# VisDiscovery

*Visualization Recommendation Engine*

Authors:

1. Ali Rashad
2. Esraa Hefny
3. Nada Essam
4. Omar Khairy
5. Passant Ibrahim
6. Yehia Arafa

Supervisors:

- Dr. Mohamed Bassyouny
- Dr. Mohamed Saad

Computer and Communications Department
Faculty of engineering
Alexandria University

Graduation Project submitted in partial fulfillment of the B.Sc. Degree

July, 2017

# Acknowledgement

First, we would like to thank Allah for helping us to complete this project successfully.
We are heartily thankful to Dr. Mohamed Bassyouny, and Dr. Mohamed Saad, who not only served as our supervisors, but also guided, encouraged and challenged us throughout the project, allowing for the completion and success of this project. This project also would have never been tested without the help of BADR for Information Technology company, who provided us with a cleaned version of Wazzuf data-set.
And Last but not least, we owe our deepest gratitude and thankfulness to our families whom without their continual support; this project would have never been successful.

# Abstract

Data analysts often build visualizations as the first step in their analytical workflow. However, when working with high-dimensional datasets, identifying visualizations that show relevant or desired trends in data can be laborious and needs a lot of efforts.

Our graduation project introduces VisDiscovery, a visualization recommendation engine, that recommends interesting visualizations to facilitate fast visual analysis: given a subset of data to be studied, VisDiscovery intelligently explores the space of visualizations, evaluates promising visualizations for trends and recommends those it deems most useful or interesting, helping data scientists and analysts connecting things in interesting ways and look at data from different angles.

# TABLE OF CONTENTS

# LIST OF FIGURES

# List of Tables

CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

Data visualization is about conveying data as efficiently as possible, data scientists and analysts always use visualization techniques to explore patterns in their data and show their result effectively. Because of the way the human brain processes information, using charts or graphs to visualize large amounts of complex data is easier than poring over spreadsheets or reports.
Data visualization is a quick, easy way to convey concepts in a universal manner, and you can experiment with different scenarios by making slight adjustments. Given a new dataset or a new question about an existing dataset, an analyst builds various visualizations to get a feel for the data, to find anomalies and outliers, and to identify patterns that might merit further investigation. However, when working with high-dimensional datasets, identifying visualizations that show interesting variations and trends in data is non-trivial: the analyst must manually specify a large number of visualizations, explore relationships between various attributes and combinations, and examine different subsets of data before finally

arriving at visualizations that are interesting or insightful. This means that he needs to manually specify and examine every visualization that could possibly be done on his data.
Our engine, VisDiscovery, tackles the problem of automatically identifying and recommending visualizations which the data analyst will find it interesting for visual analysis. VisDiscovery intelligently explores the space of visualizations, evaluates promising visualizations for trends, and recommends those it deems most useful or interesting.
With the emerge of Big Data, data analysts want robust tools to visualize the data in meaningful ways that are interactive and can deal with these huge amounts of data. Our engine deals with huge amount of data and provide meaningful visualizations in an interactive time.

## 1.2 Scope of work

Recommending visualizations is the fact that whether a visualization is interesting or not depends on a host of factors. VisDiscovery adopt a simple criterion for judging whether the visualization is interesting or not: a visualization is likely to be interesting if it displays large deviations from some reference, that deviation can often guide users towards visualizations they find interesting.
Of course, there are other elements that may make a visualization interesting. For example, data-driven metrics that can reliably identify outliers, correlations, similarity between visualizations, the presence of clusters, etc. would lead to visualizations that may be interesting to users. In addition to data, many other aspects determine the interestingness of a visualization. We can identify attributes relevant to the task using metrics based on meta-data (e.g. schema) and semantics. Similarly, aesthetics play a big role in determining whether a visualization is easy to understand, making visual design important in visualization recommendation.

(a) Interesting visualization. (b) Un-interesting Visualization

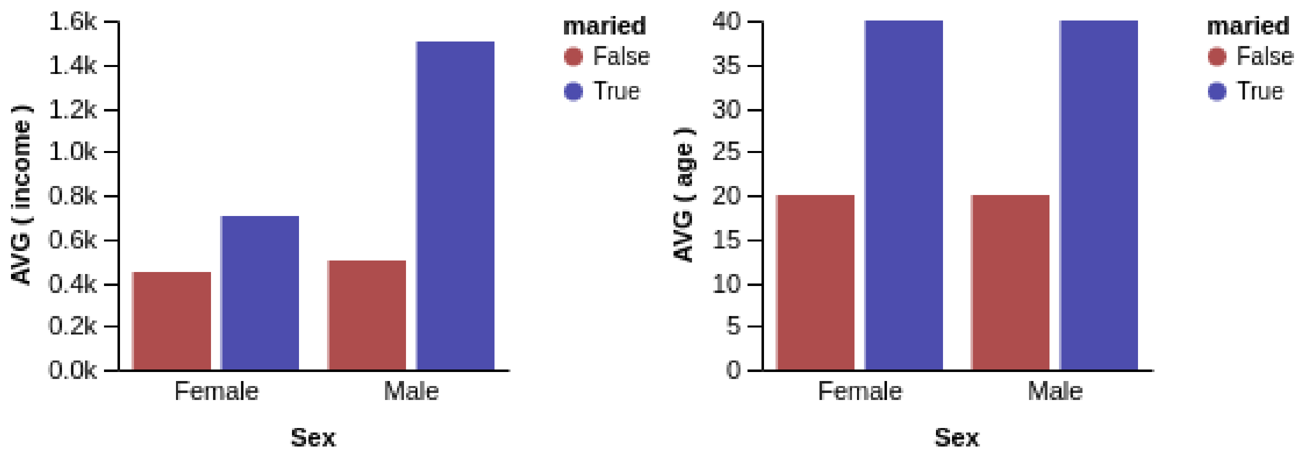


FIGURE 1.1. Motivation example

Finally, user preference, whether based on history of multiple users or a single user in a specific session, or based on explicit user input and domain knowledge, can provide insight into which visualizations are most interesting.
We used deviation as our utility metric as we mentioned before, to decide whether a certain visualization is interesting or not as it is the most proven metric that it would give satisfying results to the user.
For instance, If a data analyst wants to know how marital-status impacts socio-economic indicators like age and income, among others. He uses the US Census data [4] to conduct his analysis comparing unmarried US adults to married US adults. The analyst came with the two charts at **figure 1.1**. We know that for this particular task, analysts find the visualization in (a) interesting since it depicts a property of the data that is different for unmarried adults compared to married adults. Specifically, the chart depicts that although income for female and male unmarried adults is approximately equal, income for female, married adults is only half that of male, married adults. While (b) is a visualization that the analyst will not find interesting. Note that this chart depicts that age does not show different trends between unmarried and married adults.

What we mean by deviation bases utility metric that; visualizations which show trends in the target data (unmarried adults) that deviate from trends in reference data (married adults) are potentially interesting to analysts. Our project mainly works with big data which can be described in number of ways but actually big data means data-sets that are so large or complex that conventional data processing applications are not appropriate. From the top challenges which every professional data analysis faces is working with large data-set at the same time they want the system to be as interactive as possible.
VisDiscovery address and solves this dilemma. VisDiscovery operates on top of SQL-On-Hadoop system -as we will mention later in this report- working with huge-datasets and providing results in an interactive time.

## 1.3 Organization of the report

The report is organized into 7 chapters. In chapter 2, We will discuss background about data visualization techniques and related works. In chapter 3, the proposed algorithm of VisDiscovery along with the challenges and optimizations used will be illustrated. VisDiscovery engine design will be introduced in chapter 4. Implementation details will be explained in chapter 5. Chapter 6 will contain the case studies and results. Finally, chapter 7 will contain the conclusion and future work of VisDiscovery.

CHAPTER 2

# BACKGROUND & RELATED WORK

VisDiscovery draws on related work from multiple areas. In this section we will review systems in each of these areas, and describe how they relate to our project and what limitation they have.

## 2.1 Visualization tools

Businesses in every industry are under increasing pressure to mine their data and to spot trends, patterns, outliers, and even predict the future in order to maximize profits and beat competitors.
The visualization research community and industry has introduced a number of visualization toolkits and tools, including **Tableau**, **Spotfire**, **Grafana** and **Kibana** that enable users to build visualizations with different levels of sophistication. Some of these tools provide features that use heuristics to suggest chart types for a given set of variables, others uses elastic search to help the user to explore his data more.

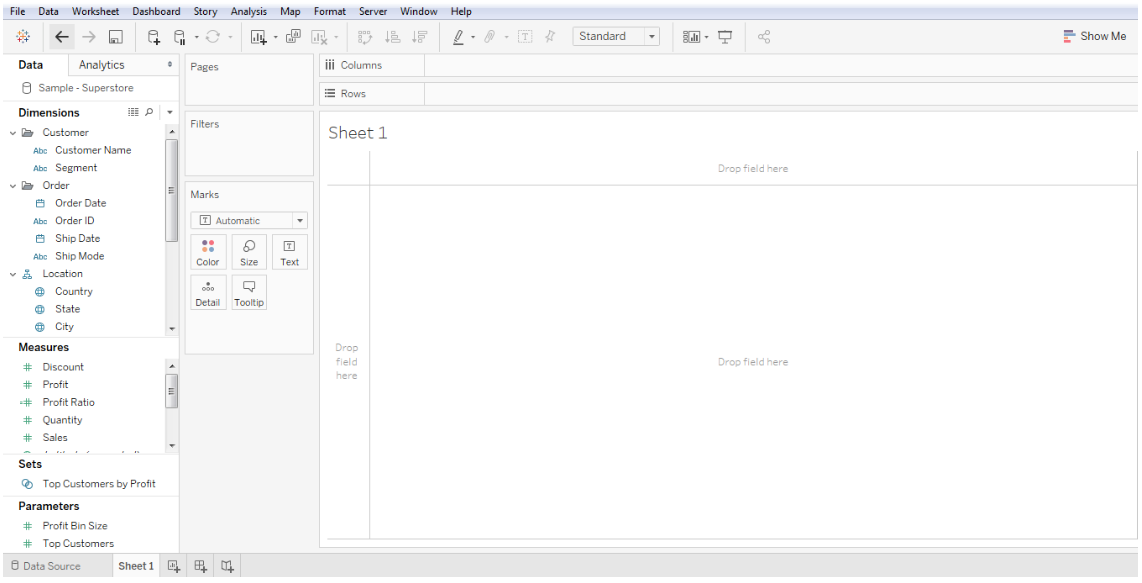


FIGURE 2.1. Tableau Desktop UI

**Limitations:** All of these tools, however, require the user to manually specify visualizations, leading to a tedious trial-and-error process to find visualizations that are interesting or useful. In contrast, VisDiscovery seeks to automatically identify and recommend interesting visualizations based on a utility metric.

## 2.1.1 Tableau

Tableau [3] [15] is an interactive data visualization tools that enables you to create interactive and apt visualizations in form of dashboards, worksheets to gain business insights for the better development of your company. It allows non-technical users to easily create customized dashboards that provide insight to a broad spectrum of information.
Tableau exposes all the different elements of a chart for the user that he can manipulate using his data to come up with a display that helps answer his questions. It also suggests a chart type according to the user data through some heuristics algorithms.

Before you analyze or visualize your data, you need to connect to your data. Tableau gives multiple options to connect to your data.
After the dataset is loaded, you can see the measures and dimensions associated with your data set on the data pane and start creating visualizations. **Figure 2.1** shows how a tableau desktop user interface looks like.

### 2.1.2 Spotfire

Spotfire [5] [28] ranks highly among users for its data visualization tools. It is popular because of its easy to use with dynamic dashboards and interactive visualization. Spotfire is available in Desktop, Cloud, and Platform editions. It also known for its Strong mobile access across a variety of platforms.
Spotfire is available in several different configurations to match the number of users you have and whether you want the software locally installed or hosted online.
The same as Tableau, Spotfire offers recommendation of chart type to ease it's usage to the users. **Table 2.1** offers some detailed differences between Spotfire and Tableau.

**Limitations:** Spotfire and Tableau is not an open source tool, they come in Professional or Personal uses which you have to buy a license from the company in order to use the system. Professional and Personal have different pricing and a few core differences, such as the number of data sources able to connect.

- **Tableau VS. Spotfire:**

| Area of comparison | Tableau | Spotfire |
|---|---|---|
| Pricing | Desktop edition, requires an upfront license | Desktop edition,offers annual subscription pricing, even though it's installed on-premise |
| Dashboards | lets users easily create interactive dashboards using custom filters and drag-and-drop functionality | It offers intuitive design dashboard and the ability to publish completed dashboards through a zero-footprint web client. Also Recommendations wizard can suggest best-practice visualizations based on the data you select |
| Reporting & Analytics Capabilities | Let's you connect to "messy" spreadsheets and fix/configure data while you sync. Pivots cross-tab data back into normalized columns, removes extraneous titles, text, and images, and reconciles metadata fields | It offer Content analytics: Decipher unstructured, text-based data from any source (e.g., CRM case notes, support tickets, weblogs, social media feeds, emails); mix unstructured content with other data to understand root cause and identify new opportunities. |
| Data Connectors | Offers a handful of native connectors that Spotfire does not: DataStax, EXASOL, Firebird, Google Analytics and BigQuery, Microsoft Azure, and SAP Sybase. | It can be connected to Apache Spark SQL, Cisco Information Server, and OData. |

TABLE 2.1. Tableau VS. Spotfire

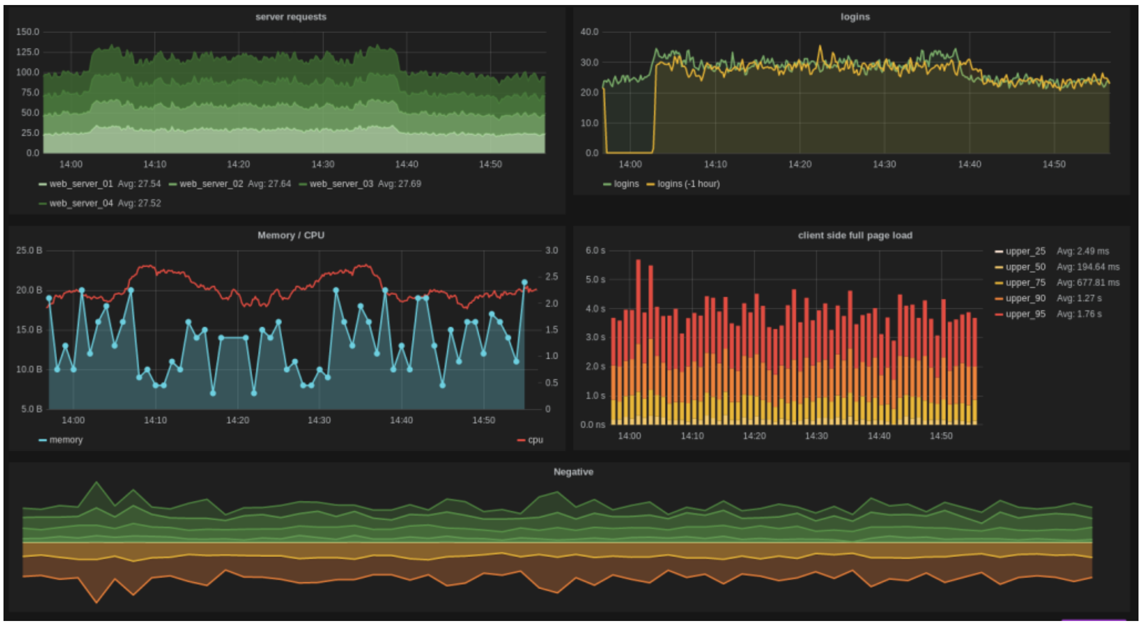


FIGURE 2.2. Grafana UI

### 2.1.3 Grafana

Grafana is an open source metric analytics & visualization suite. It is most commonly used for visualizing time series data for infrastructure and application analytics but many use it in other domains including industrial sensors, home automation, weather, and process control.
Essentially, it's a feature-rich replacement for Graphite-web – a Django based web application that renders graphs and dashboards, which helps users to easily create and edit dashboards. It contains a unique Graphite target parser that enables easy metric and function editing. Users can create comprehensive charts with smart axis formats (such as lines and points) as a result of Grafana's fast, client-side rendering even over long ranges of time that uses Flot – a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features- as a default option.
Grafana works with **Graphite**, **InfluxDB** [2], and **OpenTSDB** [23]. Newer versions can also work with other data sources such as **Elasticsearch**.

### 2.1.4 Kibana

Kibana [9] is a platform for analytics and visualization that allows you to explore, visualize, and build dashboards on top of the log data stored in **Elasticsearch** clusters. You can perform advanced data analysis and visualize your data in a variety of types of charts, tables, and maps.
Many modern IT organizations such as Netflix and Linkedin use the the popular **ELK Stack** (Elasticsearch, Logstash and Kibana) for log management. Using Kibana within this stack makes it simple to deploy and use visualizations. Once the ELK stack is installed, you will be able to access Kibana's powerful tools within the "Discover" section of the platform to explore and visualize your log data. Kibana's simple and easy-to-use dashboard can be used by anyone, even business users with minimal IT skills and knowledge.

- **Elasticsearch:**
  Elasticsearch [7] is a search engine based on Apache Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License. Official clients are available in Java, .NET (C#), Python, Groovy and many other languages. Elasticsearch is the most popular enterprise search engine followed by Apache Solr which is also based on Lucene. Elasticsearch can be used to search all kinds of documents. It provides scalable search, has near real-time search, and supports multitenancy. Elasticsearch is distributed, which means that indices can be divided into shards and each shard can have zero or more replicas. Each node hosts one or more shards, and acts as a coordinator to delegate operations to the correct shard(s). Re-balancing and routing are done automatically. Related data is often stored in the same index, which consists of one or more primary shards, and zero or more replica shards.

- **Kibana VS. Grafana:**

| Area of comparison | Kibana | Grafana |
|---|---|---|
| Logs & metrics | Runs on top of Elasticsearch and can create a comprehensive log analytics dashboard | Focuses on presenting time-series charts based on specific metrics such as CPU and I/O utilization. It does not allow for data search and exploring |
| Role-based access | By default, the Kibana dashboard is public. There are no built-in role-based access (RBA) controls. If you need to set up permission levels for multiple users, you will have to purchase Shield to add the additional configuration overhead | Grafana's built-in RBA allows you to maintain user and team access to dashboards. In addition, Grafana's rich API can be used for tasks such as saving a specific dashboard, creating users, and updating data sources. You can also create specific API keys and assign them to specific roles. |
| Integration of data sources | Kibana's native integration within the ELK Stack makes the setup fairly simple and user-friendly. | supports many different storage backends. For each data source, Grafana has a specific query editor that is customized for the features and capabilities that are included in that data source |

TABLE 2.2. Kibana VS. Grafana

## 2.2 Visualization Recommendation

There has been work which recommends visualization according to different aspects such as recommending visual encodings for different types of data [8] [13], Tools like VISO [18] use semantic ontologies to recommend visualizations, while those like Voyager 2 focus on the interface design aspects for exploratory data analysis. Another type of visualization recommendation is performed by systems like Profiler and Scorpion [11] [27] which detect specific patterns (e.g. outliers) in data. VizDeck [12] depicts all possible 2-D visualizations of a dataset on a dashboard, but this approach can quickly become intractable as the number of attributes increases. While SEEDB recommends visualizations according to some Utility metric.

- **Scalable Visualizations:** There has been some recent work on scalable visualizations that employ in-memory caching, sampling, and pre-fetching to improve the interactivity of visualization systems backed by databases (e.g., [19]). Such techniques could be employed in our setting to further improve response times (although some of these techniques, such as in-memory caching, can only work with small datasets).

- **Query Recommendation Systems:** There is related work [16] for recommending queries in databases. Such systems are designed to help users execute relevant queries over a database, typically by consulting historical query workloads and using statistical similarity or recommender algorithms to refine user inputs. These techniques focus on recommending SQL queries instead of visualizations (and hence don't focus on visually relevant utility metrics).

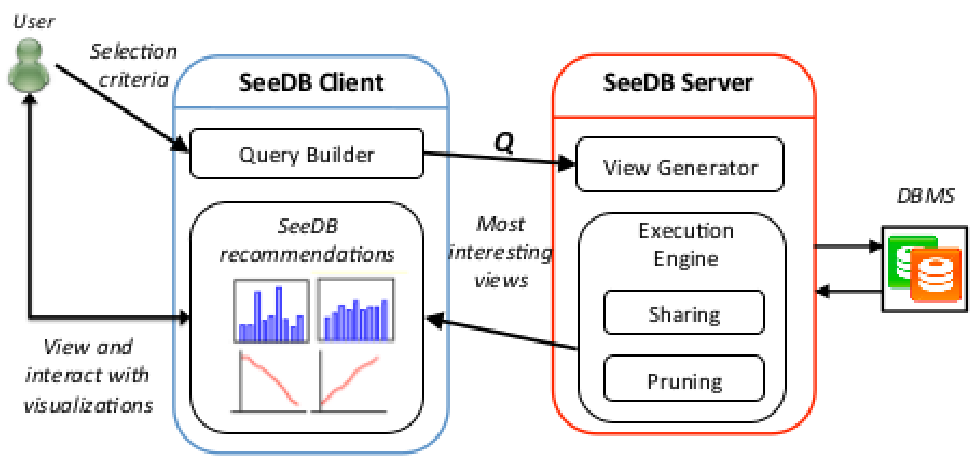


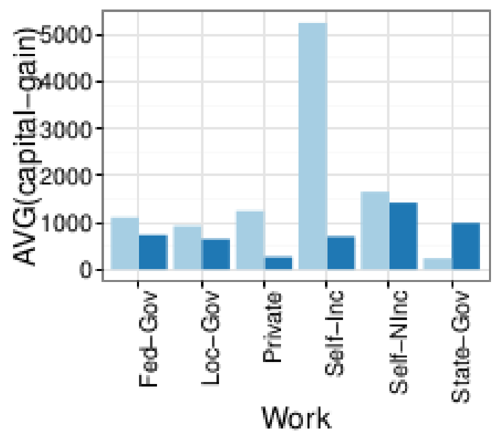
FIGURE 2.3. SEEDB Architecture

### 2.2.1 SEEDB

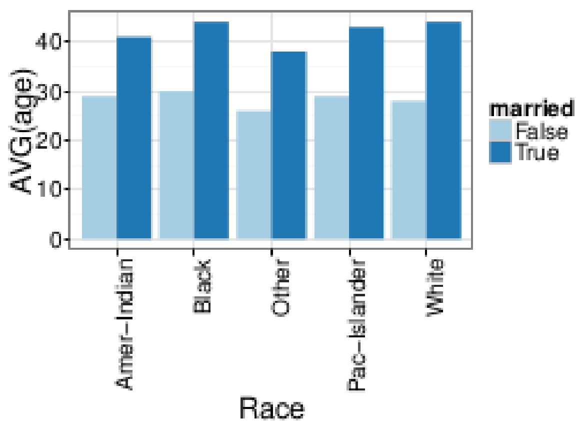
SEEDB [24] is a recommendation system which based its recommendation on deviation between two queries , target and reference queries, they used Earth Mover's Distance to compute the utility metric. It issues the two queries to any underlying DBMS.
SEEDB runs on top of a relational engine, and employs two types of optimization techniques, sharing based where they sharing computation by Combining target and reference queries and pruning-based where they do random dump sampling to save time in computation and avoid wasting computation on obviously low-utility visualizations, to obtain near-interactive performance and reduce the latency of the system.
**Figure 2.3** depicts the overall architecture of SEEDB. The SEEDB client is a web-based front-end that captures user input and renders visualizations produced by the SEEDB server. The SEEDB server is composed of two main components. The first component, the view generator, is responsible for parsing the input query, querying system metadata and generating the list of visualization queries that must be evaluated. The goal of the execution engine is to evaluate the collection of queries using our optimizations on top of the underlying DBMS.

The selected aggregate views (i.e., those with high deviation) are sent to the SEEDB client and are displayed as visualization recommendations to the user, who can then interact with these visualizations.
**Figure 2.4** validates deviation as a utility metric. they obtained ground truth data about interestingness of visualizations and evaluated SEEDB against it. They presented experts with the full set of potential aggregate visualizations and asked them to classify each visualization as interesting or not interesting in the context of the task. Of the 48 visualizations, on average, experts classified 4.5 visualizations (sd = 2.3) as being interesting for the task. The small number indicates that of the entire set of potential visualizations, only a small fraction (10%) shows interesting trends. To obtain consensus on ground truth, we labeled any visualization chosen by a majority of experts as interesting; the rest were labeled as not interesting. This process identified 6 interesting and 42 uninteresting visualizations.
**Figure 2.4-a** is a visualization recommended by SEEDB was labeled as interesting (". . . it shows a big difference in earning for self-inc adults"). In contrast, **Figure 2.4-b** was labeled as not interesting (notice the lack of deviation). While some classifications can be explained using deviation, some cannot: **Figure 2.4-c** showing high deviation was recommended by SEEDB, but was deemed uninteresting, while **Figure 2.4-d** showing small deviation was deemed interesting (". . . hours-per-week seems like a measure worth exploring").
They found that SEEDB recommendations have high quality and coverage, despite focusing on a simple deviation-based utility metric. For example, for k=3 ( number of recommendations specified by the user), all 3 visualizations recommended by SEEDB were labeled interesting. For k=5, four of the 5 recommended visualizations were labeled interesting. This indicates that the accuracy of SEEDB recommendations is very high.

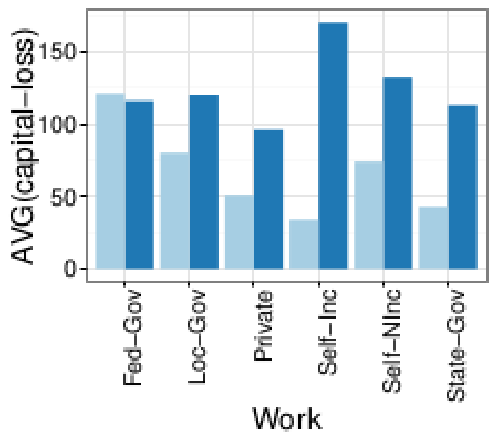
(a) High deviation, interesting.



(b) Low deviation, not interesting



(c) High deviation, not interesting



(d) Low deviation, interesting

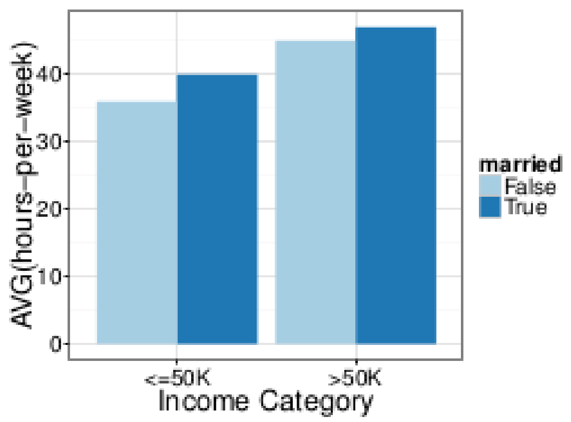


FIGURE 2.4. SEEDB deviation based ground truth

**Limitation:** One of the limitation of SEEDB is that their system only support bar-charts, another limitation is that they operate on structured data on a DBMS only unlike VisDiscovery who operates on both semi-structured data and convert it to structured data using Cloudera Impala as we will discuss later later.

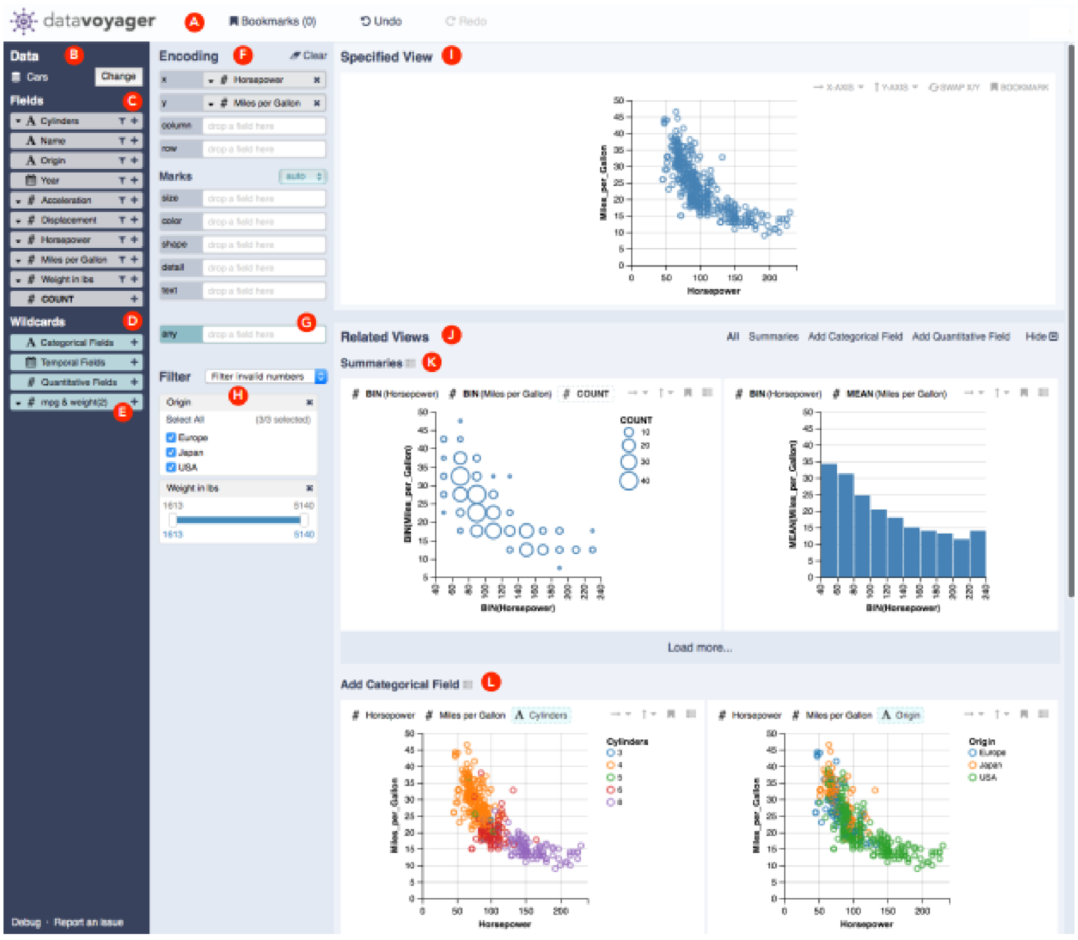


FIGURE 2.5. Voyager Interface

### 2.2.2 Voyager 2

Voyager 2 [26] is a visual analysis tool that combines manual and automatic chart specification in a single unified system. They introduce two partial specification interfaces: wildcards let users precisely vary the properties of a specification to generate multiple charts in parallel, while related views recommends visualizations relevant to the user's current focus. Both specifications and recommendations in Voyager 2 are represented using CompassQL a visualization query language based on Vega-Lite.

**Figure 2.5** shows the interface of Voyager 2 where the top panel (A) provides bookmark gallery and undo commands. The data panel (B) contains the dataset name, data fields (C), and wildcard fields (D). Wildcard fields let users create multiple views in parallel by serving as variables over an enumerated set of fields. Categorical, temporal, and quantitative field wildcards are provided by default, though users can manually author custom wildcards containing desired fields (E). The encoding panel (F) contains shelves for mapping fields to visual channels via dragand-drop, and a control for selecting mark type. Wildcard shelf (G) lets users add fields without selecting a specific channel, allowing the system to suggest appropriate encodings. The filter panel (H) shows dynamic query controls for filtering. The primary focus view (I) displays the currently specified chart. Related views (J) show recommended plots relevant to the focus view. Related summaries (K) suggest aggregate plots to summarize the data. Field suggestions (L) show the results of encoding one additional field within the focus view.

CHAPTER 3

# VisDiscovery overview

In this section we will discuss the proposed algorithm of VisDiscovery to recommend interesting visualization and what criterion we followed in proposing and implementing that algorithm.
We will also discuss the challenges we encountered in our project and how we managed to overcome them by using various optimizations to the system.

## 3.1 Problem Statement

In our work, we denote the attributes that we would like to group-by in our visualizations as dimension attributes, $A$, and the attributes that we would like to aggregate in our visualizations as measure attributes, $M$. Further, we denote by $F$ the set of potential aggregate functions over the measure attributes (e.g. COUNT,SUM, AVG). Assuming that the database we are operating on is $D$. For visualization purposes, we assume that we can group $D$ along any of the dimension attributes A and we can aggregate any of the measure attributes $M$. This leads to a two-column table that can be easily visualized via standard visualization mechanisms, such as

bar charts. In addition to the database $D$, we assume that the analyst has indicated a desire to explore a subset of data specified by a query $Q$.
The goal of VisDiscovery is to recommend visualizations of $Q$ that have high utility (which we measure using deviation, as explained in section 1.2). For instance, in our illustrative example in section 1.2, $Q$ can select any subset of records from the Census table. We denote the result of Q as $D_q$.
Each visualization can be translated into an aggregate over group-by query on the underlying data. We represent a visualization $V_i$ as a function represented by a triple $(a, m, f)$, where $m \in M$, $a \in A$, $f \in F$. We call this an aggregate view or simply a view. The aggregate view performs a group-by on and applies the aggregation function $f$ to measure attribute $m$. As an example, $V_i(D)$ represents the results of grouping the data in $D$ by $a$, and then aggregating the $m$ values using $f$, similarly $V_i(D_q)$ represents a visualization applied to the data in $D_q$.
VisDiscovery determines the utility of visualizations via deviation; visualizations that show different trends in the query dataset (i.e. $D_q$ ) compared to a reference dataset (called $D_r$ ) are said to have high utility. The reference dataset $D_r$ may be defined as the entire underlying dataset ($D$), the complement of $D_q$ ($D$ - $D_q$) or the data selected by any arbitrary query $Q$' ($D_q$'). The analyst has the option of specifying $D_r$. We used $D_r$ = $D$ as the default if the analyst does not specify a reference.
Given a view $V_i$, the deviation-based utility of $V_i$ is computed as the deviation between the results of applying $V_i$ to the query data ($D_q$), and applying $V_i$ to the reference data ($D_r$).
View $V_i$ which is applied to the query data can be expressed as query $Q_t$ below. We call this the **target view**.

$$Q_t = \text{SELECT } a, f(m) \text{ FROM } D_q \text{ GROUP BY } a$$

Similarly, view $V_i$ applied to the reference data can be expressed as $Q_R$. We call this the **reference view**.

$$Q_r = \text{SELECT } a, f(m) \text{ FROM } D_r \text{ GROUP BY } a$$

The two SQL queries corresponding to each view are referred to as **view queries**. The results of the above view queries are summaries with two columns, namely $a$ and $f(m)$. To ensure that all aggregate summaries have the same scale, we **normalize** each summary into a probability distribution.

Given an aggregate view $V_i$ and probability distributions for the **target view** $P[V_i\ (D_q)]$ and **reference view** $P[V_i\ (D_r)]$, we define the utility of $V_i$ as the distance between these two probability distributions. The higher the distance between the two distributions, the more likely the visualization is to be interesting and therefore higher the utility. Formally, if $S$ is a distance function

$$U(V_i) = S\ (P\ [V_i\ (D_q)], P\ [V_i\ (D_r)])$$

VisDiscovery supports variety of distance functions to compute utility including Euclidean and Manhattan distance. we found that they all give comparable results.

## 3.2 Challenges & Optimizations

In the previous illustrative example in section 1.2, the data analyst wanted to know how marital-status impacts socio-economic indicators like education and income, among others. For instance, he may build a chart showing average income as a function of marital status, visualize marital status as a function of education, plot the correlation with race and gender, visualize impact on hours worked per week, and so on. Depending on the types of visualizations created, the number of possible visualizations grows exponentially with the number of indicators in the dataset. As a result, creating and examining all possible visualizations quickly becomes intractable.

The two major obstacles we encountered:

- Operating on a very large and complex datasets.
- Responding with results in an interactive time scales.

To motivate the need for optimizations, we first describe how our execution engine would work without optimizations. To identify the *k* best aggregate views, VisDiscovery needs to do the following: for each aggregate view, it generates a SQL query corresponding to the **target** and **reference** view, and issues the two queries to the underlying database, Cloudera Impala in our case -as we will explain in the next chapter-. VisDiscovery repeats this process for each aggregate view. When the results are received, VisDiscovery computes the distance between the **target** and **reference** view distributions, and identifies the *k* visualizations with highest utility.
This basic implementation has many inefficiencies. In a table with a dimensions, *m* measures, and *f* aggregation functions, 2×f×a×m queries must be executed. As a result this could take >100s for large data sets. Such latencies are unacceptable for interactive use.

To reduce latency in evaluating the collection of aggregate views, VisDiscovery applies two kinds of optimizations: **Sharing-based Optimizations**, where aggregate view queries are combined to share computation in order to decrease the number of times we issue and fetch data from the database, and **Parallel SQL Execution** where we execute parallel SQL queries in the underlying database.

### 3.2.1 Sharing-based Optimizations

In the basic implementation, each visualization is translated into two view queries that get executed independently on the database. However, for a particular user input, the queries evaluated are very similar: they scan the same underlying data and differ only in the attributes used for grouping and aggregation. This presents opportunities to intelligently merge and batch queries, reducing the number of queries issued to the database and, in turn, minimizing the number of scans of the underlying data.

### 3.2.2 Parallel SQL Execution

Due to the need of executing many SQL queries on a very large data set and fetch the result in an interactive time we need to execute the queries as fast as possible that is why we used parallel query execution using Cloudera Impala as we will discuss in details in section 5.1

# Chapter 4

# System Design

System design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.
In this section we present the design of VisDiscovery in the form of various kinds of diagrams to ease the understanding of VisDiscovery goal.

## 4.1 Architecture

System architecture is a conceptual model that defines the structure, behavior, and more views of a system. It conveys the informational content of the elements comprising a system, the relationships among those elements, and the rules governing those relationships.
**Figure 4.1** shows the system architecture of VisDiscovery which can comprise the system components that work together to implement the overall system.

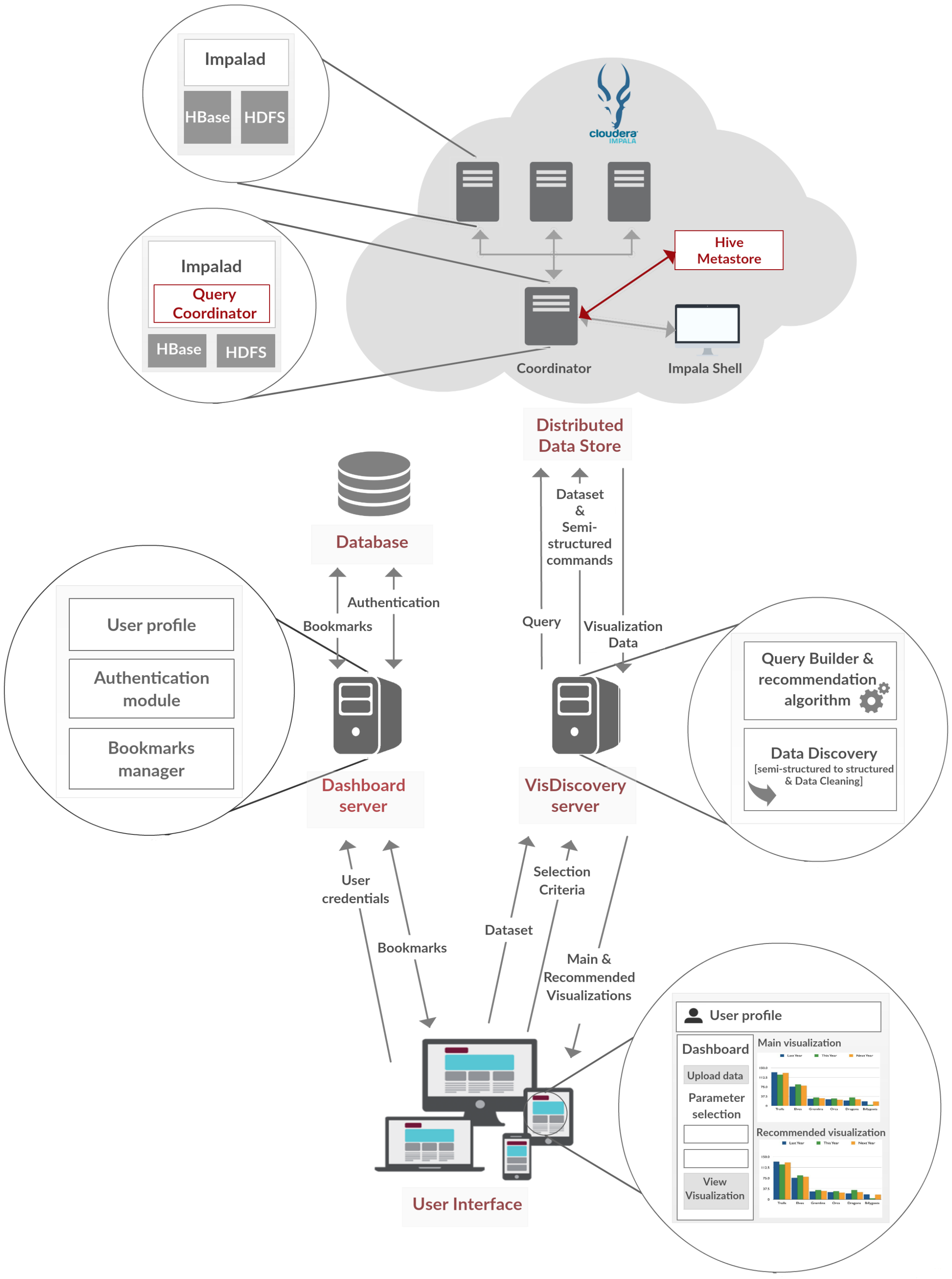


FIGURE 4.1. System Architecture

### 4.1.1 User Interface

This module is what our project mainly outputs, so we had to put a lot of effort, deciding what architecture to use and how we would implemented it. Through this module the user interacts with VisDiscovery through a user friendly interface which is easy, self-explanatory, efficient, and helpful to explore his data-set in the way which produces the desired result. The VisDiscovery front-end allows analysts to manually generate visualizations, or obtain data-driven recommendations on demand, which can be further refined using the manual interface. We designed the front-end as a mixed-initiative interface to keep the human in the loop and allow the user to steer the analysis.
This module was originally was embedded inside the VisDiscovery server, but this approach was confusing, and lacks the single responsibility rule which states that every module would just solely serve his one responsibility.
The Front-End architecture is mainly composed of three parts that capture user input and render visualizations produced by the VisDiscovery server:

- **User profile:** where the user can show his bookmarks and profile.
- **Dashboard:** containing the dataset selector which the user will operate on and the parameters selectors which is used to formulate queries.
- **Visualization display panel;** which displays the main(manual) and recommended visualizations.

### 4.1.2 VisDiscovery Server

This VisDiscovery Server is a middleware layer connecting components as Cloudera Impala with the user interface front-end which each others. It is composed mainly of two main modules:

- **Query builder and recommendation algorithm:** having some inputs from the user, VisDiscovery Server takes those inputs and runs our recommendation algorithm to build up queries, send it to Impala server to be executed and returns back the k best results to the frond-end.
- **Data discovery:** this module is mainly responsible to do two operations:
  1. Converting semi-structured data to structured, e.g., CSV into structured, e.g., Parquet which is one of the file formats **Cloudera Impala** deals with, we choosed the data-set to be stored in this particular file format for many reasons as explained in section 5.1.3.
  2. Data cleaning module, which cleans the data-set before VisDiscovery operates on it.

### 4.1.3 Dashboard Server

The Dashboard Server is used also as a middleware connecting the user interface front-end with an external database.
This module is mainly responsible for some actions as, the management of user profiles, saved visualizations, adding comments, sharing visualizations across users, and even adding multiple collaborators for the same data-set.

### 4.1.4 Distributed Data Store

To make the system scalable and operates over huge data sizes VisDiscovery uses **Cloudera Impala** to distribute the data and to execute parallel SQL queries. The implementation is explained in details in **section 5.1**. This module is responsible for storing the data and executing parallel queries. It is composed of the following components:

- **Clients;** entities including Hue, ODBC clients, JDBC clients, and the Impala Shell can all interact with Impala. These interfaces are typically used to issue queries or complete administrative tasks such as connecting to Impala.

- **Hive Metastore;** stores information about the data available to Impala. For example, the metastore lets Impala know what databases are available and what the structure of those databases is. As you create, drop, and alter schema objects, load data into tables, and so on through Impala SQL statements, the relevant metadata changes are automatically broadcast to all Impala nodes by the dedicated catalog service introduced in Impala 1.2.

- **Impalad;** this process, which runs on DataNodes, coordinates and executes queries. Each instance of Impala can receive, plan, and coordinate queries from Impala clients. Queries are distributed among Impala nodes, and these nodes then act as workers, executing parallel query fragments.

- **HBase and HDFS;** this is the storage for the data to be queried.

Queries executed using Impala are handled as follows:

1. User applications send SQL queries to Impala through ODBC or JDBC, which provide standardized querying interfaces. The user application may connect to any impalad in the cluster. This impalad becomes the coordinator for the query.

2. Impala parses the query and analyzes it to determine what tasks need to be performed by impalad instances across the cluster. Execution is planned for optimal efficiency.

3. Services such as HDFS and HBase are accessed by local impalad instances to provide data.

4. Each impalad returns data to the coordinating impalad, which sends these results to the client.

## 4.2 Entity Relation Diagram (ERD)

An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.

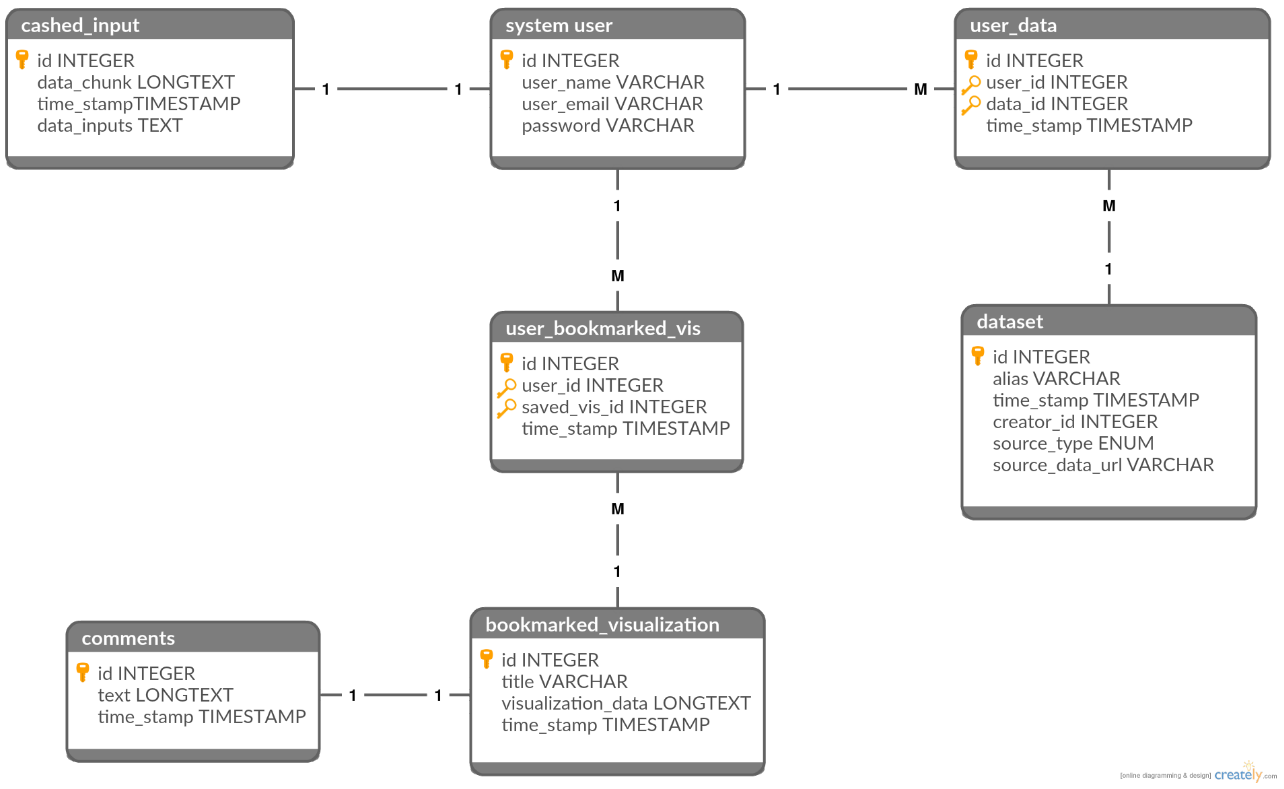


FIGURE 4.2. ERD Diagram

The ERD diagram consists of the following tables:

1. **System user**: this table contains user private personal information: and login credentials, which are required for authentication, these attributes contains:

    - **id:** primary key and unique for each user.
    - **user_name:** user first and last name.
    - **user_email:** for authentication and enabling interacting between users.
    - **password:** for authentication.
      **user_email** and **password** is responsible for authenticating the user to able to login in the system.

2. dataset: this table contains information which describes the dataset, it contains the following attributes:

    - **id:** primary key and unique for each data-set.
    - **alias:** this represents the data-set name, each table of data has its own name which represents the data-set itself.
    - **time_stamp:** represents the time of uploading this data-set to the system.
    - **creator_id:** this represents the id of the user who can modify and edit the data, not just accessing it.
    - **source_type:** describes the origin type of the data-set, in our system we accept dataset from 3 types: url, jdbc acess and csv file.
    - **source_data_url**: the actual data-set.

3. **user_data:** this table is to apply the relation between **system_user** table and **dataset** table it contains the following:
    - **id:** primary key which represent a unique value for each relation between dataset and system user.
    - **user_id:** foreign key which describes the **system_user** id.
    - **data_id:** foreign key which describes the **dataset** id.
    - **time_stamp:** this describes the time for applying the relation between the **system_user** and **dataset**.
4. **cashed_input:** this table contains the information for the latest visualizations and data inputs that the user entered and visualized the last time he/she used the system (this is save the time consumed by our system in computation, as the for the same inputs the same visualizations will appear as long as the data is not edited) , and it contains the following attributes:
    - **id:** primary key, a unique value for each cashed inputs.
    - **data_chunk:** this represents the last visualization data, visualization data are in json type.
    - **time_stamp:** time for the latest user access of the system.
    - **data_inputs:** the user inputs, inputs are selected from the dashboard.

5. **bookmarked_visualizations:** the information of the visualizations that the user bookmarked, it contains the following:
    - **id:** primary key and a unique value for each bookmarked visualization.
    - **title:** the title of the bookmarked visualization, user determines the title for each visualization when bookmarking it.
    - **visualization_data:** the data of the visualization which the user bookmarked, represented in json type.
    - **time_stamp:** the time of bookmarking the visualization.
6. **user_bookmarked_vis:** this table to apply the relation between the **bookmarked_visualization** table and **system_user** table, it contains the following:
    - **id:** primary key, the id for each relation between the bookmarked visualization and system user.
    - **user_id:** foreign key, the id of the user who can access this bookmarked visualization.
    - **saved_vis_id:** foreign key, the id of the bookmarked visualization.
    - **time_stamp:** time for applying the relation between the bookmarked visualization and system user.
7. **comments:** the information of the comments the user added on each visualization he/she bookmarked, it contains the following:
    - **id:** primary key, unique value for each block of comments.
    - **text:** the comment itself.
    - **time_stamp:** time of adding the comment.

Now we must describe the **relations** between the tables to understand the whole image of this part of the system, we have the following relations:

1. **relation between system_user and dataset:** between **system_user** table and **dataset** table we have a **user_data** table, this table applies the relation between the users and the data-sets. where each user can access and modify more than one dataset. And every dataset has only one creator (system_user) who can modify it and can have more than one users who can just access it.

2. **relation between cashed_input and system_user:** for each user there is only one cashed visualization data. And every cashed data are applied to only one user.

3. **relation between system_user and bookmarked_visualization:** between **system_user** table and **bookmarked_visualization** table there is **user_bookmarked_vis** table which applies the relation between the users and the bookmarked visualizations. Where each user can bookmark more than one visualization. And every visualization can be accessed by more than one user but only one user is able to bookmark it. User can share bookmarked visualization to be accessed by other users.

4. **relation between comments and bookmarked_visualization:** each visualization has its own comments. And every comments are applied to only one bookmarked visualization.

## 4.3 Unified Modeling Language (UML)

The Unified Modeling Language (UML) was created to forge a common, semantically and syntactically rich visual modeling language for the architecture, design, and implementation of complex software systems both structurally and behaviorally.

### 4.3.1 Use case diagram

A use case diagram is a graphic depiction of the interactions among the elements of a system. It is a methodology used in system analysis to identify, clarify, and organize system requirements.

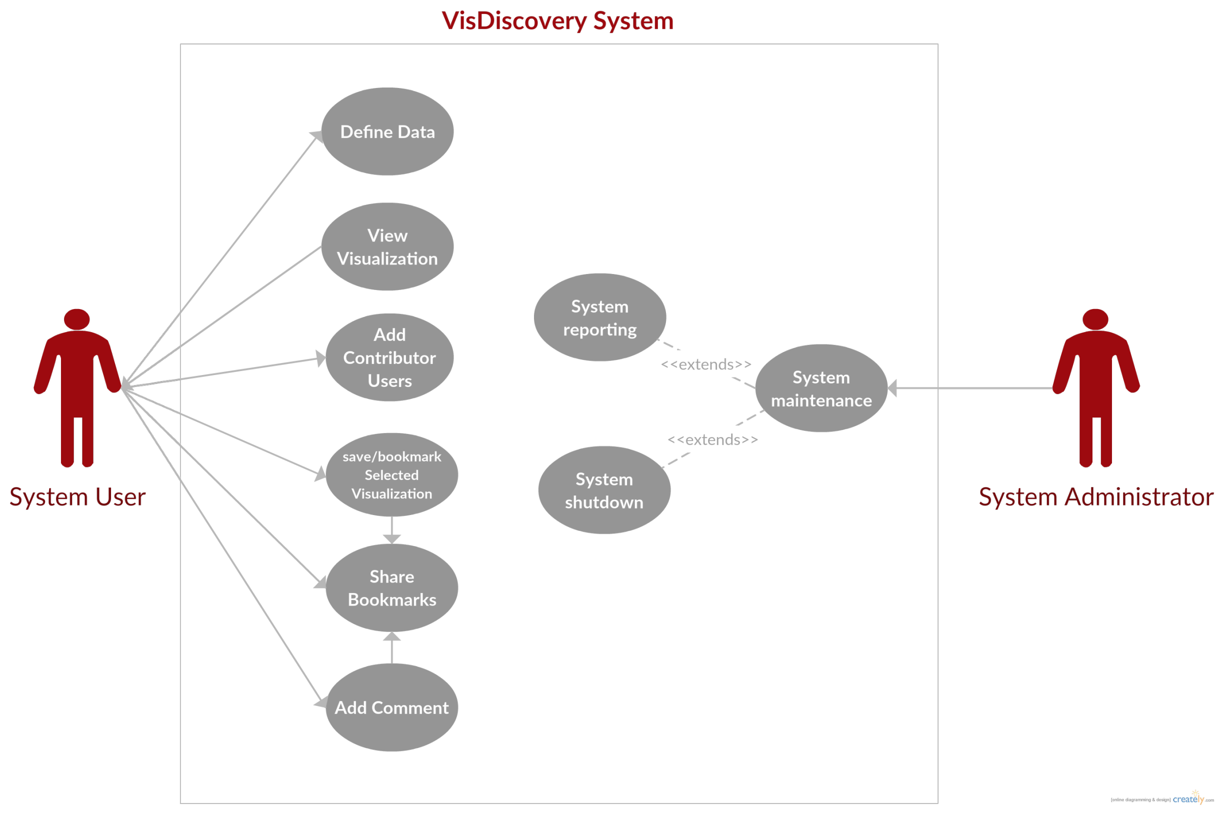


FIGURE 4.3. Use Case Diagram

VisDiscovery has two main actors interacting with the system.

1. **System User** actor which is responsible for the following actions:

   a) **Define data:** define data consists of many operations like

      - sign in/sign up to the system by providing his name, email and password.
      - uploading dataset.
      - Selecting some parameters to be visualized. e.g., x-axis, y-axis, aggregate function.

   b) **View visualizations**

   c) **Add contributors:** System User can add other users as contributers to share his data and the findings with him.

   d) **Bookmark selected visualization**

   e) **Add comments:** User can add comments to the bookmarked visualizations.

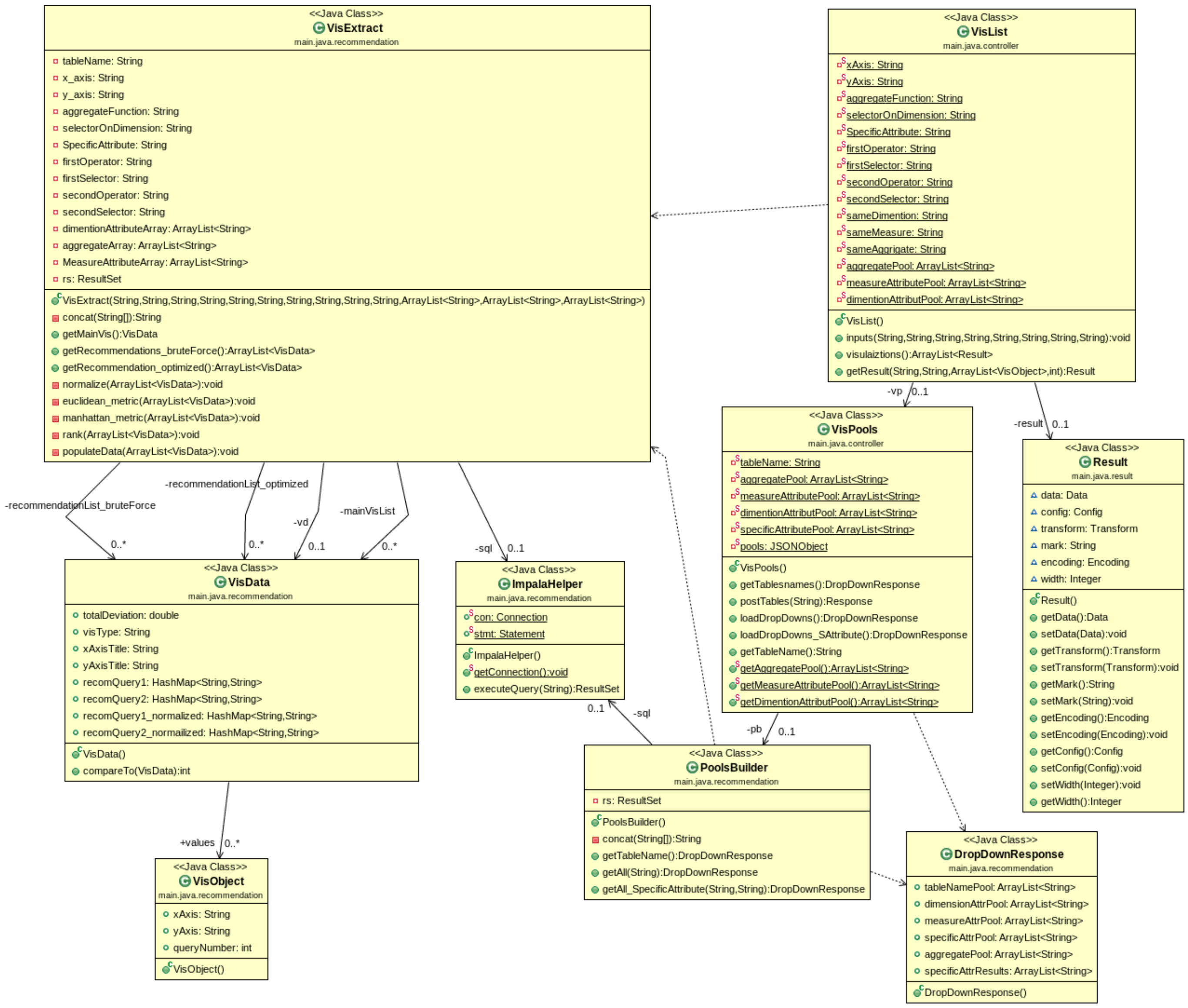
   f) **Share bookmarks:** System user can share his bookmarks with other users.

2. **System administrator** actor which is responsible for the following actions:

   a) **System maintenance:** as shutdown and troubleshooting.

### 4.3.2 Class diagram

A Class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations. It provides a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system.



<<Java Class>>
VisExtract
main.java.recommendation
tableName: String
x_axis: String
y_axis: String
aggregateFunction: String
selectorOnDimension: String
SpecificAttribute: String
firstOperator: String
firstSelector: String
secondOperator: String
secondSelector: String
dimentionAttributeArray: ArrayList<String>
aggregateArray: ArrayList<String>
MeasureAttributeArray: ArrayList<String>
rs: ResultSet
VisExtract(String,String,String,String,String,String,String,String,String,String,ArrayList<String>,ArrayList<String>,ArrayList<String>)
concat(String[]):String
getMainVis():VisData
getRecommendations_bruteForce():ArrayList<VisData>
getRecommendation_optimized():ArrayList<VisData>
normalize(ArrayList<VisData>):void
euclidean_metric(ArrayList<VisData>):void
manhattan_metric(ArrayList<VisData>):void
rank(ArrayList<VisData>):void
populateData(ArrayList<VisData>):void
<<Java Class>>
VisList
main.java.controller
xAxis: String
yAxis: String
aggregateFunction: String
selectorOnDimension: String
SpecificAttribute: String
firstOperator: String
firstSelector: String
secondOperator: String
secondSelector: String
sameDimention: String
sameMeasure: String
sameAggrigate: String
aggregatePool: ArrayList<String>
measureAttributePool: ArrayList<String>
dimentionAttributPool: ArrayList<String>
VisList()
inputs(String,String,String,String,String,String,String,String):void
visulaiztions():ArrayList<Result>
getResult(String,String,ArrayList<VisObject>,int):Result
-vp
0..1
-result
0..1
<<Java Class>>
VisPools
main.java.controller
tableName: String
aggregatePool: ArrayList<String>
measureAttributePool: ArrayList<String>
dimentionAttributPool: ArrayList<String>
specificAttributePool: ArrayList<String>
pools: JSONObject
VisPools()
getTablesnames():DropDownResponse
postTables(String):Response
loadDropDowns():DropDownResponse
loadDropDowns_SAttribute():DropDownResponse
getTableName():String
getAggregatePool():ArrayList<String>
getMeasureAttributePool():ArrayList<String>
getDimentionAttributPool():ArrayList<String>
<<Java Class>>
Result
main.java.result
data: Data
config: Config
transform: Transform
mark: String
encoding: Encoding
width: Integer
Result()
getData():Data
setData(Data):void
getTransform():Transform
setTransform(Transform):void
getMark():String
setMark(String):void
getEncoding():Encoding
setEncoding(Encoding):void
getConfig():Config
setConfig(Config):void
setWidth(Integer):void
getWidth():Integer
-recommendationList_bruteForce
-recommendationList_optimized
-vd
-mainVisList
0..*
0..*
0..1
0..*
-sql
0..1
<<Java Class>>
VisData
main.java.recommendation
totalDeviation: double
visType: String
xAxisTitle: String
yAxisTitle: String
recomQuery1: HashMap<String,String>
recomQuery2: HashMap<String,String>
recomQuery1_normalized: HashMap<String,String>
recomQuery2_normailzed: HashMap<String,String>
VisData()
compareTo(VisData):int
<<Java Class>>
ImpalaHelper
main.java.recommendation
con: Connection
stmt: Statement
ImpalaHelper()
getConnection():void
executeQuery(String):ResultSet
0..1
-sql
-pb
0..1
<<Java Class>>
PoolsBuilder
main.java.recommendation
rs: ResultSet
PoolsBuilder()
concat(String[]):String
getTableName():DropDownResponse
getAll(String):DropDownResponse
getAll_SpecificAttribute(String,String):DropDownResponse
+values
0..*
<<Java Class>>
VisObject
main.java.recommendation
xAxis: String
yAxis: String
queryNumber: int
VisObject()
<<Java Class>>
DropDownResponse
main.java.recommendation
tableNamePool: ArrayList<String>
dimensionAttrPool: ArrayList<String>
measureAttrPool: ArrayList<String>
specificAttrPool: ArrayList<String>
aggregatePool: ArrayList<String>
specificAttrResults: ArrayList<String>
DropDownResponse()

FIGURE 4.4. Class Diagram

### 4.3.3 Sequence diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

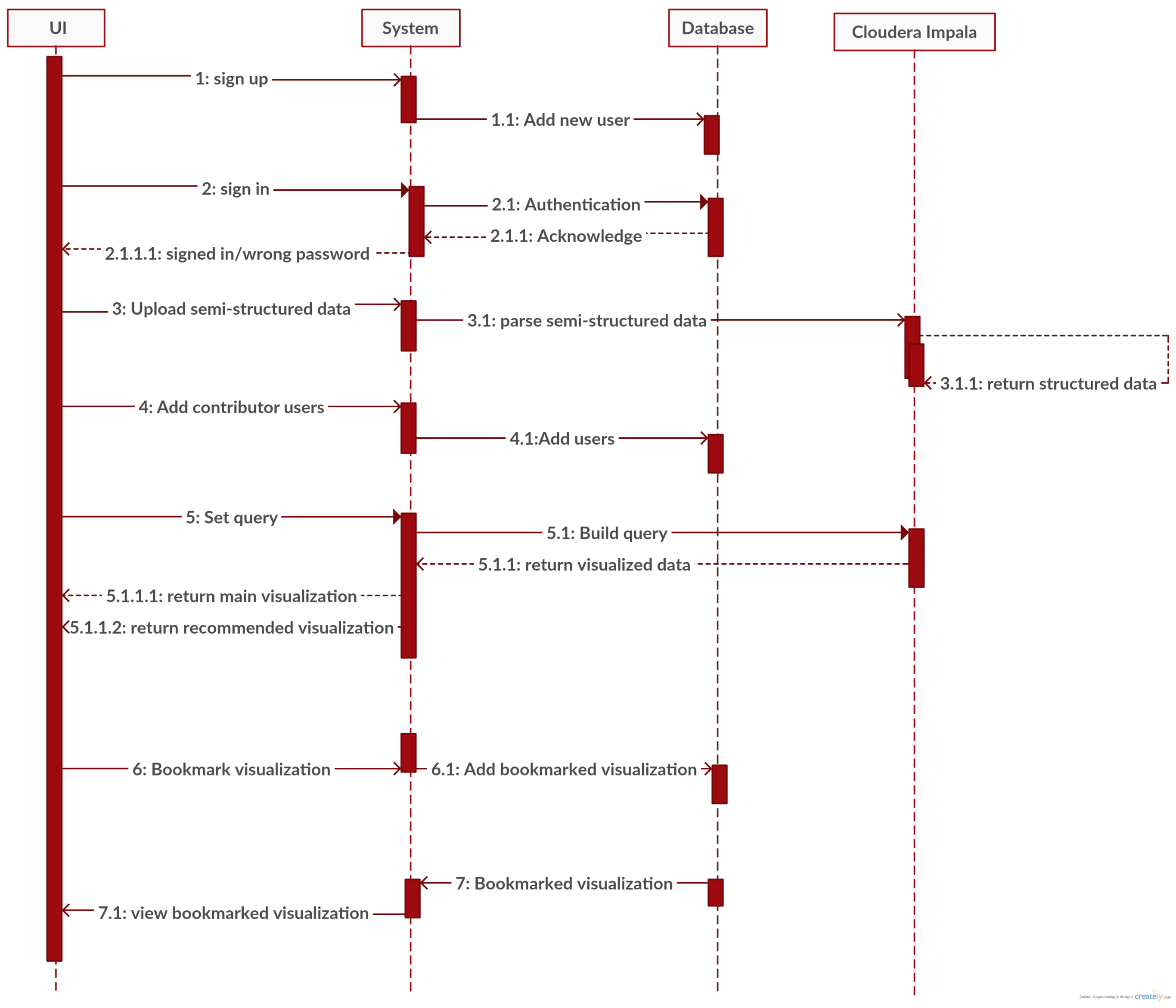

UI
System
Database
Cloudera Impala
1: sign up
1.1: Add new user
2: sign in
2.1: Authentication
2.1.1: Acknowledge
2.1.1.1: signed in/wrong password
3: Upload semi-structured data
3.1: parse semi-structured data
3.1.1: return structured data
4: Add contributor users
4.1:Add users
5: Set query
5.1: Build query
5.1.1: return visualized data
5.1.1.1: return main visualization
5.1.1.2: return recommended visualization
6: Bookmark visualization
6.1: Add bookmarked visualization
7: Bookmarked visualization
7.1: view bookmarked visualization

FIGURE 4.5. Sequence Diagram

VisDiscovery system interactions are arranged with respect to time through the following steps:

1. **Sign Up:**
    - The User sign up to the system by defining his/her information.
        - The System saves those information in the database.
2. **Sign In:**
    - The User sign in to the system by entering his/her email and password.
        - The System checks for authentication.
            - The Database sends an acknowledgment to the System for successful/unsuccessful authentication
                - The System sends the acknowledgement back to the user-interface.
3. **Upload Data-Set:**
    - The User uploads the semi-structured data to the System.
        - The System sends the semi-structured data-set to Cloudera Impala server along with the commands to parse the data and change it to structured.
            - Cloudera Impala parses semi-structured data and saves it as structured data.

4. **Add contributors:**

   - The User can add other users as contributors to share the data-set with them along with any findings.
     - The System adds the contributors in the database.

5. **Set Query and view visualizations:**

   - The User enters the query parameters for the main visualization and the specific attribute for the recommendation by selecting them from the dashboard.
     - The System builds up the query and starts the recommendation algorithm, and then sends the queries to Cloudera Impala to be executed.
       - Cloudera Impala executes the queries, and returns back visualizations data (main and recommended visualizations) to the System.
         - The System sends the main visualization and the recommendation to the user-interface.

6. **Bookmarks:**

   - The User bookmarks certain visualizations.
     - The System saves the bookmarked visualizations data in the database.
       - The User views bookmarked visualizations.

# Chapter 5

# Implementation

Here by we will discuss how we projected our thinking and ideas, into actual work, the tools we used and implemented, we will discuss why we used some - for example - file types over others, for purposes of improving performance. We have made benchmark tests and comparisons, for every tool and algorithm used, and hereby we will justify our choices in this section.

## 5.1 Hadoop

Apache Hadoop [25] is an open-source software framework used for distributed storage and processing of data-sets of big data.

### 5.1.1 Big Data

The term "big data" often refers to the data-sets that are very large and complex that traditional data precessing applications can‚Äôt deal with. We can easily define big data as the three **V**s:

- **Volume:** Organizations collect data from a variety of sources, including business transactions, social media and information from sensor or machine-to-machine data. In the past, storing it would‚Äôve been a problem ‚Äì but new technologies (such as Hadoop) have eased the burden.

- **Velocity:** Data streams in at an unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time.

- **Variety:** Data comes in all types of formats from structured, numeric data in traditional databases to unstructured text documents, email, video, audio, stock ticker data and financial transactions.

Data-sets grow rapidly -in part because they are increasingly gathered by cheap and numerous information- sensing Internet of things devices such as mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks [22].
The world’s technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s [10]; as of 2012, every day 2.5 exa-bytes(2.5 x 1018) of data are generated.

### 5.1.2 What is Hadoop

Hadoop is an open source, Java-based programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.
Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to handle thousands of terabytes of data. Its distributed file system facilitates rapid data transfer rates among nodes and allows the system to continue operating in case of a node failure. This approach lowers the risk of catastrophic system failure and unexpected data loss, even if a significant number of nodes become inoperative. Consequently, Hadoop quickly emerged as a foundation for big data processing tasks, such as scientific analytics.
Hadoop was created by computer scientists Doug Cutting and Mike Cafarella in 2006 to support distribution for the Nutch search engine. It was inspired by Google's MapReduce [6], a software framework in which an application is broken down into numerous small parts. Any of these parts, which are also called fragments or blocks, can be run on any node in the cluster. After years of development within the open source community, Hadoop 1.0 became publically available in November 2012 as part of the Apache project sponsored by the Apache Software Foundation.
Since its initial release, Hadoop has been continuously developed and updated. The second iteration of Hadoop (Hadoop 2) improved resource management and scheduling. It features a high-availability file-system option and support for Microsoft Windows and other components to expand the framework's versatility for data processing and analytics.

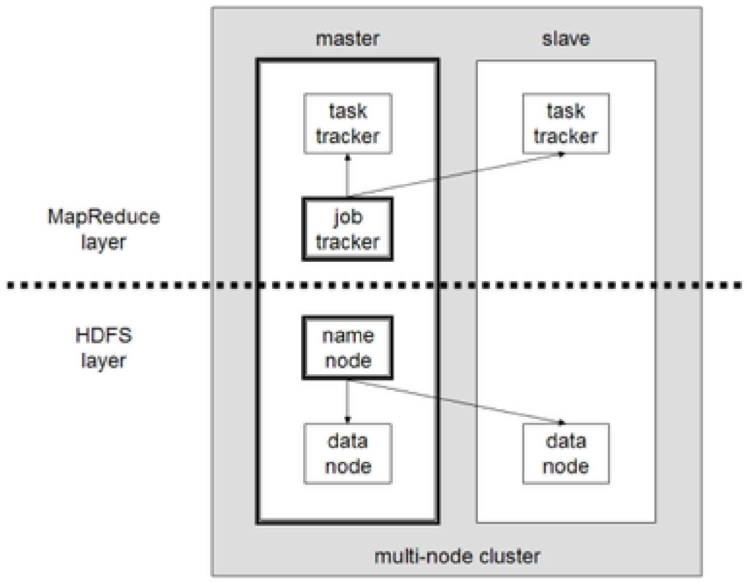


FIGURE 5.1. Hadoop Architecture

### 5.1.3 Hadoop architecture

Hadoop consists of the Hadoop Common package, which provides file system and operating system level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2)[58] and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the Java ARchive (JAR) files and scripts needed to start Hadoop.
For effective scheduling of work, every Hadoop-compatible file system should provide location awareness, the name of the rack (or, more precisely, of the network switch) where a worker node is. Hadoop applications can use this information to execute code on the node where the data is, and, failing that, on the same rack/switch to reduce backbone traffic. HDFS uses this method when replicating data for data redundancy across multiple racks. This approach reduces the impact of a rack power outage or switch failure; if any of these hardware failures occurs, the data will remain available.
A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a Job Tracker, Task Tracker, NameN-

ode, and DataNode. A slave or worker node acts as both a DataNode and TaskTracker, though it is possible to have data-only and compute-only worker nodes. These are normally used only in nonstandard applications. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard startup and shutdown scripts require that Secure Shell (SSH) be set up between nodes in the cluster.
In a larger cluster, HDFS nodes are managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thereby preventing file-system corruption and loss of data. Similarly, a standalone JobTracker server can manage job scheduling across nodes. When Hadoop MapReduce is used with an alternate file system, the NameNode, secondary NameNode, and DataNode architecture of HDFS are replaced by the file-system-specific equivalents.

### 5.1.4 HDFS

The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications.
HDFS is a distributed file system that provides high-performance access to data across Hadoop clusters. Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.
Because HDFS typically is deployed on low-cost commodity hardware, server failures are common. The file system is designed to be highly fault-tolerant, however, by facilitating the rapid transfer of data between compute nodes and enabling Hadoop systems to continue running if a node fails. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail.
When HDFS takes in data, it breaks the information down into separate pieces and distributes them to different nodes in a cluster, allowing for

parallel processing. The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others. As a result, the data on nodes that crash can be found elsewhere within a cluster, which allows processing to continue while the failure is resolved.
HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. It uses a master/slave architecture, with each cluster consisting of a single NameNode that manages file system operations and supporting DataNodes that manage data storage on individual compute nodes.

### 5.1.5 Pros and Cons of Hadoop

| Pros | Cons |
|---|---|
| Scalable | Security concerns |
| Cost effective | Vulnerable by nature |
| Flexible | Not fit for small data |
| Resilient to failure | Slow |

TABLE 5.1. Pros and Cons of Hadoop

VisDiscovery mainly relies on SQL-On-Hadoop concept, using Cloudera Impala, as we will discuss in the next section.

## 5.2 Cloudera Impala

The Apache Hadoop ecosystem is very data-centric, making it a natural fit for database developers with SQL experience. Much application development work for Hadoop consists of writing programs to copy, convert or recognize, and analyze data files. A lot of effort goes into finding ways to do these things reliably, on a large scale, and in parallel across clusters of networked machines. Impala focuses on making these activities fast and easy, without requiring you to have a PhD in distributed computing, learn a lot of new APIs, or write a complete program where your intent can be conveyed with a single SQL statement.

### 5.2.1 Why Impala

The Cloudera Impala [20] project arrives in the big data world at just the right moment. Data is growing fast out stripping already know means of saving data on single servers, Hadoop software stack is opening a field for larger audience of users and developers for the world of big data and multi cluster storage.
Impala brings hight degree of flexibility to the familiar database processes, you can access the same data with a combination of impala and other Hadoop components such as apache hive, apache pig and and Cloudera search without converting any data. Introducing parquet file columnar file format makes it simple to reorganize data for maximum query performance.
Big data resembled batch jobs that had required in the past days, or weekends to run, throught day and night, to minimize theses jobs into a matter of hours or even minutes, and with the help of impala, running SQL queries won't be easier, referring to this as responsiveness.
Impala integrates with existing Hadoop components, security, metadata, storage management and file format to still have the Hadoop flexibility,

and add capabilities that make SQL queries, much easier and faster than before.
Methods of filtering, calculating, sorting, counting and formatting that SQL offers, allows you to delegate those operations into the impala query engine, rather that generating a big data chunk of raw results, then coding a client-side business logic software that performs this operations.
The data files that impala does support are all on the open, documented, interpolated formats, some of them are even human readable like csv, and json for some point. If you want to use impala alongside hadoop components, you can do that without copying or converting data formats. You can keep using original data with files rather than being faced with the problem of conversion.
Impala architecture provides such a speed boost to SQL queries on Hadoop data that will change the way you work, the fast turn around for impala queries opens up whole new categories of problems that you can solve. Instead of treating Hadoop data analysis as a batch process that requires extensive planning and scheduling, you will get the results on time you want them. Instead of doing a mental content switch as you wait for each query job to finish, you will now be able to immediately run the query and get the results immediately to fine tune it. The rapid iterations help you zero in on the best solution without disrupting your work flow, instead of trying to shrink your data set, you can analyze everything you have, producing more accurate answers. You might receive raw data in simple formats such as delimited text files, which are bulky and not particularly efficient for querying, these aren't critical aspects for exploratory business intelligence.

### 5.2.2 Impala vs. Hive vs. SparkSQL

There are competitors for impala, in terms of software managing queries on big data distributed on multi cluster nodes. They all happen to lie in the same category of being - SQL on Hadoop - with each having a different purpose and different objective to complete.

- **Hive:** Is the apache warehouse data software that facilitates querying and managing large datasets residing in distributed storage. It built on top of apache Hadoop, with tools that enable easy data extractions, transformation, or loading. It also have a semi SQL-like query language called QL, that enables users familiar with SQL to query data. Hive, itself will not be a great choice in term of multiuser support, and does not offer that great capabilities that Impala does offer, still there is a choice of Hive-on-Spark, that will narrow the time window needed for such processing, but all in all Hive is not a great choice for business intelligence.

- **SparkSQL:** It is also an SQL on hadoop, that supports data storage in HDFS, and apache HBase, likewise Hive and Impala. It supports multiple file formats like Parquet, Avro, Text, JSON, and ORC, it offers security over authentication if given used via a yarn application. It also supports Amazon S3 as a file system for storage instead of HDFS or Hbase, But, here comes the catch, spark let users selectively use SQL constructs when writing Spark pipelines, in another meaning, it is not intended to be a general-purpose SQL layer for exploratory analysis, again it is one major difference that Impala excels at. However, Spark SQL reuses the Hive front end and metastore, to try and Hive you a full compatible existing hive data and queries similar to what impala offers. The biggest advantage however of spark, that it has the best performance between them, but least flexibility.

### 5.2.3 Impala File Formats

Impala supports 5 file formats, **Parquet**, **Avro**, **RCFile**, **SequenceFile** and **Unstructured text**. You can use different file formats similar to the storage engines or special kinds of tables in other database systems. Some file formats are more convenient to produce, such as human readable text. Others are more compact because of compression, or faster for data-warehouse-style queries because of column-oriented layout. The key advantage for Impala is that each file format is open, documented, and can be written and read by multiple Hadoop components, rather than being "black box" where the data can only be accessed by custom-built software. So you pick the best tool for each job:collecting data (typically with Sqoop and Flume), transforming it as it moves through the data pipeline (typically with Pig), and analyzing it with SQL queries (Impala,naturally) or programs written for frameworks such as MapReduce or Spark.
Mostly we are concerned about 3 file formats, Text (most convenient and flexible), Parquet file format (most compact and query optimized), and Avro file format, Other formats are not optimized for the kinds of analytic queries that is done with Impala.

- **Text File Format:** It's the most familiar and convenient for beginners. It's also the default file format for **CREATE TABLE** command. It's very flexible, with a choice of delimiter characters. You can crate textual data files with a simple Unix command, or Perl or Python script on any computer whether or not it's running Haddop. Within Impala you can change your mind any time about whether a column is a **STRING**, **INT**, **TINYINT**, **BIGNT**, and so on.
  However it is the bulkiest format, thus the least efficient for serious Big Data applications. The number 123457 takes up 7 bytes on disk; -1234567 takes up 8 bytes on disk; -1234.567 takes up 29 bytes. When you're with billions of rows, each unnecessary character represents gigabytes of wasted space on disk, and proportional amount of

wasted I/O, wasted memory, and wasted CPU cycles during queries.

- **Apache Avro:** Is a language-neutral data serialization system. It was developed by Doug Cutting, the father of Hadoop. Since Hadoop writable classes lack language portability, Avro becomes quite helpful, as it deals with data formats that can be processed by multiple languages. Avro is a preferred tool to serialize data in Hadoop. However, impala wise, Avro still limits the capability of its SQL commands and features, as well as it lowers the performance while querying large file chunks dramatically.

- **Parquet:** Is a file format originated from a collaboration work between twitter and Cloudera, optimized for data-warehouse-style, a binary file format, a truly compressed one, that packs for example numeric values into small number of bytes ( 4 or 8), a **Boolean** is packed into a single bit, a **TimeStamp** is represented in a relatively few bytes. So all else being equal, a Parquet file is smaller that the equivalent text file. Parquet is a columnar based file type, meaning that records are saved in a columns manner rather that the normal row based manner. This speeds up common kinds of data warehouse queries, such queries do not need to examine or retrieve all the columns; **SELECT *** is way too expensive when the result sets are huge or the tables have hundreds or thousands of columns. But the queries do typically need to examine all or most of the values from particular columns, because they are computing sums, averages and son on across long time periods or big geographic regions.
  If the same value is represented over and over Parquet uses run-length encoding to condense that sequence down to two values: the value that;s repeated, and how many times it's repeated. If a column has a modest number of different values up to 16K, Parquet uses dictionary encoding for that column: it makes up numeric IDs for the values and stores those IDs in the data file along with one copy

of the values, rather than repeating the values over and over. This automatically provides space reduction than repeating the values.

**Table 5.2** shows the different types of formats compression codecs and whether Impala can create and Insert in this type or not.

| File Type | Format | Compression codecs | Impala can create? | Impala can insert? |
|---|---|---|---|---|
| Parquet | Structured | Snappy, gzip | Yes | Yes |
| Text | Unstructured | LZO, gzip, bzip2, Snappy | Yes | Yes |
| Avro | Structured | Snappy, gzip, deflate, bzip2 | Yes | No |
| RCFile | Structured | Snappy, gzip, deflate, bzip2 | Yes | No |
| SequenceFile | Structured | Snappy, gzip, deflate, bzip2 | Yes | No |

TABLE 5.2. File Format Support in Impala

### 5.2.4 Connecting to Impala

Impala supports the standard JDBC interface, allowing access from commercial Business Intelligence tools and custom software written in Java or other programming languages. The JDBC driver allows you to access Impala from a Java program that you write, or a Business Intelligence or similar tool that uses JDBC to communicate with various database products.
Setting up a JDBC connection to Impala involves the following steps:

- Verifying the communication port where the Impala daemons in your cluster are listening for incoming JDBC requests.
- Installing the JDBC driver on every system that runs the JDBC-enabled application.
- Specifying a connection string for the JDBC application to access one of the servers running the impalad daemon, with the appropriate security settings.
- **Configuring the JDBC Port:** The default port used by JDBC 2.0 and later (as well as ODBC 2.x) is 21050. Impala server accepts JDBC connections through this same port 21050 by default.This port must be available for communication with other hosts on the network, for example, that it is not blocked by firewall software. If the JDBC client software connects to a different port, alternative port number must be specified with the **hs2_port** option when starting impalad.
- **Choosing the JDBC Driver:** In Impala 2.0 and later, you have the choice between the Cloudera JDBC Connector and the Hive 0.13 JDBC driver. However Cloudera recommends using the Cloudera JDBC Connector where practical. Both the Cloudera JDBC 2.5 Connector and the Hive JDBC driver provide a substantial speed

increase for JDBC applications with Impala 2.0 and higher, for queries that return large result sets.

Most Impala SQL features work equivalently through the impala-shell interpreter of the JDBC or ODBC APIs. The following are some exceptions to keep in mind when switching between the interactive shell and applications using the APIs:

- Queries involving the complex types (ARRAY, STRUCT, and MAP) require notation that might not be available in all levels of JDBC and ODBC drivers.
- The complex types available in CDH 5.5 / Impala 2.3 and higher are supported by the JDBC getColumns() API. Both MAP and ARRAY are reported as the JDBC SQL Type ARRAY, because this is the closest matching Java SQL type. This behavior is consistent with Hive. STRUCT types are reported as the JDBC SQL Type STRUCT.

### 5.2.5 PESTO

PESTO is a query builder that we developed to facilitate the connection to Cloudera impala-shell. We developed this tool after trying different methodologies and getting bad results to connect to Impala server from a java application. We created a library that can be added to any java application providing easy and familiar way to connect to Impala servers, execute queries and receive the results.
This tool is written in java and uses the open source JDBC driver that Cloudera Impala offers adding to it more functionality through providing our own high level methods that the user can use them more easily.

- **create the java object, all you have to do is :**

  ```
  JDBCclass jdbcObj = new JDBCclass(DRIVER_NAME,
  SERVER_NAME, Constants.JDBC_DEFAULT_PORT);
  ```

  Replace DRIVER_NAME with "jdbc" or "hive" note that we currently support jdbc at this stage, hive to be supported sooner than later, replace SERVER_NAME with your IP Address of the machine, or the name server if it has one. It is strongly advised not to change the Impala Default Port, but if you have changed it, the change Constants.JDBC_DEFAULT_PORT into your changed port.

- **Table 5.3** Explores the methods in jdbcObj

| Method Name | Parameters | Return | Description |
|---|---|---|---|
| createTable | String table_name, LinkedHashMap (String, String) table_data, String store_type | boolean | Method to create a table from the given parameters, table_data takes in each column name and type. |
| dropTableFrom Impala | String table_name | boolean | Method to drop a table from impala |
| dropDatabase FromImpala | String database_name | boolean | Method to drop a database from impala |
| getPrepared StmtFor Insert | String table_name, int no_of_params, int no_of_rows | Prepared Statement | Method to return a prepared statement, with ? for sql injection already in right places |
| selectFrom Table | String table_name, String select_params, String where_params, String order_params, String group_params | ResultSet | Method for select, table_name takes in table name, select_params takes in what would you select * for all |
| getPrepared StmtFor CustomQuery | String query | Prepared Statement | This method returns a prepared statement object for your custom query, you would insert the string with ? and then the variables would be set via java built in methods |

TABLE 5.3. PESTO Methods

## 5.3 Data Discovery

### 5.3.1 Semi-Structured to Structured

Data comes in different shapes and sizes, in another meaning, we have different data types lying everywhere, not all being support by the previous technologies we discussed (Impala, Hive, Spark). There are three categories of data lying around according to their structure:

- **Unstructured Data:** data that does not have a regular shape or follow a pattern for organization, it might be some Text, an email, or a binary stream that represents a 3D video file, these data structures are very bad for querying, and not all softwares supports them, impala does support it though, saving the data on HDFS.
- **Semi-structured Data:** data that does have a "semi" pattern that might be not fully readable by a human being, but still can be parsed and read using a written algorithm for example, these semi-structured data are like (CSV, XML or JSON), they mostly represent 20% of our data.
- **Structured Data:** and they represent data fully structured in a table either row based or column based, that have full access for querying like SQL tables, or excel sheets, they do unfortunately represent a small portion of our data.

While structured data represents almost 10% of our everyday data, the unstructured and semi structured data represents the rest, making the conversion process a necessity that would should be implemented in a project like VisDiscovery, converting the semi structured csv data to structured data is a main concern for our project, all of which implemented in the data discovery module.
At the start of the project, before we went for the decision of using impala

as the underlying architecture for SQL-on-Hadoop. we did implement the process using native java code, for parsing the csv file, into chunks of data arraylist interfaces and hashmaps, then processing these chunks into a sqLite file, that file would be later uploaded on a multi node cluster then used through hive, for querying on and retrieving data. Now this did have a lot of problems, apart from the sql lite file being separated in multiple node machines, and having issues resembling them again during processing or querying, this process did add a heck load of overtime during runtime, and with dealing with large data chunks, this also will allocate huge memory for the data holding arraylists and hashmaps, apart also from possibilities of system crashes due to huge memory allocations, and possible later memory leak, we knew that this approach would be a very wrong doing, and we terminated thinking through this idea.
Cloudera impala was our decision then, after days and weeks of research and progress, it offered a very stable SQL-on-Hadoop architecture, offering what would save us a lot of boiler plate codes, that would be unnecessary with impala, for inserting and querying data, but did it support converting data? Yes, and with even a greater choice of saving the data as TEXT, and what came into glory that it would also support querying data that is saved as a TEXT file in impala.
Impala does have an awesome set of commands that would save you the tons of code about how to write and migrate data from csv semi-structured type into full structured sql, we used these tools, imported into the PESTO library we did use for connecting as the jdbc bridge between our VisDiscovery engine and impala as we mentioned before , to automatically convert a locally defined file on the impala server, into the HDFS file system, that is ready for querying and visualizing.
These set of commands do perform a great functionality of converting the data from semi structured to fully structured without the overhead, and the possibility of system crashes or memory leaks when dealing with large files.

## 5.4 Vega-Lite

Vega-Lite [21] is a high-level grammar for interactive graphics. It provides a concise JSON syntax for supporting rapid generation of interactive multi-view visualizations to support analysis. Vega-Lite can serve as a declarative format for describing and creating data visualizations. To use Vega-Lite, our compiler compiles a Vega-Lite specification into a lower-level, more detailed Vega specifications and rendered using Vega‚Äôs compiler.

### 5.4.1 Visualization Grammar

Vega-Lite combines a grammar of graphics with a novel grammar of interaction. In this section, we describe Vega-Lite's basic visual encoding constructs and an algebra for view composition.

1. **Unit Specification**
   A unit specification describes a single Cartesian plot, with a backing data set, a given mark-type, and a set of one or more encoding definitions for visual channels such as position (x, y), color, size, etc. Formally, a unit view consists of a four tuple:
   unit := (data, transforms, mark-type, encodings)
   The data definition identifies a data source, a relational table consisting of records (rows) with named attributes (columns). This data table can be subject to a set of transforms, including filtering and adding derived fields via formulas.

2. **View Composition Algebra**
   Given multiple unit specifications, composite views can be created using a set of composition operators (Layer, Concatenation, Facet and repeat)

3. **Nested views**
   Composition operators can be combined to create more complex nested views or dashboards, with the output of one operator serving as input to a subsequent operator. For instance, a layer of two unit views might be repeated, and then concatenated with a different unit view.
   The one exception is the layer operator, which only accepts unit views to ensure consistent plots. For concision, two dimensional faceted or repeated layouts can be achieved by applying the operators to the row and column channels simultaneously. When faceting a composite view, only the dataset targeted by the operator is partitioned; any other datasets specified in sub-views are replicated.

### 5.4.2 Vega-Lite Specification

Vega-Lite specifications are JSON objects that describe a diverse range of interactive visualizations. The simplest form of specification is a specification of a single view, which describes a view that uses a single mark type to visualize the data. Besides using a single view specification as a standalone visualization, Vega-Lite also provides operators for composing multiple view specifications into a layered or multi-view specification. These operators include layer, facet, concat, repeat.

1. **Data**
   The basic data model used by Vega-Lite is tabular data, similar to a spreadsheet or a database table. Individual data sets are assumed to contain a collection of records, which may contain any number of named data fields.
   Vega-Lite's optional top-level data property describes the visualization's data source as part of the specification, which can be either inline data (values) or a URL from which to load the data (URL).

Alternatively, we can create an empty, named data source (name), which can be bound at runtime.

2. **Transformation**
   Data transformations in Vega-Lite are described via either top-level transforms (the transform property) or inline transforms inside encoding (aggregate, bin, timeUnit and sort).
   When both types of transforms are specified, the top-level transforms are executed first based on the order in the array. Then the inline transforms are executed in this order: bin, timeUnit, aggregate and sort.

   - **Bin:** transform in the transform array has the following properties.
   - **Calculate:** used to derive a new field.
   - **Filter:** used to filter data.
   - **Summarize:** transform in the transform array with specific properties.
   - **TimeUnit:** transform in the transform array with other specific properties.

3. **Mark**
   Marks are the basic visual building block of a visualization. They provide basic shapes whose properties (such as position, size, and color) can be used to visually encode data, either from a data field, or a constant value.
   The mark property of a single view specification can either be (1) a string describing mark type or (2) a mark definition object.

4. **Encoding**
   An integral part of the data visualization process is encoding data with visual properties of graphical marks. Vega-Lite's top-level encoding property represents key-value mappings between encoding channels (such as x, y, or color) and its definition object, which describes the encoded data field or constant value, and the channel's scale and guide (axis or legend).

**Figure 5.2** shows the class diagram and how we implemented those specifications(Mark, encoding, etc.) in our engine, so as to return one nested JSON object (Result) that is to be sent to the front end to be visualized.

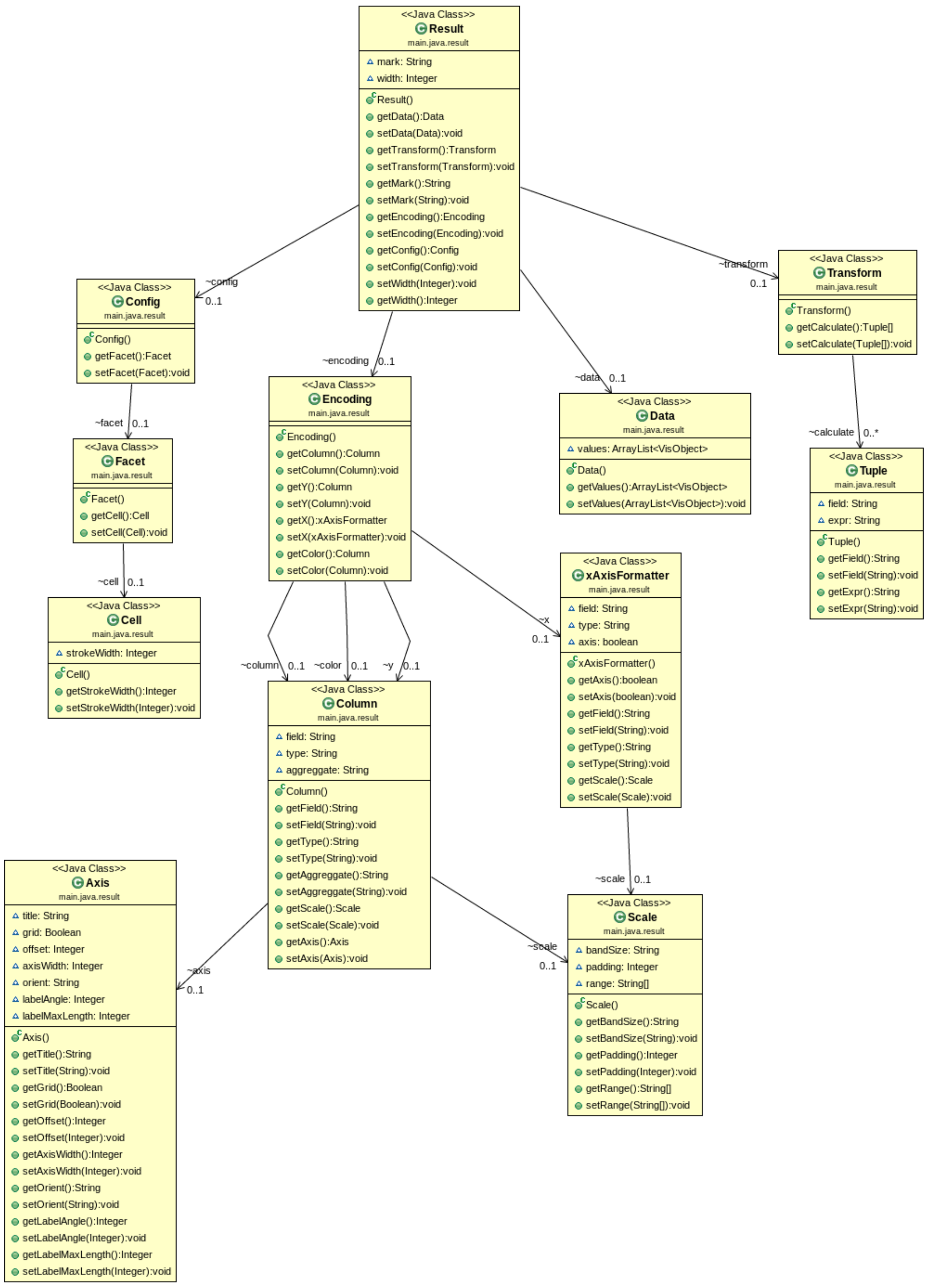


FIGURE 5.2. VisDiscovery Vega-Lite Class Diagram

### 5.4.3 D3

Vega-Lite uses D3.js to render visualizations. D3 (Data-Driven Documents or D3.js) is a JavaScript library for visualizing data using web standards. D3 helps you bring data to life using SVG, Canvas and HTML. D3 combines powerful visualization and interaction techniques with a data-driven approach to DOM manipulation, giving you the full capabilities of modern browsers and the freedom to design the right visual interface for your data. D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction. D3 is not a monolithic framework that seeks to provide every conceivable feature. Instead, D3 solves the crux of the problem: efficient manipulation of documents based on data. This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as HTML, SVG and CSS.
With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviors for interaction and animation. D3‚Äôs functional style allows code reuse through a diverse collection of official and community-developed modules.

- **Selection:**
  D3 employs a declarative approach, operating on arbitrary sets of nodes called selections and you can still manipulate individual nodes as needed. D3 provides numerous methods for mutating nodes: setting attributes or styles; registering event listeners; adding, removing or sorting nodes; and changing HTML or text content. These suffice for the vast majority of needs. Direct access to the underlying DOM is also possible, as each D3 selection is simply an array of nodes.

- **Transformation:**
  D3 does not introduce a new visual representation. D3's vocabulary of graphical marks comes directly from web standards: HTML, SVG, and CSS. For example, you can create SVG elements using D3 and style them with external stylesheets. You can use composite filter effects, dashed strokes and clipping. If browser vendors introduce new features tomorrow, you,Äôll be able to use them immediately,Äîno toolkit update required. And, if you decide in the future to use a toolkit other than D3, you can take your knowledge of standards with you!.

- **Transitions:**
  D3's focus on transformation extends naturally to animated transitions. Transitions gradually interpolate styles and attributes over time. Tweening can be controlled via easing functions such as "elastic", "cubic-in-out" and "linear". D3,Äôs interpolators support both primitives, such as numbers and numbers embedded within strings (font sizes, path data, etc.), and compound values. You can even extend D3,Äôs interpolator registry to support complex properties and data structures. By modifying only the attributes that actually change, D3 reduces overhead and allows greater graphical complexity at high frame rates.

## 5.5 Jersey

Jersey is an open source RESTful Web Services framework, for developing RESTful Web Services in Java.

### 5.5.1 REST API

REST is an architectural style which is based on web-standards and the HTTP protocol. REST was first described by Roy Fielding in 2000. In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods. In a REST based architecture you typically have a REST server which provides access to the resources and a REST client which accesses and modifies the REST resources.
REST specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.
Every resource should support the HTTP common operations. REST allows that resources have different representations, e.g., text, XML, JSON etc. The REST client can ask for a specific representation via the HTTP protocol (content negotiation).
The GET, POST, PUT, PATCH and DELETE methods are typical used in REST based architectures.

**Table 5.4** is an explanation of these operations

| Route | Method | Action |
|---|---|---|
| /resource | POST | Insert resource. |
| /resource | GET | Retrive all resources. |
| /resource/id | GET | Retrive resource with id specific from resources. |
| /resource/id | PUT | Update the whole resource even if only one attribute updated with id specific. |
| /resource/id | PATCH | Update some or one attribute in the resource with id specific. |
| /resource/id | DELETE | Delete resource with id specific. |

TABLE 5.4. Standard Rest Methods

### 5.5.2 RESTFUL web services

A RESTFul web services are based on HTTP methods and the concept of REST. A RESTFul web service typically defines the base URI for the services, the supported MIME-types (XML, text, JSON, user-defined, ...etc) and the set of operations (POST, GET, PUT, DELETE) which are supported.
RESTful web services are built to work best on the Web.The following principles encourage RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI:** A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

- **Uniform interface:** Resources are manipulated using a fixed set of four create, read, update, delete operations:PUT, GET, POST, and DELETE.
- **Self-descriptive messages:** Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.
- **Stateful interactions through hyperlinks:** Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

### 5.5.3 JAX-RS with Jersey

Java defines REST support via the Java Specification Request (JSR 311). This specification is called JAX-RS (The Java API for RESTful Web Services). JAX-RS uses annotations to define the REST relevance of Java classes. JAX-RS is a set of APIs to developer REST service. JAX-RS is part of the Java EE6, and make developers to develop REST web application easily.
**Jersey** provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation. Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides it's own API that extend the JAX-RS toolkit with additional features and utilities to

further simplify RESTful service and client development.
The Jersey implementation provides a library to implement Restful web services in a Java servlet container. The Jersey implementation also provides a client library to communicate with a RESTful web service.We used jersey as framework for backend implementation.
**Table 5.5** describe each URL and its usage.

| Route | Method | Usage |
|---|---|---|
| /api/charts/inputs | POST | used to send the selected attributes required for visualization. |
| /api/charts/inputs /visualizations | GET | Used to generate required data for visualization and return it as JSON object for Vega light to visualize it. |
| /api/choices/postTableName | POST | Used to get selected table name from the user side. |
| /api/choices/getPools | GET | Used to load drop down menus with required data for the user to choose from. |
| /api/choices/getSAttrPoolsRes | GET | Used to load the specific attribute menu with data for the user to choose from. |

TABLE 5.5. VisDiscovery Routes

# Chapter 6

# Case Studies

This section contains the difference data-sets we operated on in order to prove the concept of VisDiscovery, the description of each of these data and how we prepared the data before using it and the output VisDiscovery found and recommended after using each of these data.

## 6.1 Wuzzuf

Wuzzuf.net is created and managed by BasharSoft, a technology firm founded in 2009 and one of very few companies in the MENA region specialized in developing Innovative online Recruitment Solutions for top enterprises and organizations.
Since May 2012, Wazzuf successfully served 10,000+ top companies and employers in Egypt, 1.5 MILLION CVs were viewed on their platform and 100,000+ job seekers directly hired through them. In total, 250,000+ open job vacancies were advertised and now, 500,000+ users visit their website each month looking for jobs at top Employers.
They are now expanding their success to the Gulf region. They're helping

employers and job seekers from UAE, Qatar and other gulf countries find their right match through their intelligent real-time recommendations and around the clock support.
They have released a sample of their dataset including 2 CSV files:

- **Wuzzuf_Job_Posts_Sample:** a sample of jobs posted on WUZZUF during 2014-2016.
- **Wuzzuf_Applications_Sample:** the corresponding applications (Excluding some entries).

We are concern here with **Wuzzuf_Job_Posts_Sample**

### 6.1.1 Catalog & Data Description

**Wuzzuf_Job_Posts_Sample** contains the following fields:

- **post_id:** the unique identifier of each job.
- **city_name:** the city where the job is in it.
- **title_name:** the title of the job.
- **job_category_1**, **job_category_2** and **job_category_3:** the most 3 relevant categories of the job post, e.g., IT/Software Development
- **job_industry_1**, **job_industry_2** and **job_industry_3:** the most 3 relevant industries of the job post, e.g., Computer Software
- **salary_minimum:** the minimum salary limit of the job.
- **salary_maximum:** the maximum salary limit of the job.
- **career_level:** the level the job need, e.g., Entry Level, Manager
- **num_vacancies:** the number of open vacancies for this job post.

- **experience_years:** the number of years of experience.
- **post_date:** the timestamp of the post.
- **views:** count of number who viewed the job post.
- **job_description:** the detailed description for the job post.
- **job_requirements:** the main job requirements for the job post.
- **payment_period:** the salary payment interval, e.g., Per Month
- **currency:** the salary currency, e.g., Egyptian Pound
- **applicants_count:** number of applicants who applied for this job

### 6.1.2 Data cleaning

The published data-set had many free-text fields, Wuzzuf system does not enforce a certain list of items to choose for them, which makes processing and aggregation difficult. A common handling such as lower case all values and remove trailing spaces was performed.
**BADR for Information Technology** [1] company gratefully has provided us with a cleaned version of Wazzuf data-set, and here I will say exactly what they have done and what attributes they changed in order to have a clean version which would be more useful and easier to operate on. After examining the data carefully they came up with the fact that some fields needed special handling such as:

1. **city_name:** this attribute is free text attribute, which represents Egyptian cities, but it has the following problems:
    - Misspelling of words. (i.e. cairo , ciro , ciaro)
    - Arabic names
    - Outside Egypt cities (i.e. riyadh, doha)

- General Cities (i.e. all egypt cities , any location)
- Group of Cities (i.e."cairo, alexandria - damanhor")

They solved all the above issues by :

- Outside Egypt cities: a static list of outside cities has been mapped to category "outside".
- Arabic (Non-ascii) names: has been replaced statically be the corresponding english words.
- General Cities: a static list of outside cities has been mapped to category "any"
- Remove Not Needed substrings such as "el" and "al".
- Replace "and" and "or" substrings with "-" to be splitted on next steps.
- Group of Cities : attribute has been splitted on several delimiters.
- Misspelling of words: a static list of valid cities and its states in Egypt has been created , each misspelled word has been mapped to the most similar word of valid cities, a threshold T has been used to accept only similarities above that threshold, otherwise city will mapped to "any" category.
- Added new **state** attribute by mapping each city_name to its state from valid cities& states categories.

2. **job_category_1**, **job_category_2** and **job_category_3** attributes: cleaning was done by removing placeholder text "Select" from all 3 attributes, and merging the 3 attributes into one attribute called **job_categories**

3. **job_industry_1**, **job_industry_2**, **job_industry_3** attributes: : cleaning was done by removing placeholder text "Select" from all 3 attributes, and merging the 3 attributes into one attribute called **job_industries**.

4. **experience_years** attribute: they the experience_years attribute to 2 new attributes **experience_years_min** and **experience_years_max** , which contains the minimum and maximum years respectively needed for job.

5. **post_date** attribute: a new attribute called **post_timestamp** was generated which has the POSIX timestamp value of the post_date attribute (i.e., the number of seconds that have elapsed since January 1, 1970 midnight UTC/GMT).

6. **job_description** and **job_requirements** attributes: they noticed that job_requirements attribute are normally empty, so they added new derived attribute called **description** which contains the concatenation of job_requirements and job_description attributes.

### 6.1.3 Findings

In this section the result of using VisDiscovery with Wuzzuf data-set are presented proofing that VisDiscovery facilitates the analysis through recommending interesting visualizations (those who have the biggest deviation) and discard others visualizations automatically.

- For instance, if we want to know how cities of Egypt can be affected by different attributes, we are concern here with **Alexandria** and **Cairo** as they are the biggest cities in Egypt, therefore it is expected that they also have the most job offers and salaries.

All what the user have to do after choosing Wazzuf data-set is to determine the **city_name** as the **Specific Attribute** and then choosing **Alexandria** and **Cairo** as the first and second selectors respectively, so that VisDiscovery starts building the queries and runs the recommendation algorithm on them using these two selectors.
The first visualization VisDiscovery recommends is **figure 6.1**.
It visualizes the salary currency VS. the average minimum salary in each of the two cities, Alexandria and Cairo.

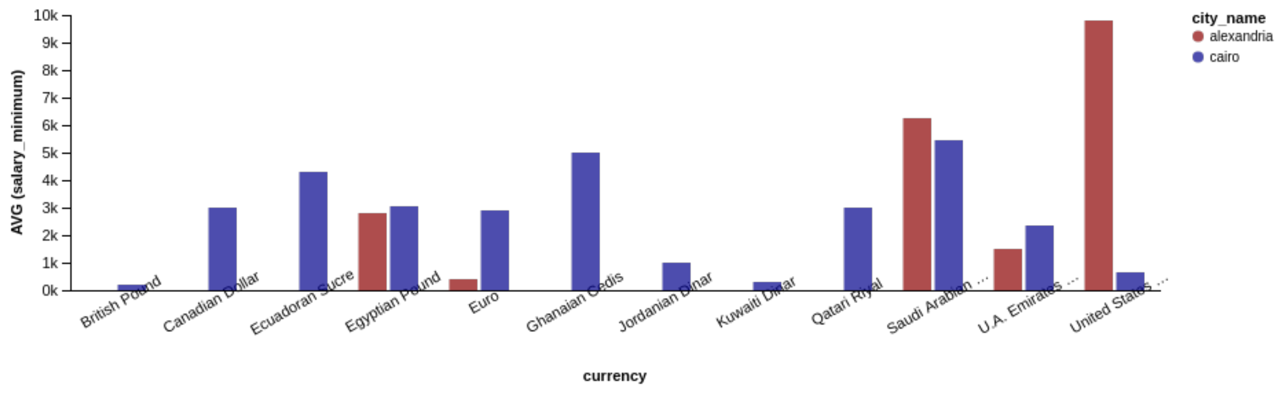


FIGURE 6.1. Average salary minimum vs currency

From the figure we can see that the Egyptian Pound has nearly equal minimum salary in both cities, and so as the Saudi Arabian Riyal and the U.A. Emirates Dirham but with a small disparity between Alexandria and Cairo. The eye catcher here is the United States Dollar which have a very big divergence between Alexandria and Cairo. The average minimum salary of jobs with United States Dollar as the currency in Alexandria is nearly 14 times that of Cairo. Also the average minimum salary of jobs with Euro as its currency of Cairo is way more than that of Alexandria. This can insight us that to think that maybe there are more US companies in Alexandria than in Cairo and the opposite is true for Europan companies.

**Figure 6.2 (a)** and **(b)** came as second and third recommendations and they prove each others.

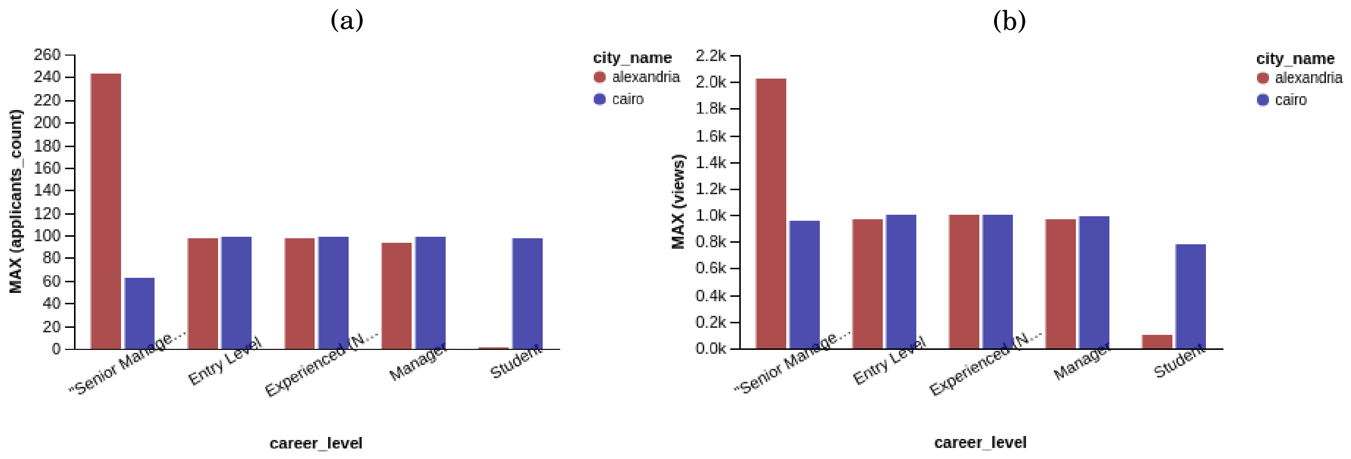


FIGURE 6.2. Maximum applicants count and maximum views vs career level

**Figure 6.2 (a)** visualizes the **career_level** VS. the MAX **applicants_count**, comparing those in Alexandria to Cairo. This means that how the number of people applying for a job is affected by their career level.
While **Figure 6.2 (b)** visualizes the **career_level** VS. the MAX **views**,

comparing those in Alexandria to Cairo. This means that how the number of people viewing a certain job is affected by their career level.
We can see from the two figures that for **Entry Level**, **Manager** and **Experienced Non Manager**, there is no difference between Alexandria and Cairo, whether in the number of people who viewed the job or in the number of people who applied for the job.
The catch here is that for the Senior Management (e.g. VP, CEO), the number of people who viewed the job in Alexandria is way more than those in Cairo as shown **(b)**, consequently the number of people who applied for the job with a Senior Management career level in Alexandria is more than that of Cairo as shown in **(b)**. The opposite is true for student as career level.
We can conclude that Alexandria has more jobs for Senior Management, but Cairo has more jobs for student.

VisDiscovery also recommended **figure 6.3**, which is the **job_categories** VS. MAX (**num_vacancies**), which means the maximum number of job vacancies each category have in both Alexandria compared to Cairo.
And **figure 6.4**, which is the **job_categories** VS. MAX **salary_minimum**, which means the minimum salary each category have in both Alexandria compared to Cairo.
For instance, if we are concerned with **IT/Software Development** career category in the to figures below. We can notice that the number of job vacancies is way more in Cairo compared to Alexandria, the same for the minimum salary. Consequently we can conclude that as the number of vacancies increases the minimum salary increases so that those vacancies could be filled. However, if we are concern with **Business** category, we can easily notice that there few opportunities in both Alexandria and Cairo, and the minimum salary for those in Cairo is way more than those in Alexandria.

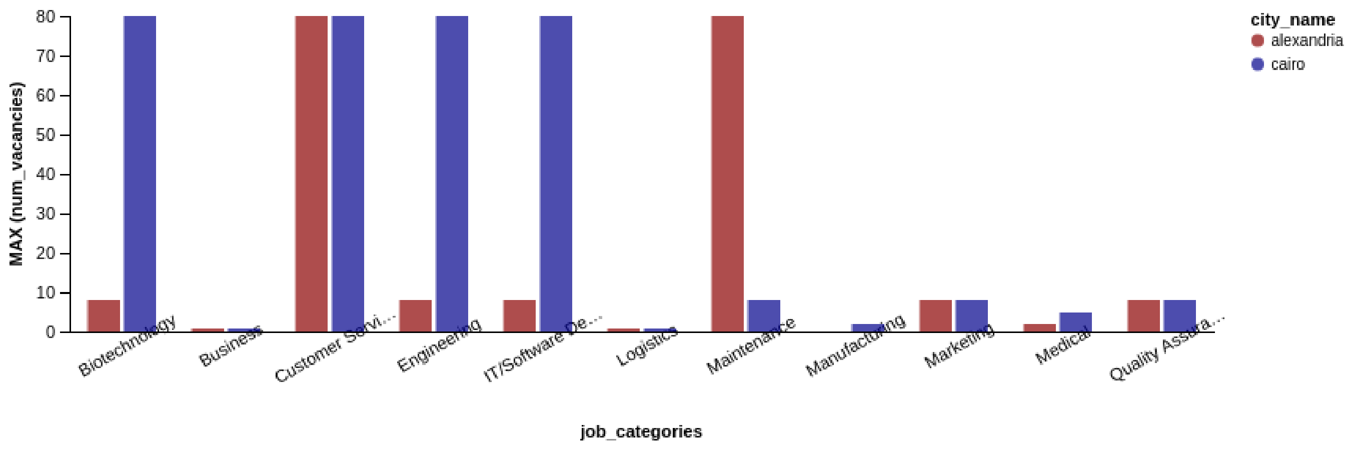


FIGURE 6.3. Maximum number of vacancies vs job categories

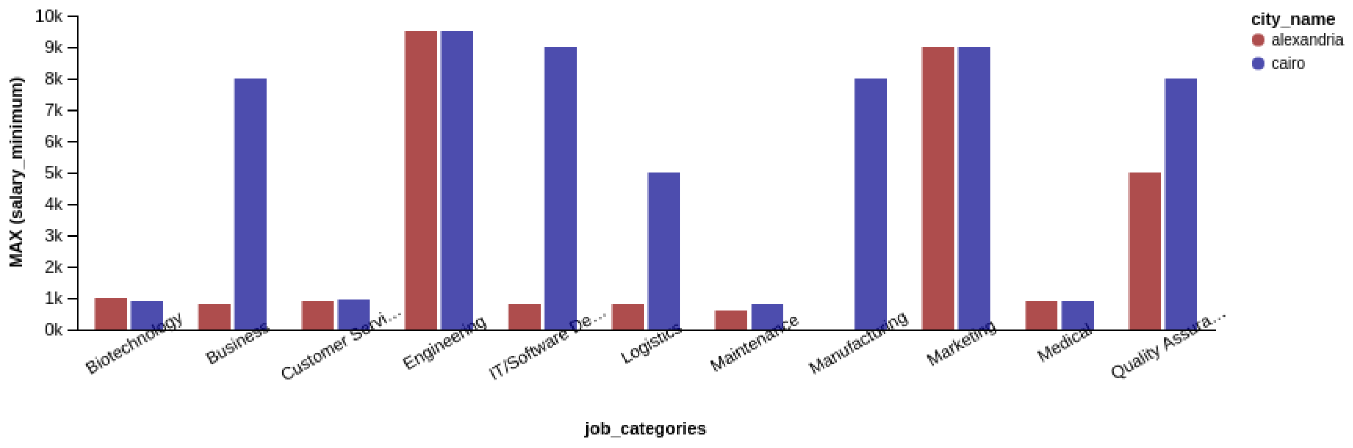


FIGURE 6.4. Maximum salary minimum vs job categories

- Next we want to know how job categories can be affected by various attributes, and see what VisDiscovery would recommend regarding this matter. We will be concern with two categories, **IT/Software Development** and **Engineering**.

The first visualization VisDiscovery recommended is **figure 6.5** , which is **city_name** VS. AVG (**num_vacancies**). Meaning that for each how many average job vacancy each city have for each of our two categories.

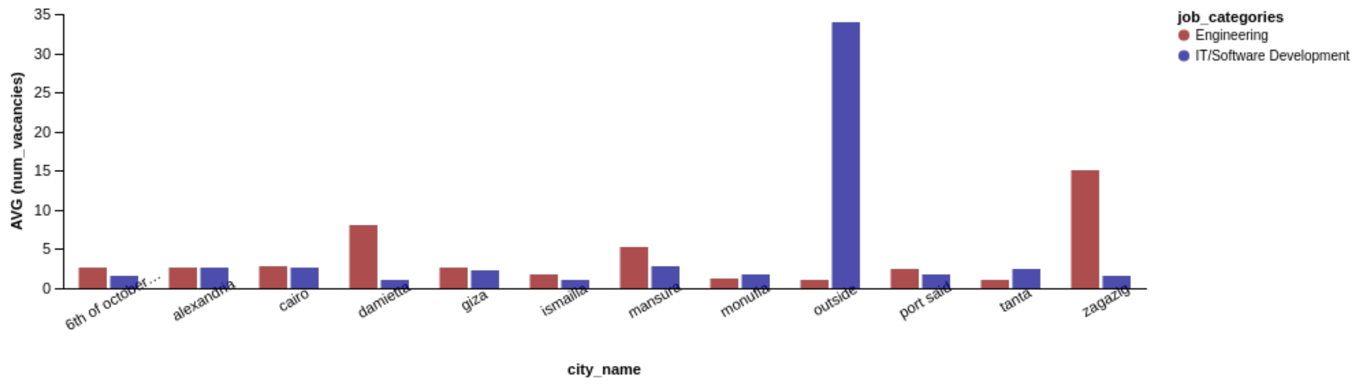


FIGURE 6.5. Average number of vacancies vs city name

We can easily notice from **figure 6.5** that all the cities the average job vacancies are nearly equal for both categories. However, the **outside** city, which means outside Egypt for Software Development is highly deviated from the others. Meaning that there is more job vacancies for Software Developers outside Egypt.

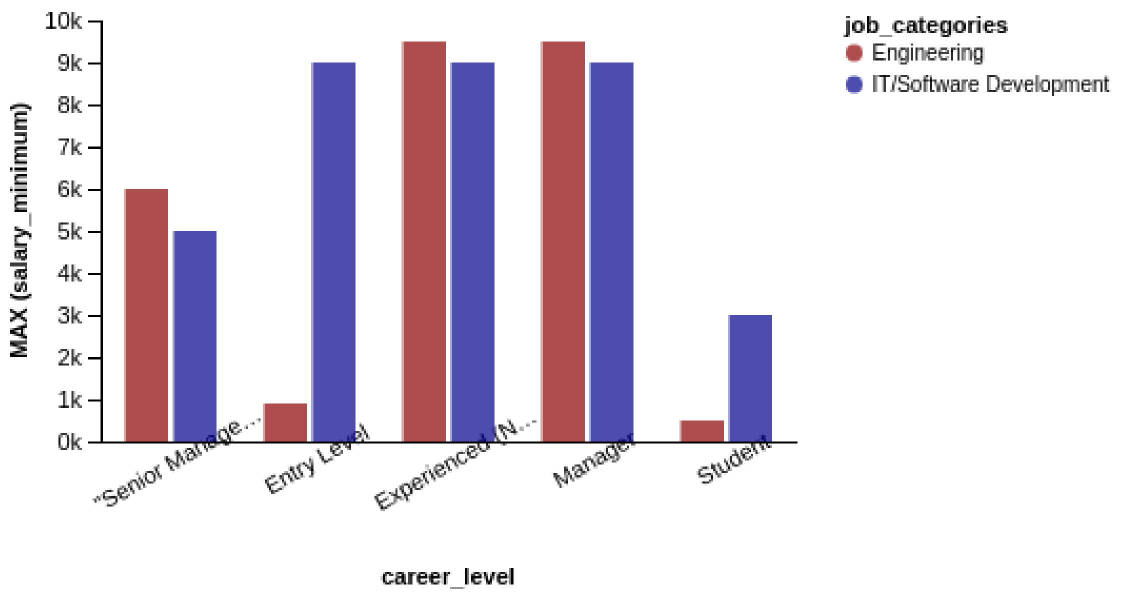


FIGURE 6.6. Maximum salary minimum vs career level

The next recommendation was **figure 6.6**, which visualize the **career_level** on the x-axis, and the MAX(**salary_minimum**) on the y-axis. Which means, for each career level what is the minimum salary, comparing each of our chosen job categories.
From **figure 6.6**, we can notice that the career level Student and the career level Entry Level of Software Development is more than that of Engineering. However, when the career level increases, the minimum salary for the two categories become nearly equal.

## 6.2 Traffic Violations

This dataset contains traffic violation information from all electronic traffic violations issued. Any information that can be used to uniquely identify the vehicle, the vehicle owner or the officer issuing the violation will not be published.

### 6.2.1 Catalog & Data Description

The name of the fields are self explanatory. But here will discuss in details what each field expresses :

- **date_of_stop:** Date of the traffic violation.
- **time_of_stop:** Time of the traffic violation.
- **Agency:** Agency issuing the traffic violation. e.g., **MCP** is Montgomery County Police.
- **SubAgency:** Court code representing the district of assignment of the officer.
- **description:** Text description of the specific charge.
- **location:** Location of the violation, usually an address or intersection.
- **latitude:** Latitude location of the traffic violation.
- **longitude:** Longitude location of the traffic violation.
- **accident:** If traffic violation involved an accident.
- **personal_injury:** If traffic violation involved Personal Injury.
- **property_damage:** If traffic violation involved Property Damage.

- **fatal:** If traffic violation involved a fatality.
- **commercial_license:** If driver holds a Commercial Drivers License.
- **HAZMAT:** If the traffic violation involved hazardous materials.
- **commercial_vehicle:** If the vehicle committing the traffic violation is a commercial vehicle.
- **alcohol:** If the traffic violation included an alcohol related
- **work_zone:** If the traffic violation was in a work zone
- **state:** State issuing the vehicle registration.
- **vehicle_type:** Type of vehicle, e.g., **Automobile**, **Station Wagon**, **Heavy Duty Truck**, etc.
- **make:** Manufacturer of the vehicle, e.g: **Ford**, **Chevrolet**, **Honda**, etc.)
- **Color:** Color of the vehicle.
- **charge:** The specific charge amount.
- **race:** Race of the driver. e.g., **Asian**, **Black**, **White**, **Other**.
- **driver_city:** City of the driver‚Äôs home address.
- **driver_state:** State of the driver‚Äôs home address.
- **DL_state:** State issuing the Driver‚Äôs License.
- **Geolocation:** Geo-coded location information.

### 6.2.2 Data cleaning

After removing the dirty and the null records here are the attributes which we altered in order to make the data-set more useful and easier to operate on:

- **charge:** the charge amount contained some dirty bits and dashes, we removed those dirty bits, e.g., **13-bf** we change it to **13**.
- **Make:** it had some problems as Chevrolet sometimes they named it chevo and sometimes chevro, so we had to clean all those repeated aliases for the same name.
- **SubAgency:** We gave it codes instead of names, e.g., **1st district**=R15, **2nd district**=Rockville B15, **3rd district**=Bethesda SS15, **4th district=**Silver Spring WG15, **5th district**=Wheaton G15, **6th district**=Germantown M15, **Headquarters and Special Operations**= Gaithersburg / Montgomery Village HQ15.

### 6.2.3 Findings

The results this data-set came was as following:

- For instance, if we want to know the effect of **Gender** (comparing Male to Female) on the various attributes of the data-set. VisDiscovery recommended number of visualizations, we will present here the 3 best recommendation we got.

The First recommendation was **figure 6.7**, which visualize the **Driver_state** VS SUM(**Charge**). Meaning that, combining all the drivers in each state, how much charges in US dollars did each state payed, comparing females and males.

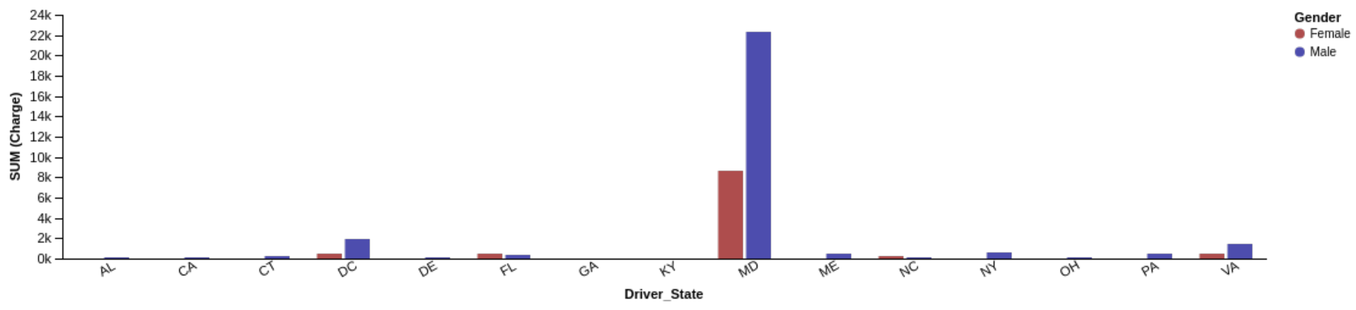


FIGURE 6.7. Sum charge vs driver state

We can easily notice from **figure 6.7** that drivers in Maryland (MD) state are the most drivers paying charges for violating traffic, not only that it also shows that the males in MD pays nearly double the fines that females pay, subsequently meaning that they violate the traffic more than women do.

The second recommendation was **figure 6.8**,which visualize the **race** VS SUM(**Charge**). Meaning that, how much charges in US dollars did each race payed, comparing females and males.

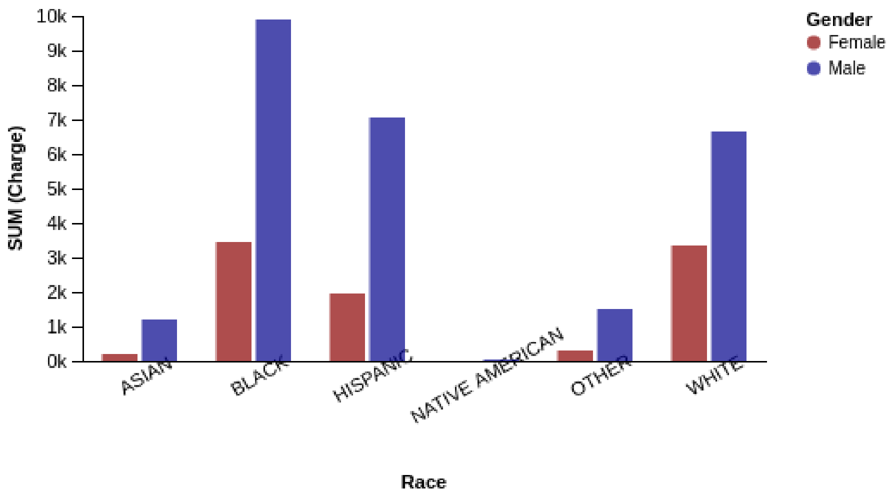


FIGURE 6.8. Sum charge vs race

**Figure 6.8** shows that the sum of charges for all Black males are way more than those for black women, the same also for Hispanics. However for White and Asian the deviation isn't that big compared to Black and Hispanic.

The third recommendation was **figure 6.9**, which visualize **Personal_injury** on the x-axis and SUM (**charge**) on the y-axis.

This visualization compares the male who had personal injuries to the female by summing all the charges they payed in US Dollars.

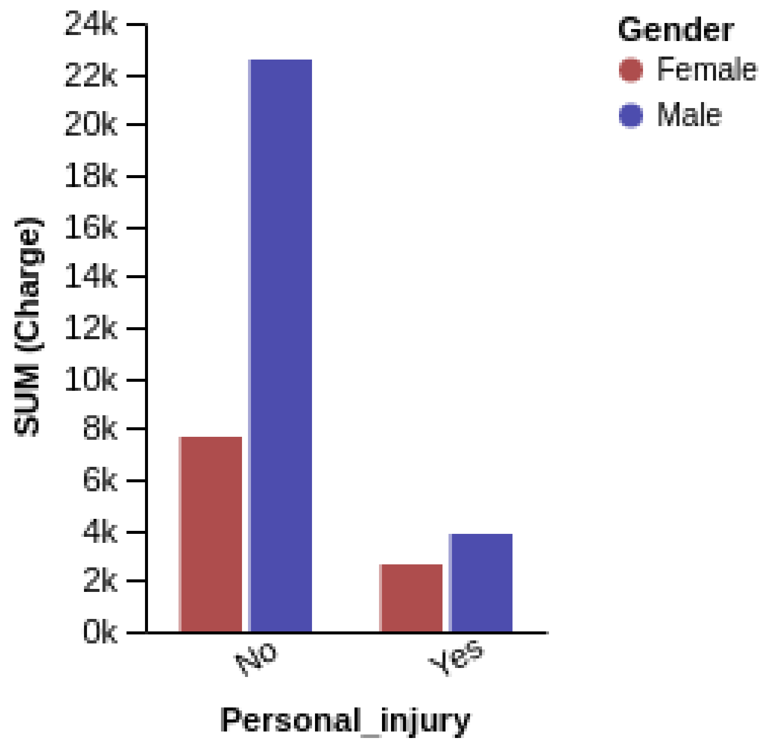


FIGURE 6.9. Sum charge vs personal injury

If we looked at **Figure 6.9**, we will notice that it depicts a property of the data that has proven in figure 6.8 and 6.7, but this visualization emphasizes it more. The chart depicts that although male and female who had been injured in accident nearly pays the same fines, men who had been injured pays more fines that female. Subsequently also meaning that males violate the traffic more than females do.

## 6.3 NYC Yellow Taxi

New York City, being the most populous city in the United States, has a vast and complex transportation system, including one of the largest subway systems in the world and a large fleet of more than 13,000 yellow and green taxis, that have become iconic subjects in photographs and movies.
The subway system digests the lion share of NYC's public transport use, but the 54% of NYC's residents that don't own a car and therefore rely on public transportation still take almost 200 million taxi trips per year!
Data about these taxi trips has been available to the public since 2013, making it a data scientist's dream. We endeavoured into this gold mine using 2.5 years of NYC taxi trip data - around 440 million records - going from January 2013 to June 2015.
The NYC Yellow Taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP).

### 6.3.1 Catalog & Data Description

The Yellow Taxi dataset contains the following fields:

- **Passenger_count:** The number of passengers in the vehicle. This is a driver-entered value.
- **trip_distance:** The elapsed trip distance in miles reported by the taximeter.
- **pickup_longitude:** Longitude where the meter was engaged.

- **pickup_latitude:** Latitude where the meter was engaged.
- **dropoff_longitude:** Longitude where the meter was disengaged.
- **dropoff_latitude:** Latitude where the meter was disengaged.
- **payment_type:** A numeric code signifying how the passenger paid for the trip. e.g., **1**, **2**, **3**, **4**, **5**, **6**.
- **payment_amount:** The total amount the passanger paid for the trip.
- **fare_amount:** The time-and-distance fare calculated by the meter.
- **tip_amount:** Tip amount, this field is automatically populated for credit card tips. Cash tips are not included.
- **tolls_amount:** Total amount of all tolls paid in trip.
- **pickup_datetime:** The date and time when the meter was engaged.
- **dropoff_datetime:** The date and time when the meter was disengaged.

### 6.3.2 Data cleaning

We had to parse the NYC Yellow Taxi records and remove dirty records (e.g. nulls, invalid geographical coordinates, etc.). Also the published dataset had some attributes which we had to change it in oder for the dataset to become more descriptive and easy to be dealt with. We had to introduced some new columns to the dataset based on the previous columns, so we can use them as extra dimensions, some of the previous columns were both used as measure and dimensions, but the following ones, were only used as dimensions:

- **payment_type:** this was available in the first version of the data as integers as we specified before, but after cleaning, and to make the visualization more, e.g, **1**=cash, **2**=credit card, **3**=other,
- **payment_amount_range:** amount of total payment paid is divided into specified ranges to show which sectors has the more variety of records, e.g., <10$ , between 10$ and 50$, etc.
- **tip_amount_range:** amount of tips paid to the driver is divided into specified ranges to show which sectors has the more variety of records, e.g., <10$ , between 10$ and 50$, etc.
- **fare_amount_range:** Amount of fare calculated by the fare counter in the taxi paid is divided into specified ranges to show which sectors has the more variety of records, e.g., <10$ , between 10$ and 50$, etc.
- **taxes_amount_range:** amount of taxes applied on each trip is divided into specified ranges to show which sectors has the more variety of records, e.g., <10$ , between 10$ and 50$, etc.
- **tolls_amount_range:** amount of tolls paid on bridge entrance or some roads is divided into specified ranges to show which sectors has the more variety of records, e.g., <10$ , between 10$ and 50$, etc.
- **trip_length:** trip distance in km is divided into sectors or ranges, to show relations between different trip distances.

### 6.3.3 Findings

In this section the results of the data analysis on NYC Yellow Taxi dataset are presented proofing that VisDiscovery facilitates the analysis by recommending interesting visualizations (those who have the biggest deviation) and discard others visualizations.
Full closure disclaimer: we took a lot of time anlaysing results, and going through hundreds of visualizations, to pick up the most eye catching, or the ones with most non redundant data, as we processed here a total of almost 10 million records (over 10 million with some thousands), and our engine processed the data in two and half minutes without using any optimization techniques. However, when applying shared-based optimization technique the precessing time was reduced to nearly 1 minute and 20 seconds on a single node machine. And it will be reduced more when using multi-node cluster ( using parallel SQL query execution).

- For instance, if we want to know the affect of trip length to the various attributes in the dataset. We are concern with the difference between trip lengths under 10 miles and those between 10 miles and 50 miles.
  VisDiscovery recommended nearly 280 visualizations, hereby we will discuss the first 3 visualizations recommended by VisDiscovery.

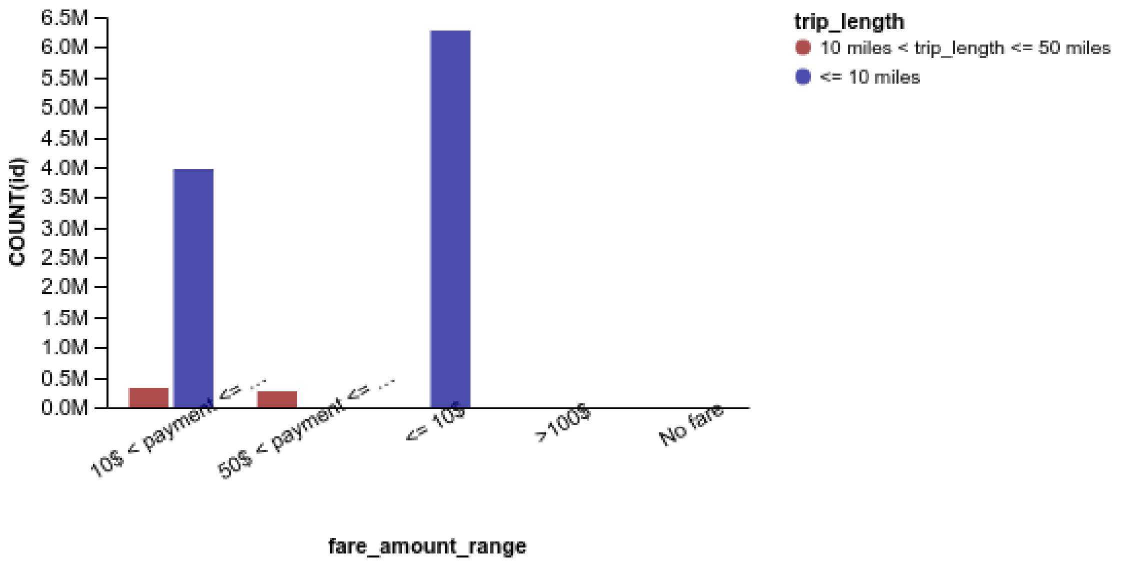


FIGURE 6.10. Count of trips vs fare amount range

The first recommendation, is count of personals vs fare amount range, and this is typically shows, the relation between what most people would pay for a trip. We chose a specific attribute of trip length, which is the range of miles of a trip. The visualization shows that, people who use yellow taxi, would mostly pay under 10$, for a trip that is under 10 miles, and this is true, as NYC average trips will be around 10 miles or so. Comes in second place the trips which exceeeded 10$, with still trips under 10 miles exceeding, which this could be due to long trips during peak times, where there is traffic rush.

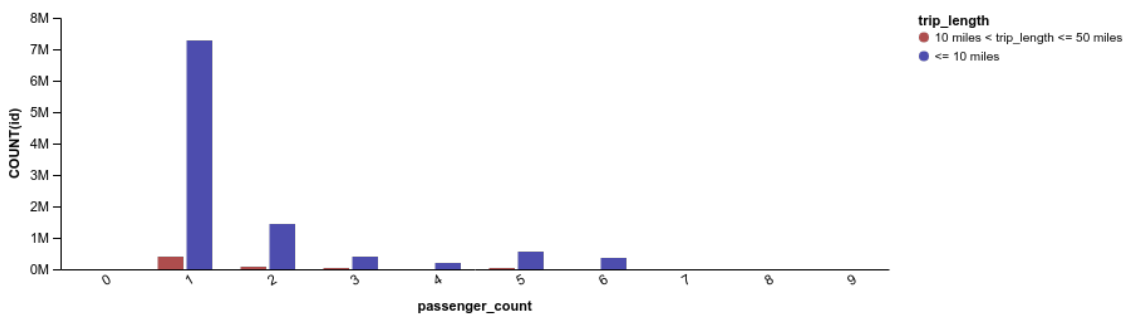


FIGURE 6.11. Count of trips vs passenger count on trips

The Second visualization our engine recommended, was one that is clearly proofs an every day fact. How many times you always notice a taxi with only one passenger ? How many times you yourself take a ride to home from your work, which might not be that long, but with you really tired from work or school, you decide take a cab home, or whenever you are late for or school, so you find the only way to do so is by riding a cab ? Hereby, comes one of the most eye catching visualizations we found, count of trips vs passenger count, comparing trip length of less than 10 miles vs those of between 10 miles and 50 miles, and it shows that the majority goes to those very basic every day trips of one passenger in a less than 10 mile course, totalling almost 7 million trips, and then decreasing dramatically with more number of passengers, which is really frequently we could see in every day trips.

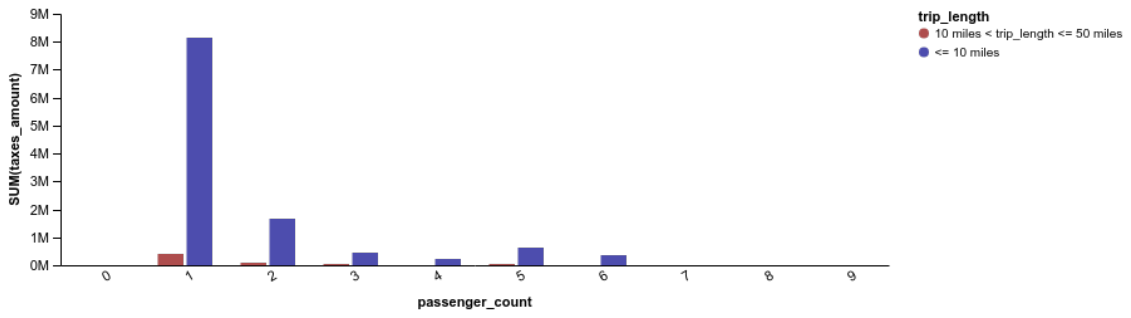


FIGURE 6.12. Sum of taxes amount vs passenger count on trips

The third recommendation our engine recommended and we saw it was really worth mention, and it do proof the theory the previous visualization did say so is, sum of taxes amount on trips vs passenger count, as we said from the previous visualization, it looks like most trips are the trips where there is only one passenger, on a trip less than 10 miles, this is the most common in the urban cities like NYC, we can always see that. These are the most number of trips made, and hence the most taxes would come from such trips. It is clearly spiked in trips with only one passenger and less than 10$, bringing up almost 8 million dollars in total.

CHAPTER 7

# Conclusions & Future Work

## 7.1 Conclusion

This thesis has presented the work done on our graduation project. We present VisDiscovery, a visualization recommendation engine that helps data analysis rapidly identify interesting and useful visualizations using a deviation-based metric. Our engine operates with huge data sizes and takes any semi-structured data as its source and operate on it.
In order to make our system more interactive and reduce the latency of the response we applied two kind of optimizations; Sharing-based Optimizations and Parallel SQL Execution.
VisDiscovery runs on top of Cloudera Impala and Hadoop which allows parallel SQL queries execution and big data storage.
In conclusion, this is an important first step in exploring of visualization recommendation tools, paving the way towards rapid visual data analysis.

## 7.2 Future Work

Many extensions to our engine can be added some are optimization techniques to improve the interactivity of our system more while others are tools to facilitate the usage of our system to the users more.

- **DB Sampling [17]:** One approach for improving the speed of our visualization engine is via data reduction in order to reduce the computational overhead, but this would be at a potential cost in visualization accuracy. Common data reduction techniques, such as uniform and stratified sampling, do not exploit the fact that the sampled tuples will be transformed into a visualization for human consumption. In order to do that we must deliver an algorithm that guarantees high quality visualizations with a small subset of the entire dataset.

- **Indexing [29] [30]:** Even though existing database indexes (e.g., B+Tree) speed up the query execution, they suffer from two main draw- backs: (1) A database index usually yields 5% to 15% additional storage overhead which results in non-ignorable dollar cost in big data scenarios especially when deployed on modern storage devices like Solid State Disk (SSD) or Non-Volatile Memory (NVM). (2) Maintaining a database index incurs high latency because the DBMS has to find and update those index pages affected by the underlying table changes. This introduces Hippo a sparse indexing approach that efficiently and accurately answers database queries while occupying up to two orders of magnitude less storage overhead than de-facto database indexes, i.e., B + -tree. To achieve that, Hippo stores pointers of pages instead of tuples in the indexed table to reduce the storage space occupied by the index. Furthermore, Hippo maintains histograms, which represent the data distribution for one or more pages, as the summaries for these pages. This structure significantly shrinks index storage footprint without compromising much on performance of common analytics queries, i.e., TPC-H workload. Moreover, Hippo achieves about three orders of magnitudes less maintenance overhead compared to the B + -tree. Such performance benefits make Hippo a very promising alternative to index data in big data application scenarios. Furthermore, the simplicity of the proposed structure makes it practical for database systems vendors to adopt Hippo as an alternative indexing technique. In the future, we plan to adapt Hippo to support more complex data types, e.g., spatial data, unstructured data.

- **More Charts Types:** choosing an appropriate type of chart that correctly describes the data is very important to understand the information in these data in an easy and straightforward fashion. Therefor solving the problem of identifying and showing the appropriate type of chart according to some user‚Äôs input so we help them understand what they want from the data easily. For example, box plots and layered charts can provide more statistical information, including variability and trend lines, in addition to the central tendency of a distribution VisDiscovery now uses only one type of charts which is bar charts as recent work has shown that bar charts are the overwhelming majority of visualizations created using visual analytics tools [14].

# Bibliography

[1] *BADR For Information Technology*.
`www.badrit.com`.

[2] *influxData*.
`www.influxdata.com`.

[3] Pat Hanrahan, Christian Chabot, Chris Stolte, *Tableau public*.
`www.tableaupublic.com.`, 2003.

[4] Uci machine learning repository, *U. M. L. Repository*, 2015.

[5] C. Ahlberg, *Spotfire: An information exploration environment*, SIGMOD Rec., 25 (1996), pp. 25–29.

[6] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Commun. ACM, 51 (2008), pp. 107–113.

[7] C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide*, O'Reilly Media, Inc., 1st ed., 2015.

[8] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, *Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals*, Data Min. Knowl. Discov., 1 (1997), pp. 29–53.

[9] Y. Gupta, *Kibana Essentials*, O'Reilly Media, Inc., 1st ed., 2015.

[10] M. HILBERT AND P. LOPEZ, *The World's Technological Capacity to Store, Communicate, and Compute Information*, Science, (2011), pp. 60–65.

[11] S. KANDEL, R. PARIKH, A. PAEPCKE, J. HELLERSTEIN, AND J. HEER, *Profiler: Integrated statistical analysis and visualization for data quality assessment*, in Advanced Visual Interfaces, 2012.

[12] A. KEY, B. HOWE, D. PERRY, AND C. ARAGON, *Vizdeck: Self-organizing dashboards for visual analytics*, in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD '12, New York, NY, USA, 2012, ACM, pp. 681–684.

[13] J. MACKINLAY, *Automating the design of graphical presentations of relational information*, ACM Trans. Graph., 5 (1986), pp. 110–141.

[14] K. MORTON, M. BALAZINSKA, D. GROSSMAN, AND J. MACKINLAY, *Support the data enthusiast: Challenges for next-generation data-analysis systems*, Proc. VLDB Endow., 7 (2014), pp. 453–456.

[15] D. MURRAY, *Tableau Your Data!: Fast and Easy Visual Analysis with Tableau Software*, Wiley Publishing, 1st ed., 2013.

[16] M. P. AND N. E., *A survey of query recommendation techniques for data warehouse exploration*, 2011.

[17] Y. PARK, M. CAFARELLA, AND B. MOZAFARI, *Visualization-aware sampling for very large databases*, in Data Engineering (ICDE), 2016 IEEE 32nd International Conference on, IEEE, 2016, pp. 755–766.

[18] J. POLOWINSKI AND M. VOIGT, *Viso: A shared, formal knowledge base as a foundation for semi-automatic infovis systems*, in CHI

'13 Extended Abstracts on Human Factors in Computing Systems, CHI EA '13, New York, NY, USA, 2013, ACM, pp. 1791–1796.

[19] E. A. RUNDENSTEINER, P. R. DOSHI, AND M. O. WARD, *Prefetching for visual data exploration*, Database Systems for Advanced Applications, International Conference on, 00 (2003), p. 195.

[20] J. RUSSELL, *Getting Started with Impala: Interactive SQL for Apache Hadoop*, O'Reilly Media, Inc., 1st ed., 2014.

[21] A. SATYANARAYAN, D. MORITZ, K. WONGSUPHASAWAT, AND J. HEER, *Vega-lite: A grammar of interactive graphics*, IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), (2017).

[22] T. SEGARAN AND J. HAMMERBACHER, *Beautiful Data: The Stories Behind Elegant Data Solutions*, O'Reilly, 2009.

[23] T. SUNA, *Opentsdb*.
`www.opentsdb.net`.

[24] M. VARTAK, S. RAHMAN, S. MADDEN, A. PARAMESWARAN, AND N. POLYZOTIS, *Seedb: Efficient data-driven visualization recommendations to support visual analytics*, Proc. VLDB Endow., 8 (2015), pp. 2182–2193.

[25] T. WHITE, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 1st ed., 2009.

[26] K. WONGSUPHASAWAT, Z. QU, D. MORITZ, R. CHANG, F. OUK, A. ANAND, J. MACKINLAY, B. HOWE, AND J. HEER, *Voyager 2: Augmenting visual analysis with partial view specifications*, in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, CHI '17, New York, NY, USA, 2017, ACM, pp. 2648–2659.

[27] E. Wu and S. Madden, *Scorpion: Explaining away outliers in aggregate queries*, Proc. VLDB Endow., 6 (2013), pp. 553–564.

[28] M. Xavier, *TIBCO Spotfire for Developers*, Packt Publishing, 2013.

[29] J. Yu, R. Moraffah, and M. Sarwat, *Hippo in action: Scalable indexing of a billion new york city taxi trips and beyond*, in 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017, 2017, pp. 1413–1414.

[30] J. Yu and M. Sarwat, *Hippo: A fast, yet scalable, database indexing approach*, CoRR, abs/1604.03234 (2016).