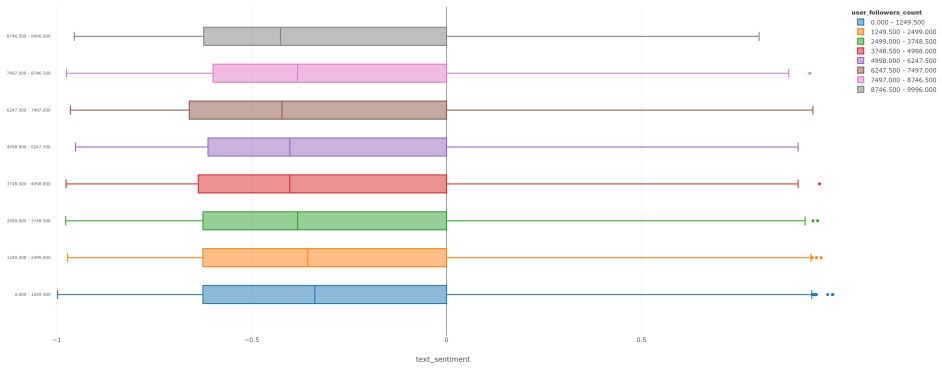


Figure 4.7: Plot - Sentiment of Tweets made by Users with varying popularity

Additionally, we can use a faceted box plot to observe whether this relation between users' follower count and text sentiment differs between verified and non-verified users. This type of faceted plotting is useful to observe the impact of a moderating variable on the relation between two other variables. Figure 4.8 shows how such a faceted box plot can be created and figure 4.9 shows the resultant plot. A possible observation might be that while verified users in the three least influential categories of users tend to have more negative tweets in general than their non-verified counterparts, this doesn't seem to always be the case as the follower count increases. Further analysis is needed to make any concrete claims. SoDa-TAP$_{v2}$ allows the creation of faceted scatter plots and faceted violin plots as well.

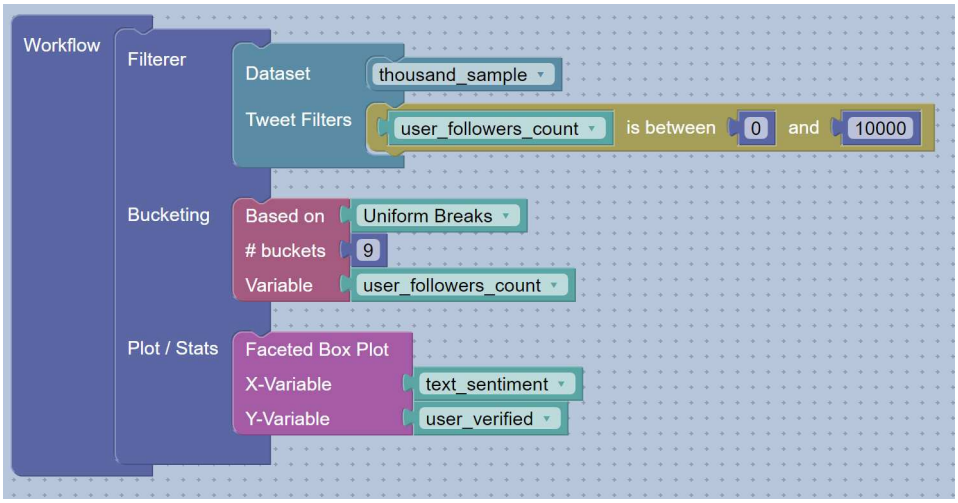


Figure 4.8: Query - Effect of a moderating variable

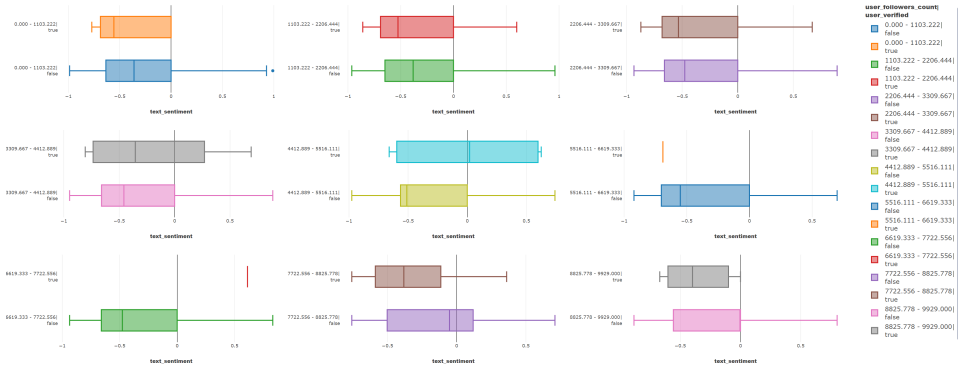


Figure 4.9: Plot - Effect of a moderating variable

(2) Is there a difference in the phrasing of tweets between users with varying popularity? Do the words used demonstrate different big-five personality traits? To assess this difference, we create 9 different data-subsets to generate treemaps for each of them. We use the create-data-subset workflow along with a bucket-picker using natural breaks, to create 9 different data-subsets one by one. We only consider tweets of users with less than 100000 followers to ensure dense buckets. The determined natural breaks are shown in Figure 4.10.

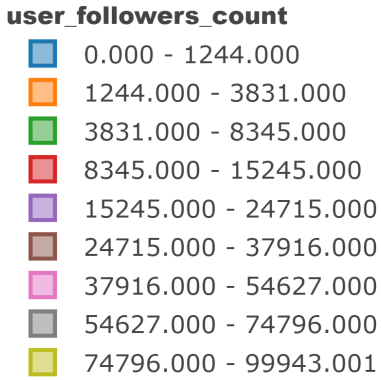


Figure 4.10: Followers data-subset ranges for immigration dataset

Figure 4.11 shows how we save the first bucket created with natural breaks as a new data-subset.

We run the same workflow 8 times changing the bucket number each time to create 9 different data-subsets based on natural breaks. We then filter each data-subset to retain a maximum of 10000 tweets in each data-subset to ensure

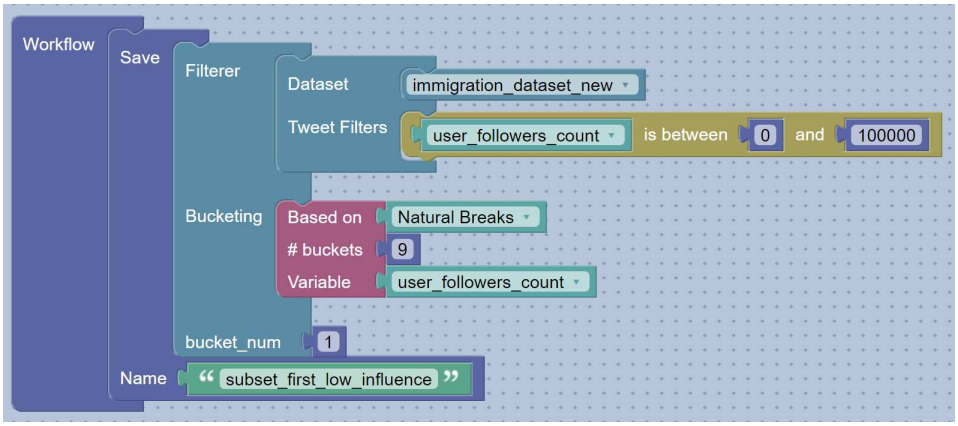
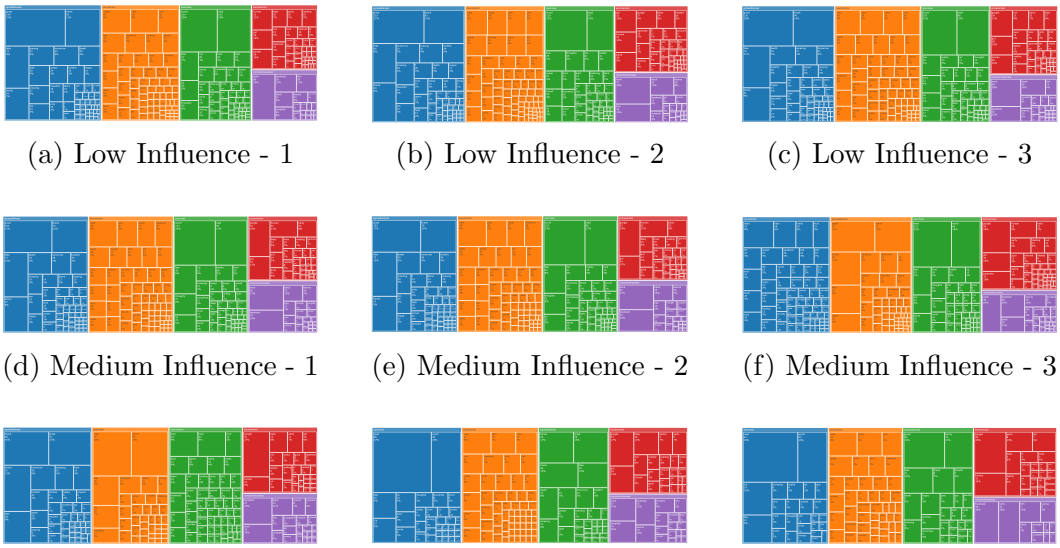


Figure 4.11: Create data-subset workflow with bucket-picker

consistency in density across data-subsets. We regard the first 3 sub-datasets as tweets by low-influence users, the second 3 as tweets by medium-influence users, and the last 3 as tweets by high-influence users. Once the data-subsets are created, we use the data preview tab to construct treemaps of big-five personality traits for each data-subset. Figure 4.12 shows a 3x3 grid created using the treemaps of big-five personality traits for the 9 data-subsets. They are placed in order of a growing number of followers.



(a) Low Influence - 1 (b) Low Influence - 2 (c) Low Influence - 3

(d) Medium Influence - 1 (e) Medium Influence - 2 (f) Medium Influence - 3

(g) High Influence - 1 (h) High Influence - 2 (i) High Influence - 3

Figure 4.12: Big-Five Personality Traits For Users with Varying Popularity

A quick observation looking at Figure 4.12 is that although the first five data-subsets (Low Influence-1 to Medium Influence-2) strictly exhibit the or-

der agreeableness -> neuroticism -> openness, this order tends to shift within users with higher influence (Medium Influence-3 to High Influence-3) who typically start exhibiting a higher amount of openness, with openness as the most dominant trait in the last 2 treemaps (High Influence-2 & High Influence-3). Therefore, it might be up for speculation that users with a higher influence tend to demonstrate more openness, while less influential users tend to be more agreeable.

Note: It is important to realize that we have made the above observation based on the change in the dominant order of personality traits, and not simply based on the frequency of the appearance of words in general. This is because the distribution of words across the treemap is largely influenced by the dictionary used, and one category (e.g. agreeableness) might have more common words than the other categories; therefore it might naturally appear higher up in the list. However, as we have observed, the shifting order in the frequently used words might indicate a difference in big-five personality traits across lower and higher influence users. Additionally, the purpose of this particular analysis is only to demonstrate how SoDa-TAP$_{v2}$ can be used to make preliminary observations using its preview features. As such, further observation is required to make any concrete claims.

4.3 Replication study

To evaluate the generality of SoDa-TAP$_{v2}$, we replicated some of the methodology of a research study. Our approach was not aimed at replicating or verifying the exact results of the study, since we are using a different dataset, but at demonstrating that our platform can efficiently handle the analyses commonly required in such studies. As such, we make no statements about the quality or accuracy of the research but simply demonstrate how the statistics and plots generated in it can conveniently be generated on our platform. We demonstrate each of the relevant queries on our VQL, and the resultant plots, highlighting the platform's ability to generate comparable visualizations quickly and conveniently. The original study we attempt to replicate the methodology

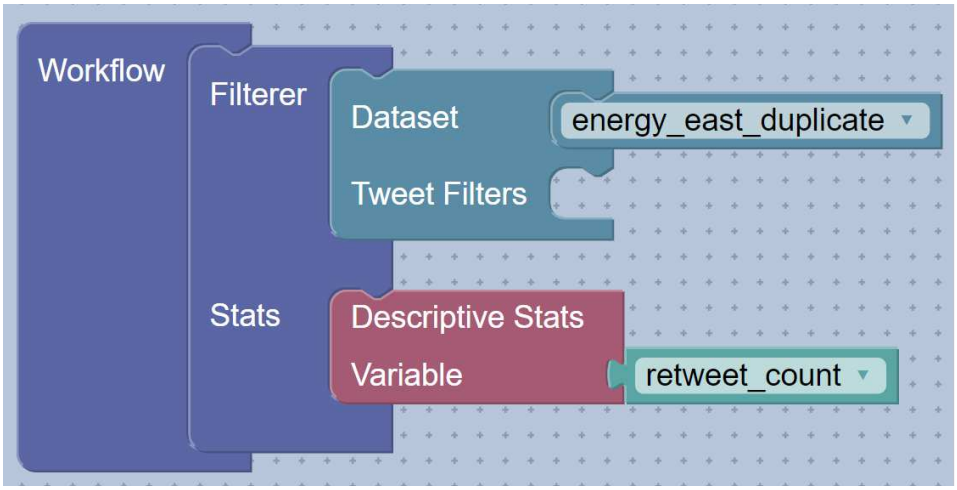


Figure 4.13: Examining descriptive statistics for a variable (Step 1)

of is a senior honors thesis work, titled "Understanding the Factors that Influence Tweet Popularity" [55], completed at the University of New Hampshire. We chose this study as it contains multiple visualizations and is straightforward enough for us to demonstrate some basic functionality of our platform. We will take a step-by-step approach, mentioning a step taken in the paper, and then demonstrating the equivalent methodology on our platform.

1) Step 1: The study lists the descriptive statistics for each of the variables used in their regression model, stating the mean, median, and standard deviation of each variable. Using SoDa-TAP$_{v2}$ we can use the statistics workflow, as shown in Figure 4.13, to generate the descriptive statistics for a variable, returning its min, max, average, sum, sum of squares, population variance, sampling variance, standard deviation, population standard deviation, sampling standard deviation, standard deviation bounds, skewness, and kurtosis.

2) Step 2: The study conducts regression analysis, using a single dependent variable and multiple independent variables. Using the SoDa-TAP$_{v2}$ statistics workflow supports the application of several statistical tests conveniently on the columns of a dataset. A similar analysis our system allows is an ANCOVA, as shown in Figure 4.14, returning the sum of squares

(quantifying how much of the variability in the dependent variable can be attributed to each independent variable), the degrees of freedom for each variable, the F-statistics, and the ($PR > F$) value (associating the p-value with the f-statistic) for each variable.

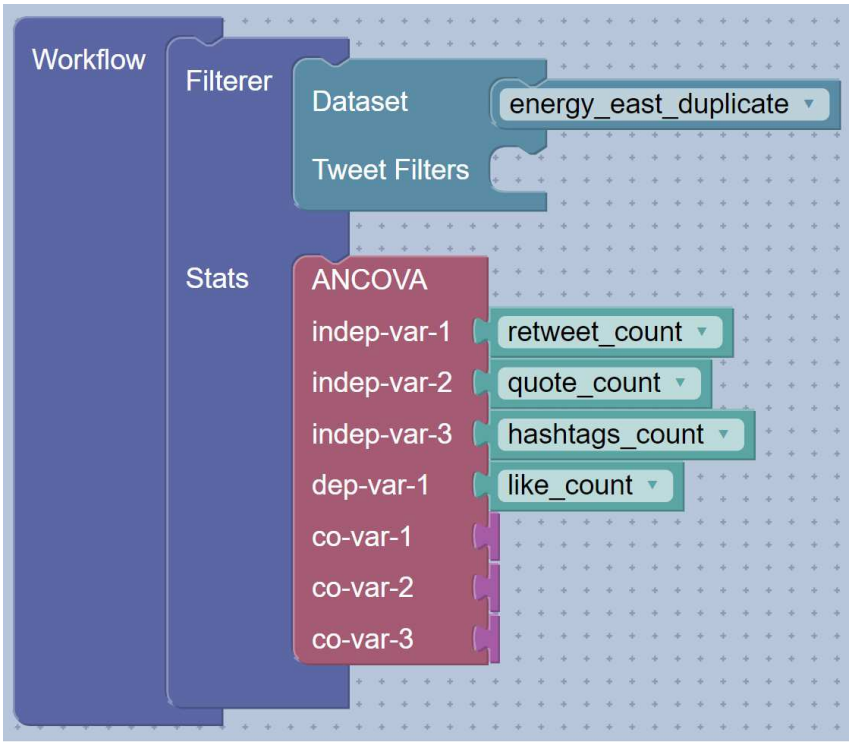
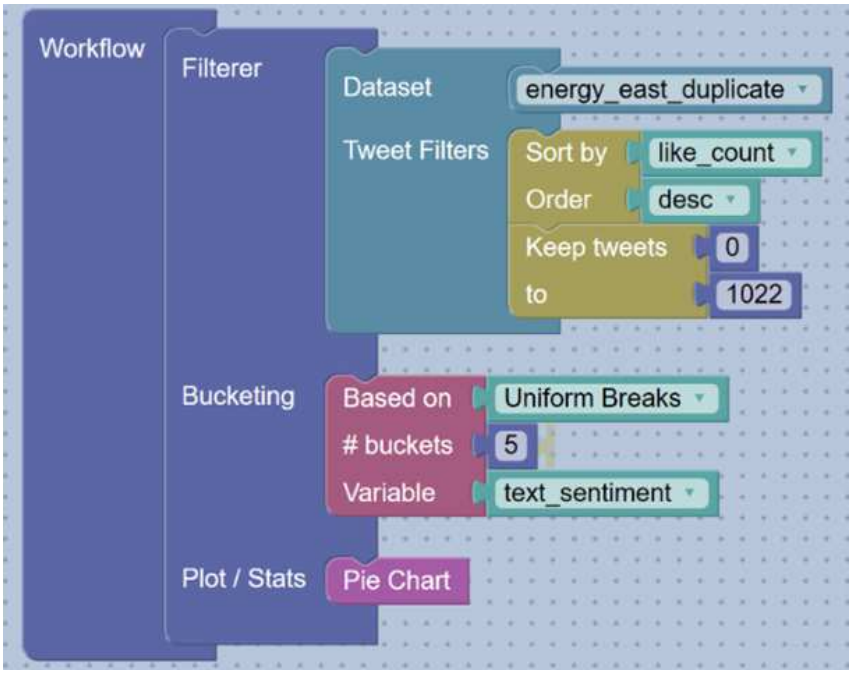


Figure 4.14: Statistical Analysis of the influence of three different parameters on the number of "likes" a post receives. (Step 2)

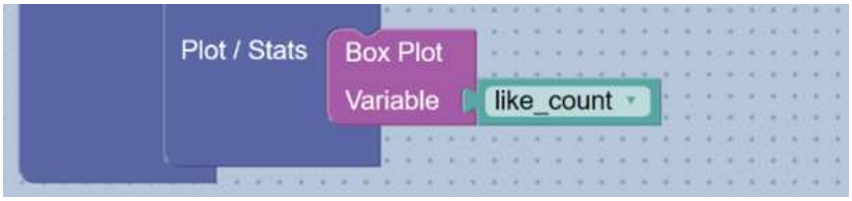
3) Step 3: The study then proceeds to conduct sentiment analysis on tweets using the MeaningCloud API. Our data pipeline conducts aspect-based sentiment analysis on tweets during the ingestion phase, using the VADER library, and appends the sentiment as a field to each document in the index, where the sentiment is a number within the range -1.0 (extremely negative) to 1.0 (extremely positive).

4) Step 4: The study demonstrates the distribution of the 1022 most impactful tweets, based on sentiment. Using the SoDa-TAP$_{v2}$ intra-dataset comparison block, we can sort the tweets based on their number of likes in descending order, keeping the first 1022 tweets, creating 5 uniform

buckets using the text sentiment field, and using a pie chart to demonstrate the distribution within each bucket, as demonstrated in Figure 4.15a. The resulting plot, compared to the plot from the study is shown in Figure 4.16.



(a)



(b)

Figure 4.15: Intra-dataset comparison using Bucketing with Uniform Breaks, Visualized with two different types of plots: (a) a pie chart and (b) a box plot. (Steps 4 and 5)

5) Step 5: The study demonstrates the average number of likes and retweets based on sentiment. Using SoDa-TAP$_{v2}$ we formulate this query similar to the previous one, simply replacing the pie chart with a boxplot using like_count as the input variable, as shown in Figure 4.15b. For the number of retweets, the input variable can simply be changed to retweet_count. The resulting plot, compared to the plot from the study, for average number of likes, is shown in Figure 4.17.

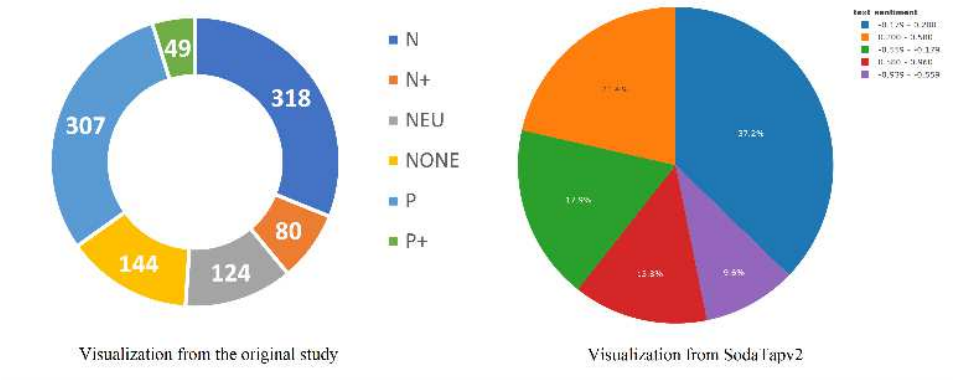


Figure 4.16: Replicating the original study pie chart. (Step 4)

6) Step 6: The study examines the distribution of tweets made by verified and non-verified users in the top 1022 tweets. Using the SoDa-TAP$_{v2}$ intra-dataset comparison block with unique/discrete bucketing, we can create buckets based on the author_verified variable, that can then be compared using a pie chart as shown in Figure 4.18. The resultant plot, compared with the plot from the original study is shown in Figure 4.19.

7) Step 7: The study calculates the correlation between the number of followers and the number of likes, and generates a scatter plot to visualize it. Using SoDa-TAP$_{v2}$, we can use the statistics workflow to calculate the correlation between the two variables as shown in Figure 4.20a. To generate the scatter plot, the intra-dataset comparison block can be used, using the no_bucketing block to specify that we won't be creating any, as shown in Figure 4.20b. The scatter plot from the original study, compared with the one generated in SoDa-TAP$_{v2}$ is shown in Figure 4.21.

4.4 VQL Performance Analysis

We evaluate the performance of the VQL by recording the time taken to carry out several operations. The analysis is conducted on a Linux machine with 16 cores and 56 GB of RAM.

The performance was tested on 5 different datasets with increasing sizes:

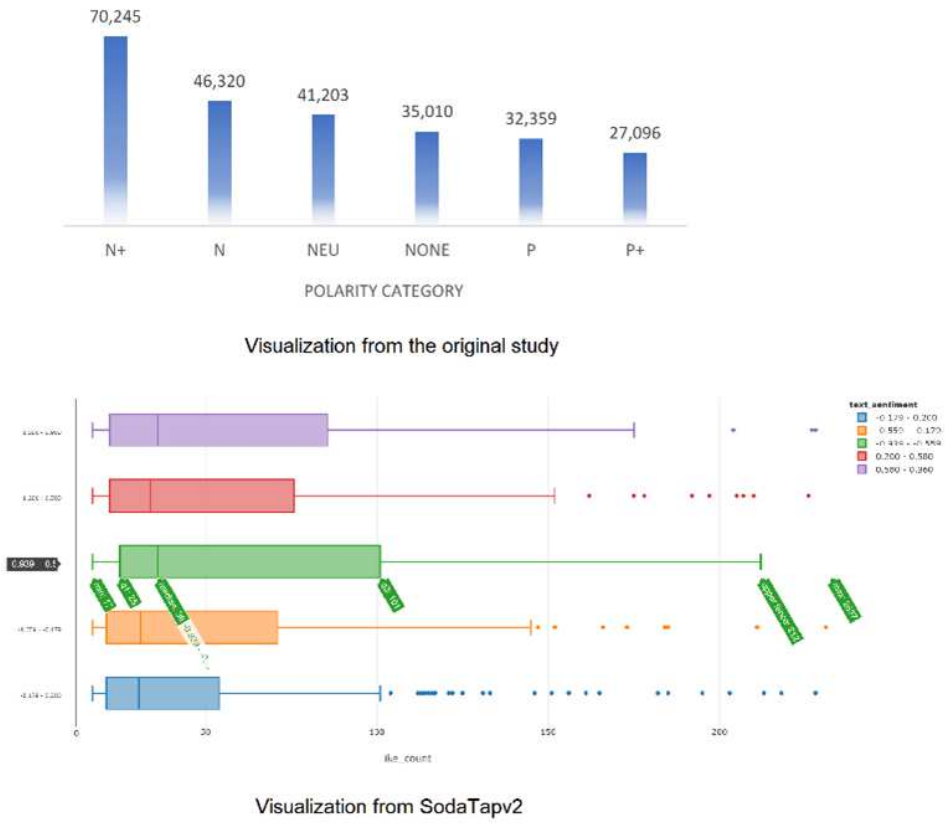


Figure 4.17: Plotting the average number of likes (Step 5)

- D1: A dataset containing energy-related tweets curated during the development of version 1, with approximately 300 documents.
- D2: A dataset containing Jason Kenney's Tweets, curated during the development of version 1, with approximately 10,000 documents.
- D3: A dataset containing tweets debating the energy-east pipeline, curated during the development of version 1, with approximately 150,000 documents.
- D4: A sample from "The Pushshift Telegram" dataset [11], containing approximately 350,000 documents.
- D5: A sample from the "Harvard Dataverse Immigration and Travel Ban Tweet Ids" dataset [64], containing approximately 1 Million documents.

It is important to note that:

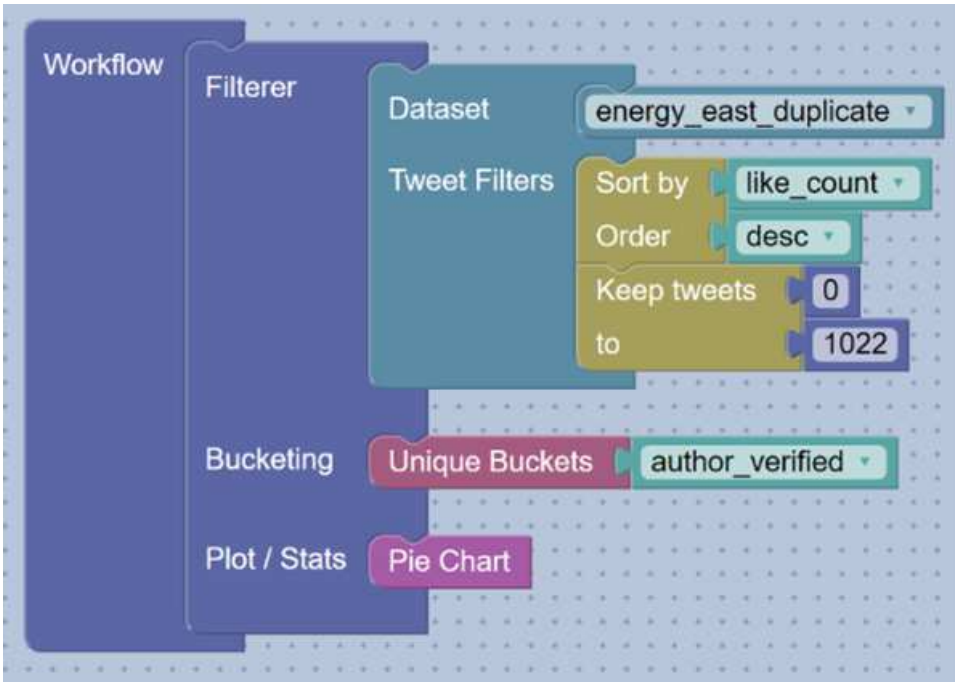


Figure 4.18: Examining verified and non-verified users (Step 6)

- Each dataset has different characteristics that can affect the execution times.
- The execution times are only for the first execution of the query. For subsequent executions, the results will be retrieved immediately from the cache.
- For bucketing using clustering, the manual version of K-Means should be used for bigger datasets. The automated k-means approach runs the k-means algorithm multiple times to determine the optimal number of clusters. As such, it is only suitable for small datasets. The HDB-SCAN implementation is expensive as well and is only suitable for small datasets.

The performance was tested for the following operations:

- OP1: retrieve 2 string fields for the entire dataset
- OP2: retrieve 2 string fields for the entire dataset (including sorting with numeric field)

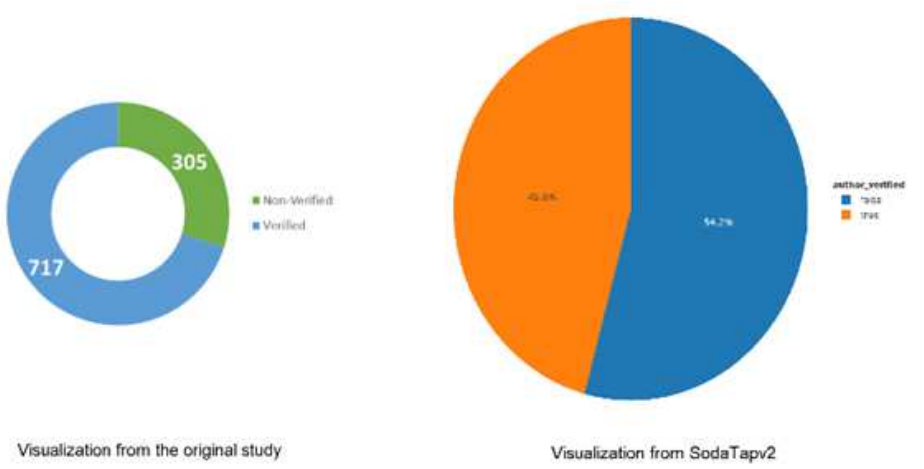
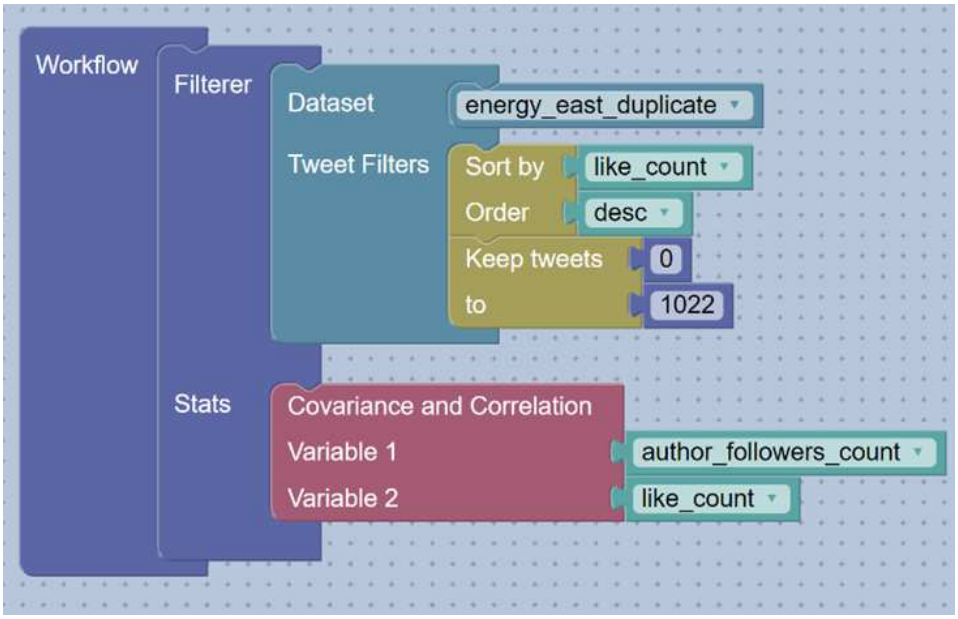
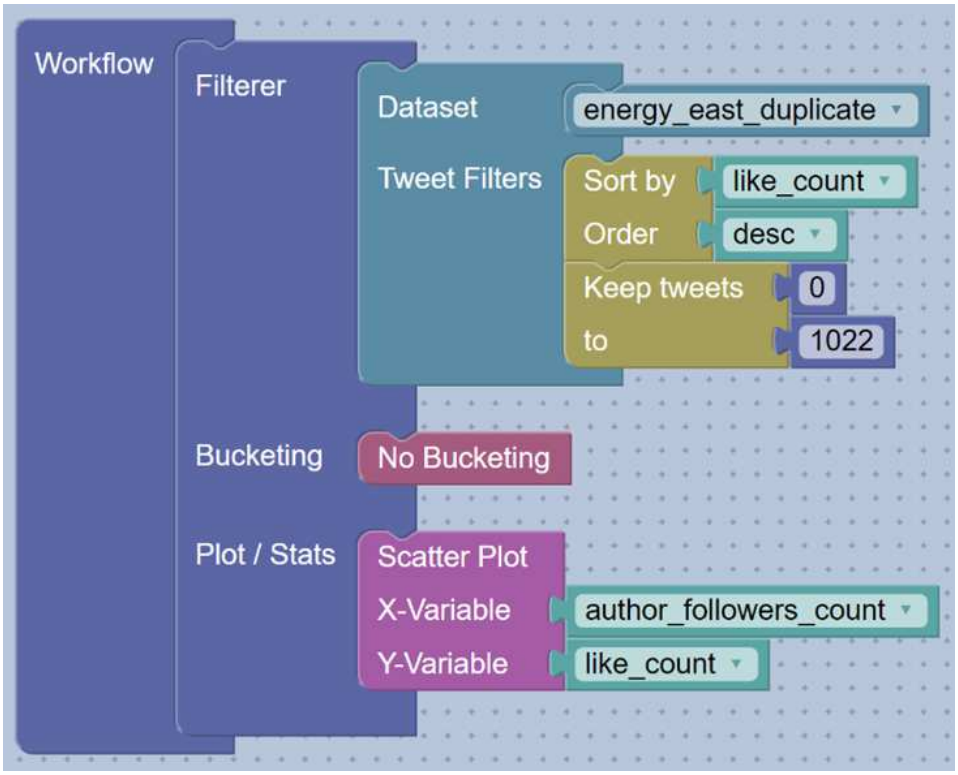


Figure 4.19: The numbers of verified and non-verified users (Step 6)

- OP3: retrieve 2 string fields with comparison filter (text_sentiment >0)
- OP4: position filtering
- OP5: retrieve 2 string fields with a tweet count filter (between 2 and 30 tweets)
- OP6: retrieve 2 string fields with author metric filter (avg text_sentiment >0)
- OP7: retrieve 2 string fields with a count comparison filter: count(text_sentiment >0) >count(text_sentiment <0)
- OP8: Bucketing using text content with K-Means (manual with 2 clusters) - including text vectorization
- OP9: Bucketing using text content with automated K-Means (multiple runs) - including text vectorization
- OP10: Bucketing using text content with HDBSCAN - including text vectorization
- OP11: Bucketing using Jenks Natural Breaks with 3 buckets
- OP12: Bucketing using Uniform Breaks using Natural breaks with 3 buckets



(a)



(b)

Figure 4.20: Followers and Likes (Step 7)

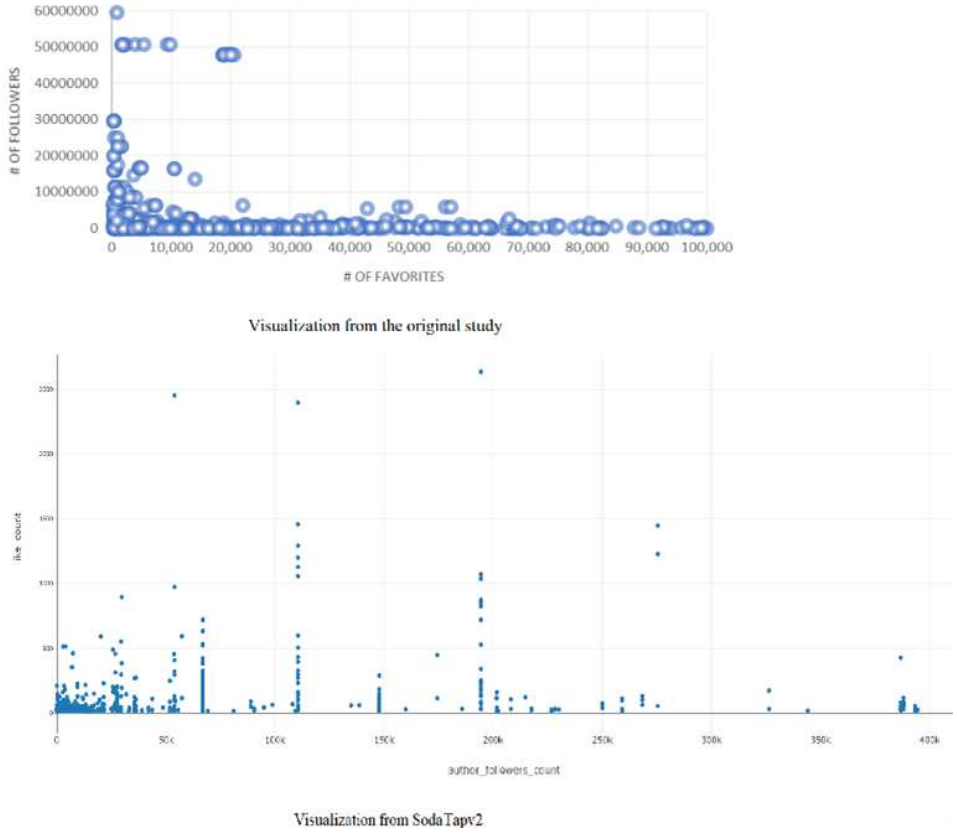


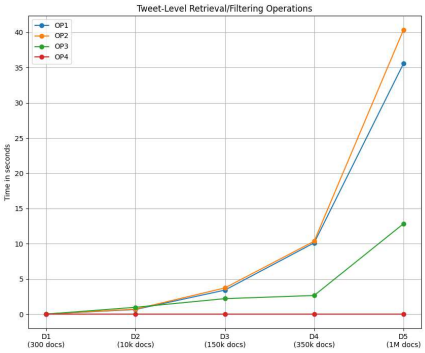
Figure 4.21: Scatterplot Comparison (Step 7)

- OP13: Bucketing using Quartile Breaks

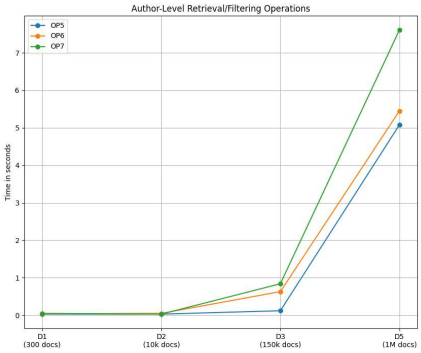
Table 4.2 shows the time in seconds for each of these operations. Author-level filtering operations could not be conducted on D4, as it doesn't contain an author_id field for aggregation. Figures 4.22a, 4.22b, 4.22c, and 4.22d show graphs of these operation times. Computationally inexpensive and expensive bucketing operations are separated into two separate graphs to avoid scaling issues.

Table 4.2: Dataset Operation Times

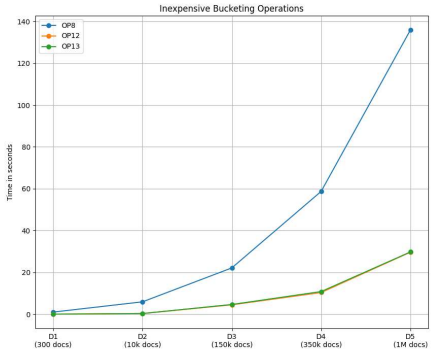
Operation vs Dataset	**D1**	**D2**	**D3**	**D4**	**D5**
OP1	0.01s	0.66s	3.41s	10.12s	35.57s
OP2	0.01s	0.71s	3.72s	10.35s	40.36s
OP3	0.004s	0.97s	2.20s	2.65s	12.83s
OP4	negligible	negligible	negligible	negligible	negligible
OP5	0.03s	0.03s	0.12s	not applicable	5.08
OP6	0.04s	0.05s	0.63s	not applicable	5.45s
OP7	0.05s	0.03s	0.84s	not applicable	7.61s
OP8	0.98s	5.90s	22.12s	58.71s	135.94s
OP9	3.31s	23.39s	841.35s	-	-
OP10	0.36s	21.40s	-	-	-
OP11	0.02s	0.86s	48.39s	275.67s	2472.98s
OP12	0.01s	0.31s	4.45s	10.34s	29.64s
OP13	0.01s	0.28s	4.63s	10.81s	29.76s



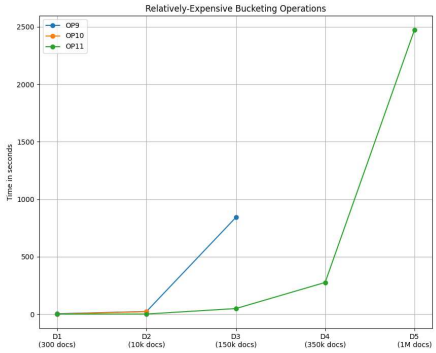
(a) Operation times for tweet-level retrieval/filtering operations



(b) Operation times for author-level retrieval/filtering operations



(c) Operation times for inexpensive bucketing oeprations



(d) Operation times for expensive bucketing operations

Figure 4.22: Execution time for various operations

Chapter 5

Conclusion

5.1 Contributions

This thesis and SoDa-TAP$_{v2}$ make the following contributions to the state of the art.

First, it provides a simple data-ingestion and analysis pipeline for social data. This pipeline extends the corresponding component of the first version of the tool in several important dimensions.

- A variety of data-labeling tools in the form of classifiers based on LLMs: SoDa-TAP$_{v2}$ incorporates multiple HuggingFace models, that enable automated categorization and tagging of social data. These models have been handpicked to allow classification across a wide range of categories, including toxicity, irony, sentiment, political bias, financial topics, etc.

- Image analysis with the Google Vision API: The multifaceted nature of social data necessitates that images accompanying the text can also be analyzed. This is particularly important as our energy-east case study 4.1 indicates that oftentimes, images have the most dominant impact on a reader's interaction with the post.

- Labeling with GPT 3.5 Turbo Instruct API: SoDa-TAP$_{v2}$ allows convenient interaction with the Instruct API to facilitate tasks like classification, removing the need to train an in-house model for small inexpensive use cases.

- Elasticsearch as the data store: The distributed storage and data manipulation capabilities of Elasticsearch, along with its rich Query DSL serve as the backbone of SoDa-TAP$_{v2}$'s filtering and analysis capabilities post-storage.

Second, it provides a visual query language through which domain experts with little to no programming knowledge can review their data and evaluate hypotheses of interest. Through our evaluation experiments, we have demonstrated that the SoDa-TAP$_{v2}$ VQL possesses several desirable attributes.

- It is easy to use, featuring an intuitive drag-and-drop interface based on the Blockly language editor. This visual programming paradigm often feels more familiar to users, lowering the barrier to data analysis for non-experts, as well as domain experts lacking experience with traditional programming languages who want to engage directly with their data.

- It is expressive, enabling users to filter and slice datasets to create distinct sample sets and save them for future use, perform aggregations, categorize data into buckets through classification, clustering, and natural breaks, and compare these buckets using statistical analyses and visualizations. Sections 4.1, 4.2, and 4.3, reporting two studies by our group as well as a replication of an extensive third-party study, document the language's expressiveness.

- SoDa-TAP$_{v2}$ scales well, as the performance analysis study in Section 4.4 demonstrates. The performance is evaluated on five datasets of varying sizes, containing 300, 10000, 150000, 350000, and 1 million documents, respectively. It exemplifies the system's ability to conduct a variety of operations efficiently on even the biggest dataset.

Finally, the system provides a simplified user interface for end-users to interact with all the system's aforementioned features from the comfort of a browser tab.

5.2 Future Work

Analyzing Images and Memes Contained in Tweets

SoDa-TAP$_{v2}$ currently relies on Google Vision API for some of its image analysis capabilities, which can prove to be expensive for large datasets, and doesn't encapsulate all the potential analysis that can be conducted on social media data. A console application offered by SoDa-TAP version 1 is also available but is extremely expensive computationally. Future work aims to introduce an efficient image analysis component to support a variety of interesting features.

Facial Expression Recognition for Sentiment Analysis We intend to train a facial expression recognition model to assess the sentiment conveyed by faces within images. This model will be trained on the FER-2013 dataset that categorizes facial expressions into seven categories: anger, disgust, fear, happy, sad, surprise, and neutral, within 32000 images.

Perception Analysis of Images To address the challenges of image context and viewer perception, we plan to implement a model like EmoNet. EmoNet is trained to recognize the emotion perceived by viewers from visual content. It categorizes perceptions into twenty emotions, including i) Craving ii) Sexual Desire iii) Romance iv) Disgust v) Entrancement vi) Interest vii) Aesthetic Appreciation viii) Horror ix) Empathic Pain x) Anxiety xi) Boredom xii) Confusion xiii) Adoration xiv) Surprise xv) Joy xvi) Fear xvii) Amusement xviii) Sadness ix) Awe and xx) Surprise. As images are difficult to categorize, and the context is often vague, such an analysis will provide better context into the 'intention' of the image, and what emotion it was supposed to induce, rather than simply the emotion displayed.

Contextual Understanding through Action Recognition and Scene Analysis We intend to use action recognition and scene understanding models to identify activities and settings within an image, to infer thematic elements and the overarching sentiment, e.g. "joy" in an image of children playing.

Utilization of Saliency Maps to Focus Image Analysis To aid the computational viability of the above analyses, we aim to employ saliency maps, to highlight regions within images most likely to attract viewer attention. Tools

such as Keras-vis support the creation of these maps, which can direct the analytical models to the most pertinent areas of an image. This technique will be particularly useful to reduce the area of the image that has to be analyzed, making the computationally expensive image analyses more viable within our tool.

Cloud Migration

In future iterations, we plan to migrate SoDa-TAP$_{v2}$ to a cloud platform like Google Cloud Platform (GCP). While our current setup with open-source tools has been effective, such setups can cause frequent development and management challenges, as we have experienced with SoDa-TAP$_{v2}$. The biggest problems we have faced are complex configurations, concerns like version conflicts and difficult interoperability of different tools, and maintenance overheads on development time. These issues are expected to become increasingly problematic as the system is extended further and as data processing demands grow. A managed data pipeline solution with a cloud provider instead offers the stability necessary to extend such a system with a small team like ours, without running into maintenance issues continuously. Using services like Google Kubernetes Engine (GKE) will help in the orchestration of Docker containers, allowing efficient resource management and automated scaling. For message queueing, Google Cloud Pub/Sub can replace Kafka, to provide a properly configured scalable messaging service, more efficient than a minimally configured local Kafka setup. A migration to a cloud platform like GCP will also naturally make it easier to use tools like Google Vision, as tools by a single cloud provider are configured for optimal interoperability. Lastly, the monitoring and management solutions that accompany these tools will provide comprehensive oversight and control over all components of the pipeline.

References

[1] *5 types of costly data waste and how to avoid them, cio*, https://www.cio.com/article/307487/5-types-of-costly-data-waste-and-how-to-avoid-them.html, (Accessed on 05/25/2024).

[2] J. Alwidian, S. Rahman, M. Gnaim, and F. Al-Taharwah, "Big data ingestion and preparation tools," *Modern Applied Science*, vol. 14, p. 12, Aug. 2020. DOI: 10.5539/mas.v14n9p12.

[3] *Apache kafka*, https://kafka.apache.org/, (Accessed on 05/27/2024).

[4] *Apache lucene*, https://lucene.apache.org/, (Accessed on 05/27/2024).

[5] *Apache spark*, https://spark.apache.org/, (Accessed on 05/27/2024).

[6] *Apache zookeeper*, https://zookeeper.apache.org/, (Accessed on 05/27/2024).

[7] *Audiense*, https://www.audiense.com/, (Accessed on 05/25/2024).

[8] *Auth0*, https://auth0.com/, (Accessed on 05/27/2024).

[9] *Avg aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-avg-aggregation.html, (Accessed on 05/27/2024).

[10] *Awario*, https://awario.com/, (Accessed on 05/25/2024).

[11] J. Baumgartner, S. Zannettou, M. Squire, and J. Blackburn, "The pushshift telegram dataset," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 14, pp. 840–847, May 2020. DOI: 10.1609/icwsm.v14i1.7348.

[12] E. Bjarnason, B. Gislason Bern, and L. Svedberg, "Inter-team communication in large-scale co-located software engineering: A case study," *Empirical Software Engineering*, vol. 27, Mar. 2022. DOI: 10.1007/s10664-021-10027-z.

[13] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," vol. 3, Jan. 2001, pp. 601–608.

[14] *Blockly - google for developers*, https://developers.google.com/blockly, (Accessed on 05/27/2024).

[15] *Brand24*, https://brand24.com/, (Accessed on 05/25/2024).

[16] *Bucket aggregations — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket.html, (Accessed on 05/27/2024).

[17] *Bucket selector aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline-bucket-selector-aggregation.html, (Accessed on 05/27/2024).

[18] *Buzzsumo*, https://buzzsumo.com/, (Accessed on 05/25/2024).

[19] *Cardinality aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-cardinality-aggregation.html, (Accessed on 05/27/2024).

[20] *Celery*, https://docs.celeryq.dev/en/stable/getting-started/introduction.html, (Accessed on 05/27/2024).

[21] *Cloud vision documentation*, https://cloud.google.com/vision/docs, (Accessed on 05/27/2024).

[22] *Communalytic*, https://communalytic.org/, (Accessed on 05/25/2024).

[23] *Confluent_kafka api*, https://docs.confluent.io/platform/current/clients/confluent-kafka-python/html/index.html#pythonclient-adminclient, (Accessed on 05/27/2024).

[24] *Countvectorizer — pyspark master documentation*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.CountVectorizer.html, (Accessed on 05/27/2024).

[25] *Cratedb*, https://cratedb.com/, (Accessed on 05/27/2024).

[26] *Crowdtangle*, https://www.crowdtangle.com/, (Accessed on 05/25/2024).

[27] *Datatables*, https://datatables.net/, (Accessed on 05/27/2024).

[28] *Discovertext*, https://discovertext.com/, (Accessed on 05/25/2024).

[29] *Docker*, https://www.docker.com/, (Accessed on 05/27/2024).

[30] A. Eckert and P. Juvonen, *Developing computational thinking with social robots in linguistically heterogeneous classrooms – the case of misty*, May 2024. DOI: 10.21203/rs.3.rs-4363387/v1.

[31] *Elasticsearch*, https://www.elastic.co/elasticsearch, (Accessed on 05/27/2024).

[32] *Elasticsearch dsl*, https://elasticsearch-dsl.readthedocs.io/en/latest/, (Accessed on 05/27/2024).

[33] *Elasticsearch filtered aliases - how to create, with examples*, https://opster.com/guides/elasticsearch/data-architecture/elasticsearch-filtered-aliases/, (Accessed on 05/26/2024).

[34] K. Elvira, O. Smyshliaeva, I. Panova, E. Neumoina, and A. Ponachugin, "Using visual-block programming environments to create robotic systems," in Mar. 2024, pp. 281–285, ISBN: 978-3-031-51271-1. DOI: 10.1007/978-3-031-51272-8_46.

[35] T. Engelthaler and T. Hills, "Humor norms for 4,997 english words," *Behavior Research Methods*, vol. 50, Jul. 2017. DOI: 10.3758/s13428-017-0930-6.

[36] *Extended stats bucket aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline-extended-stats-bucket-aggregation.html, (Accessed on 05/27/2024).

[37] *Filter aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-filter-aggregation.html, (Accessed on 05/27/2024).

[38] *Finiteautomata bertweet-base-sentiment-analysis - hugging face*, https://huggingface.co/finiteautomata/bertweet-base-sentiment-analysis, (Accessed on 05/27/2024).

[39] *Flask*, https://flask.palletsprojects.com/en/3.0.x/, (Accessed on 05/27/2024).

[40] *Flower*, https://flower.readthedocs.io/en/latest/, (Accessed on 05/27/2024).

[41] *Gensim - topic modelling for humans*, https://radimrehurek.com/gensim/index.html, (Accessed on 05/27/2024).

[42] *Github - elastic/elasticsearch-py: Official python client for elasticsearch*, https://github.com/elastic/elasticsearch-py, (Accessed on 05/27/2024).

[43] *Google code word2vec*, https://code.google.com/archive/p/word2vec/, (Accessed on 05/27/2024).

[44] *Google trends*, https://trends.google.com/trends/, (Accessed on 05/25/2024).

[45] A. Gruzd, P. Mai, and Z. Vahedi, "Studying anti-social behaviour on reddit with communalytic," Jun. 2020. DOI: 10.31124/advance.12453749.v1.

[46] C. A. G. Gutierrez, "Soda-tap: A data platform for social media analysis," Master's thesis, University of Alberta, Faculty of Graduate and Postdoctoral Studies, Fall, 2022. DOI: 10.7939/r3-v1t4-nb16. [Online]. Available: https://doi.org/10.7939/r3-v1t4-nb16.

[47] C. Gutierrez Gutierrez, A. Whittaker, K. Patenio, *et al.*, "Analyzing and visualizing twitter conversations," Dec. 2021.

[48] *Hdbscan, fast density based clustering, the how and the why - john healy - youtube*, https://www.youtube.com/watch?v=dGsxd67IFiU&ab_channel=PyData, (Accessed on 05/27/2024).

[49] *How to connect apache spark with elasicsearch - big data landscape - medium*, https://medium.com/@big_data_landscape/how-to-connect-apache-spark-with-elasicsearch-3f9d17eaacd9, (Accessed on 05/27/2024).

[50] M. Hu and B. Liu, "Mining and summarizing customer reviews," Aug. 2004, pp. 168–177. DOI: 10.1145/1014052.1014073.

[51] *Huggingface transformers*, https://huggingface.co/docs/transformers/en/index, (Accessed on 05/27/2024).

[52] *Immigration and travel ban tweet ids - gwu libraries dataverse*, https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/5CFLLJ, (Accessed on 05/27/2024).

[53] *Insights - audience intelligence*, https://www.audiense.com/products/audiense-insights, (Accessed on 05/26/2024).

[54] M. Irfan and J. George, "A systematic review of challenges, tools, and myths of big data ingestion," in Jan. 2022, pp. 481–494, ISBN: 978-981-19-2210-7. DOI: 10.1007/978-981-19-2211-4_43.

[55] K. A. Jalbert, "Understanding the factors that influence tweet popularity," Honors Theses and Capstones, University of New Hampshire, 2021. [Online]. Available: https://scholars.unh.edu/honors/588.

[56] *Jenkspy - pypi*, https://pypi.org/project/jenkspy/, (Accessed on 05/27/2024).

[57] *Kafka connect rest interface for confluent platform*, https://docs.confluent.io/platform/current/connect/references/restapi.html, (Accessed on 05/27/2024).

[58] *Keyhole*, https://keyhole.co/, (Accessed on 05/25/2024).

[59] *Kibana*, https://www.elastic.co/kibana, (Accessed on 05/27/2024).

[60] E. Klekovkin and A. Suntsov, "Application of visual programming for automation tasks in construction," *Construction and Geotechnics*, vol. 14, pp. 128–143, Jun. 2023. DOI: 10.15593/2224-9826/2023.2.10.

[61] *Lda — pyspark master documentation*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.clustering.LDA.html, (Accessed on 05/27/2024).

[62] H. Lefebvre, C. Legner, and M. Fadler, "Data democratization: Toward a deeper understanding," Sep. 2021.

[63] L. Libkin, "Expressive power of query languages," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 1081–1083, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1239. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1239.

[64] J. Littman, *Immigration and Travel Ban Tweet Ids*, version V1, 2018. DOI: 10.7910/DVN/5CFLLJ. [Online]. Available: https://doi.org/10.7910/DVN/5CFLLJ.

[65] K. Malhotra, A. Kalra, A. Kumar, M. Majmundar, G. Wander, and A. Bawa, "Understanding the digital impact of world hypertension day: Key takeaways," *European Heart Journal - Digital Health*, vol. 3, Aug. 2022. DOI: 10.1093/ehjdh/ztac039.

[66] A. Malik, A. Johri, R. Handa, H. Karbasian, and H. Purohit, "How social media supports hashtag activism through multivocality: A case study of ilooklikeanengineer," *First Monday*, vol. 23, Nov. 2018. DOI: 10.5210/fm.v23i11.9181.

[67] L. McInnes, J. Healy, and S. Astels, "Hdbscan: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, Mar. 2017. DOI: 10.21105/joss.00205.

[68] *Metrics aggregations — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics.html, (Accessed on 05/27/2024).

[69] V. Mirone, F. Di Bello, S. Morra, *et al.*, "Telemedicine and social media: A contemporary analysis of the most shared content by internet users," *Archivio Italiano di Urologia e Andrologia*, vol. 96, Apr. 2024. DOI: 10.4081/aiua.2024.11206.

[70] *Models - hugging face*, https://huggingface.co/models, (Accessed on 05/27/2024).

[71] *Models - openai api*, https://platform.openai.com/docs/models/gpt-3-5-turbo, (Accessed on 05/27/2024).

[72] *Modular applications with blueprints — flask documentation (3.0.x)*, https://flask.palletsprojects.com/en/3.0.x/blueprints/, (Accessed on 05/27/2024).

[73] M. Moens, L. Doorslaer, M. Billot, *et al.*, "Examining the type, quality, and content of web-based information for people with chronic pain interested in spinal cord stimulation: Social listening study," *Journal of medical Internet research*, vol. 26, e48599, Jan. 2024. DOI: 10.2196/48599.

[74] *Netlytic*, https://netlytic.org/index.php, (Accessed on 05/25/2024).

[75] *Numpy*, https://numpy.org/, (Accessed on 05/27/2024).

[76] J. Obiała, K. Obiała, M. Mańczak, J. Owoc, and R. Olszewski, "Covid-19 misinformation: Accuracy of articles about coronavirus prevention mostly shared on social media," *Health Policy and Technology*, vol. 10, Nov. 2020. DOI: 10.1016/j.hlpt.2020.10.007.

[77] *Pandas - python data analysis library*, https://pandas.pydata.org/, (Accessed on 05/27/2024).

[78] *Pipeline aggregations — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline.html, (Accessed on 05/27/2024).

[79] *Plotly express in python*, https://plotly.com/python/plotly-express/, (Accessed on 05/27/2024).

[80] *Plotly javascript graphing library*, https://plotly.com/javascript/, (Accessed on 05/27/2024).

[81] V. Ponizovskiy, M. Ardag, L. Grigoryan, R. Boyd, H. Dobewall, and P. Holtz, "Development and validation of the personal values dictionary: A theory-driven tool for investigating references to basic human values in text," *European Journal of Personality*, vol. 34, Aug. 2020. DOI: 10.1002/per.2294.

[82] *Pushdown*, https://docs.datavirtuality.com/v3/pushdown, (Accessed on 05/27/2024).

[83] *Pushdown vs pullup processing; why things are changing - by kirk haslbeck - collibradq - medium*, https://medium.com/owl-analytics/pushdown-vs-pullup-processing-why-things-are-changing-ae9ae5c2badc, (Accessed on 05/27/2024).

[84] *Pyppeteer - pypi*, https://pypi.org/project/pyppeteer/, (Accessed on 05/27/2024).

[85] *Range query — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-range-query.html, (Accessed on 05/27/2024).

[86] *Redis - the real-time data platform*, https://redis.io/, (Accessed on 05/27/2024).

[87] *Requests - pypi*, https://pypi.org/project/requests/, (Accessed on 05/27/2024).

[88] *Research in the sociology of organizations - emerald insight*, https://www.emerald.com/insight/publication/issn/0733-558X, (Accessed on 05/27/2024).

[89] D. Rohlinger, K. Rose, S. Warren, and S. Shulman, "Does the musk twitter takeover matter? political influencers, their arguments, and the quality of information they share," *Socius: Sociological Research for a Dynamic World*, vol. 9, p. 237 802 312 311 521, Feb. 2023. DOI: 10.1177/23780231231152193.

[90] A. Rovetta, "An integrated infoveillance approach using google trends and talkwalker: Listening to web concerns about covid-19 vaccines in italy," *Healthcare Analytics*, vol. 4, Oct. 2023. DOI: 10.1016/j.health.2023.100272.

[91] *Running spark on mesos - spark 2.4.7 documentation*, https://spark.apache.org/docs/2.4.7/running-on-mesos.html, (Accessed on 05/27/2024).

[92] *Running spark on yarn - spark 2.4.7 documentation*, https://spark.apache.org/docs/2.4.7/running-on-yarn.html, (Accessed on 05/27/2024).

[93] S. Santarossa, J. Lacasse, J. Larocque, and S. Woodruff, "#Orthorexia on instagram: A descriptive study exploring the online conversation and community using the netlytic software," *Eating and Weight Disorders - Studies on Anorexia, Bulimia and Obesity*, vol. 24, pp. 1–8, Apr. 2019. DOI: 10.1007/s40519-018-0594-y.

[94] *Schema registry overview*, https://docs.confluent.io/platform/current/schema-registry/index.html, (Accessed on 05/27/2024).

[95] *Scikit-learn*, https://scikit-learn.org/stable/, (Accessed on 05/27/2024).

[96] *Scratch - educators*, https://scratch.mit.edu/educators, (Accessed on 05/27/2024).

[97] *Scratch - imagine, program, share*, https://scratch.mit.edu/, (Accessed on 05/27/2024).

[98] *Scripted metric aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-scripted-metric-aggregation.html, (Accessed on 05/27/2024).

[99] *Socioviz*, https://socioviz.net/, (Accessed on 05/25/2024).

[100] *Spark standalone mode - spark 2.4.7 documentation*, https://spark.apache.org/docs/2.4.7/spark-standalone.html, (Accessed on 05/27/2024).

[101] *Sprout social*, https://sproutsocial.com/, (Accessed on 05/25/2024).

[102] J. Ge-Stadnyk and U. Gretzel, "The role of humour in driving customer engagement," in Jan. 2017, pp. 461–474, ISBN: 978-3-319-51167-2. DOI: 10.1007/978-3-319-51168-9_33.

[103] *Statsmodels*, https://www.statsmodels.org/stable/index.html, (Accessed on 05/27/2024).

[104] K. Stolpe and J. Hallström, "Visual programming as a tool for developing knowledge in stem subjects: A literature review," in Jan. 2024, pp. 130–169, ISBN: 978-90-04-68791-2. DOI: 10.1163/9789004687912_007.

[105] *Structured streaming + kafka integration guide*, https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html, (Accessed on 05/25/2024).

[106] *Submitting applications - spark 3.5.1 documentation*, https://spark.apache.org/docs/latest/submitting-applications.html, (Accessed on 05/27/2024).

[107] *Talkwalker*, https://www.talkwalker.com/, (Accessed on 05/25/2024).

[108] *Term query — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-term-query.html, (Accessed on 05/27/2024).

[109] *Term-level queries — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html, (Accessed on 05/27/2024).

[110] *Terms aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-terms-aggregation.html, (Accessed on 05/27/2024).

[111] *Terms query — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-terms-query.html, (Accessed on 05/27/2024).

[112] *The hdbscan clustering library*, https://hdbscan.readthedocs.io/en/latest/, (Accessed on 05/27/2024).

[113] *The healthcare hashtag project*, https://www.symplur.com/healthcare-hashtags/, (Accessed on 05/25/2024).

[114] *Top hits aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-top-hits-aggregation.html, (Accessed on 05/27/2024).

[115] *Twarc2*, https://twarc-project.readthedocs.io/en/latest/twarc2_en_us/, (Accessed on 05/27/2024).

[116] *Vadersentiment - pypi*, https://pypi.org/project/vaderSentiment/, (Accessed on 05/27/2024).

[117] *Variable width histogram aggregation — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-variablewidthhistogram-aggregation.html, (Accessed on 05/27/2024).

[118] *What is a push-down?* https://glossary.airbyte.com/term/push-down/, (Accessed on 05/27/2024).

[119] *What is wsgi?* https://wsgi.readthedocs.io/en/latest/what.html, (Accessed on 05/27/2024).

[120] Wikipedia contributors, *Backus–naur form — Wikipedia, the free encyclopedia*, [Online; accessed 27-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Backus%E2%80%93Naur_form&oldid=1225174908.

[121] Wikipedia contributors, *Declarative programming — Wikipedia, the free encyclopedia*, [Online; accessed 27-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Declarative_programming&oldid=1224915321.

[122] Wikipedia contributors, *Jenks natural breaks optimization — Wikipedia, the free encyclopedia*, [Online; accessed 27-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Jenks_natural_breaks_optimization&oldid=1205900617.

[123] *Wildcard query — elasticsearch guide [8.13] — elastic*, https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-wildcard-query.html, (Accessed on 05/27/2024).

[124] J. Womack and D. Jones, "Lean thinking: Banish waste and create wealth in your corporation," *Journal of the Operational Research Society*, vol. 48, Nov. 1997. DOI: 10.1057/palgrave.jors.2600967.

[125] A. Yoon and A. Copeland, "Toward community-inclusive data ecosystems: Challenges and opportunities of open data for community-based organizations," *Journal of the Association for Information Science and Technology*, vol. 71, no. 12, pp. 1439–1454, 2020. DOI: https://doi.org/10.1002/asi.24346. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.24346. [Online]. Available: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.24346.

Appendix A

The VQL BNF

<workflow>::= <inter-dataset-comparison> | <intra-dataset-comparison> | <data-subset-creation>|<statistics-workflow>

<inter-dataset-comparison>::= <multiple-filterers> <comparative-analysis>

<intra-dataset-comparison>::= <filterer> <bucketing> <comparative-analysis>

<data-subset-creation>::= <filterer> “new-dataset-name” | <bucket-picker> “new-dataset-name”

<statistics-workflow>::=<filterer><stat>

<multiple-filterers>::= <filterer> | <filterer> <filterer>

<comparative-analysis>::= <plot>|<stat>

<filterer>::= <tweet-filterer> | <author-filterer>

<bucketing>::= <continuous-bucketing> | <discrete-bucketing> | <clustering> | <no-bucketing>

<bucket-picker>::= <filterer><bucketing>“bucket-num”

<stat>::= “descriptive-stats” | “Covariance & Correlation” | “t-test” | “AN(C)OVA” | “MAN(C)OVA”

<plot>::= “pie-chart” | “box-plot” | “violin-plot” | “scatter-plot” | “faceted-box-plot” | “faceted-violin-plot” | “faceted-scatter-plot” | “line-plot”

<tweet-filterer>::= <dataset> <tweet-filters>

<author-filterer>::= <dataset> “pre-sample-size” <author-filters>

<continuous-bucketing>::= <continuous-bucketing-algorithm> | “num-buckets” | “numeric-variable”

<discrete-bucketing>::= “any-type-of-variable”

<clustering>::= <clustering-algorithm> “num-clusters (optional)”

<dataset>::= “any-index–or-alias”

<tweet-filters>::= <tweet-filter> | <tweet-filter> <tweet-filter>

<author-filters>::= <author-filter> | <author-filter> <author-filter>

<continuous-bucketing-algorithm>::= “Jenks-natural-breaks” | “breaks-with-uniform-range” | “breaks-into-quartiles”

<clustering-algorithm>::= “auto-K-Means” | “manual-K-means” | “HDBSCAN”

<tweet-filter>::= <range-filter> | <comparison-filter> | <sorting> | <position-filter>

<author-filter>::= <tweet-count-filter> | <author-metric-filter> | <count-comparison-

filter>

<range-filter>::= (“numeric-variable” | “date-variable”) (“number (from)” “number (to)” | “date (from)” “date (to)”)

<comparison-filter>::= “any-type-of-variable” <operator (overloaded)> “value-compatible-with-variable-type”

<sorting>::= <sorting-order> “any-type-of-variable”

<position-filter>::= “integer (from)” “integer (to)”

<tweet-count-filter>::= “integer (from)” “integer (to)”

<author-metric-filter>::= <aggregation-metric> “numeric-variable” <numeric-operator> “number”

<count-comparison-filter>::= “numeric-variable” <numeric-operator> “number” <numeric-operator> “numeric-variable” <numeric-operator> “number”

<operator>::= “contains” | “does-not-contain” | <numeric-operator>

<sorting-order>::= “asc” | “desc”

<aggregation-metric>::= “average” | “minimum” | “maximum” | “sum” | “median” | “median-absolute-deviation”

<numeric-operator>::= “<” | “>” | “≤ ” | “ ≥ ” | “=”