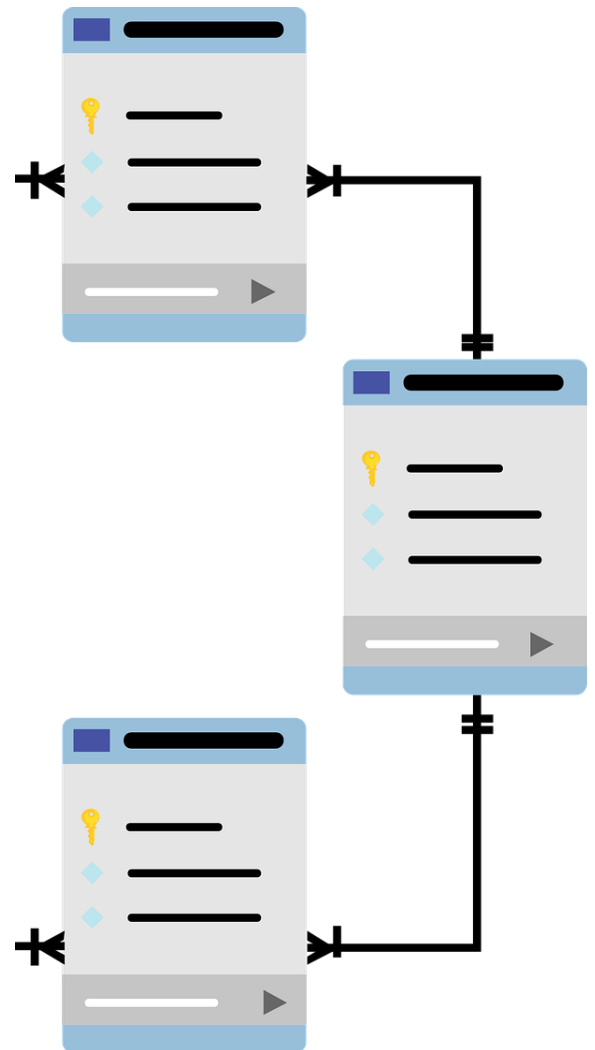


# Beta Reports

## (Relational Database Interface for Business)



---

Authored by:  
Loai Bitawi

---

## Table of Content

### Contents

Table of Content .....	2
Introduction.....	3
Beta Reports .....	5
Database Flexibility.....	7
Table View with additional data.....	7
Descriptive Statistics .....	9
Charts.....	10
Sorting and Aggregations .....	12
Discussions and Conclusions .....	13
Personal Reflection .....	14
References.....	14
Appendix.....	15
Appendix I .....	15
Appendix II.....	39

---

## Introduction

In our daily life, Data is considered crucial success factor for any business, especially large ones. Day by day, data-driven decision making is becoming the main logical stream used for supporting ideas and taking decisions that are built on solid ground.

For these reasons, many software development firms tend to develop tools that help retrieving, analyzing, and reporting data. One of the main skills in this area is Structured Query Language (SQL), which is one of the most powerful interfaces with databases. SQL is an easy-to-learn language, and it simulates regular data requests such as “select employees and their salaries from HR dataset” is translated as “Select Employee, Salary From Database.HR\_Salaries”.

Although this language is useful for IT persons, it is considered rocket science for business and management. Management layers need any information to be given at a blink, thus query language won't be efficient for them. In other words, they care about the total revenue number, but how it is calculated.

For this sake, software developers tend to design useful tools for management that help them retrieve needed information in a fast and easy way.

There are many ways of providing fast insights such as: Reports, Dashboards, and performance monitors.

Reports are useful, customizable, and can deliver deep insights in addition to overview insights. However, reports need a dedicated person to prepare the report and send it to management, which consumes time and effort, in addition to the fact that it is error-prone.

Another tool that solves the mentioned problems in reports is creating a dashboard. Dashboards are predefined sets of figures and charts organized in one single sheet, providing direct insights regarding certain topics. Dashboards are fast, useful, and can be updated at any needed rate. There are many dashboard software that compete in terms of complexity, flexibility, prices, features, ...etc.

Examples of dashboard software include Tableau, Microsoft Power BI, Oracle BI, Qlik. According to (Gartner 2020) in their Magic Quadrant for Analytics and Business Intelligence Platforms report, Power BI is considered the leader of BI software among other rivals. Figure 1 shows Gartner's Magic Quadrant for BI platforms for 2020.



**Figure 1: Magic Quadrant for Analytics and Business Intelligence Platforms**

Microsoft Power Bi provides high flexibility in retrieving data from large set of sources like relational databases, Facebook API's, Twitter API's, semi-structured forms like Json and HTML, ...etc. Moreover, Power Bi provides large set of visualization aids developed to provide the best experience. It also gives ability to use one visual in the dashboard to filter the results of another visual. Currently, Microsoft are improving AI analytics to such dashboards.

Despite all the flexibility in such dashboards, it requires much time and effort to build the dashboard, and changes in entity connections might provide wrong insights. Finally, it needs human efforts to build the dashboard for management, and used by management as static reports.

Finally, performance monitors are useful for near real-time analysis, but unfortunately, the insights are time bounded and changes constantly, so the insights from such tool can't be reported for higher management, or used for strategic decisions.

This project would implement a reporting and visualizing interface that tries to mix between reports and dashboards, in a way that presents flexibility, modularity and ease of use. An advantage of this software is that it only needs an understanding of the domain, the rest is left for the interface to manage.

In the next section, full analysis and representation of the interface would be discussed.

## Beta Reports

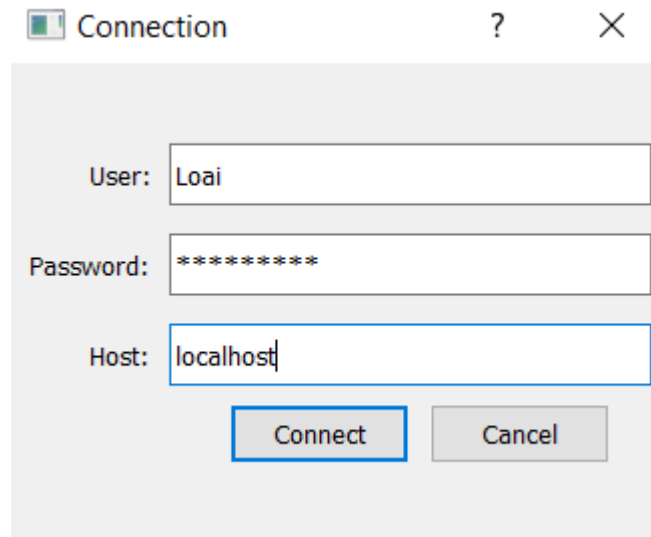
Following to the discussion in the introduction, this project would implement an interface that is a mix between dashboards and custom reports. Beta reports is a flexible reporting tool that can deliver fast insights needed for management and decision making in any business. The interface was designed in a way that is easy to deal with. It simulates management needs and it has a great potential of being developed on a higher scale.

The main features included in this interface:

1. Ability to deploy any database in case it is well-structured in terms of primary and foreign keys.
2. Descriptive Statistics.
3. Table View with additional data.
4. Charts (Line, Bar, Scatter, and pie).
5. Sorting and Aggregations.
6. Table automatic joins up to 3 tables.

Each of these features will be discussed in the following subsections.

When first running the program, the interface will require the connection parameters. The interface won't open until correct parameters are provided and connection to database is established. Figure 2 represents connection window.

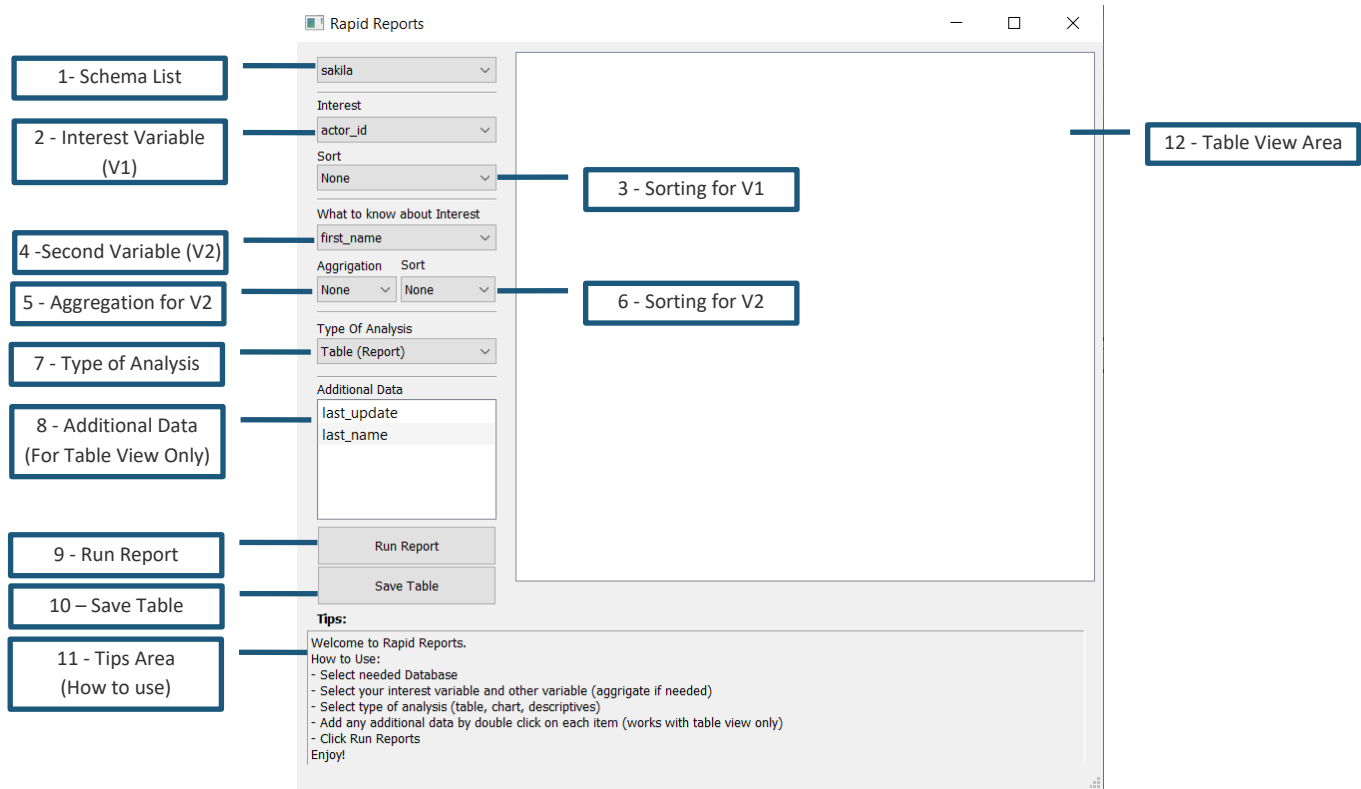


The screenshot shows a window titled "Connection" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are three input fields: "User:" with the text "Loai", "Password:" with masked characters "\*\*\*\*\*", and "Host:" with the text "localhost". Below these fields are two buttons: "Connect" and "Cancel". The "Connect" button is highlighted with a blue border.

*Figure 2 Connection Screen*

The connection screen will provide the user with error messages if fields are empty, or provided parameters aren't correct. Note that if the interface or the machine running the interface aren't capable of reaching the database, the interface will return to the user with a message of incorrect parameters. After successful connection, the connection screen will disappear automatically and the main screen will appear.

Figure 3 shows the main interface of Beta Reports:



**Figure 3 Beta Reports Main interface**

1. Schema List: all the schemas established in the database.
2. Interest Variable: all the column names in the tables of the selected schema.
3. Sorting for V1: sorting the result data Ascending or Descending.
4. Second Variable: all the column names in the tables of the selected schema (excluding selected in V1).
5. Aggregation for V2: grouping V2 results by Sum, Count, Average, Min, and Max.
6. Sorting for V2: sorting the result data Ascending or Descending.
7. Type of Analysis: the shape of the result data from a predefined set (Table view 'Report', Chart, and Descriptive Statistics).
8. Additional Data: for table view only, select the additional columns to add to the report.
9. Run Report: Button used to run the needed analysis, show the charts, or other analysis type.
10. Save Table: Exporting table to Excel file.
11. Tips Area: hint and tips on how to use current selected feature. If charts, it suggests the best data shape that fits the selected chart.
12. Table View Area: a specified area where the result table is shown.
13. Chart Type: Hidden menu that appears when analysis type is chart. The menu provides Line Chart, Pie Chart, Scatter Plot, Bar Chart, and histogram.

---

The interface backend was developed using python, and the front end was also developed using PYQT5 library in python, which provides flexibility in items used in the interface, in addition to emitted signals from each item for control purposes.

The next subsections would introduce the main features of Beta Reports, discuss in details the algorithmic steps of each feature, and the output.

## Database Flexibility

The first feature of this interface is that it is designed to be flexible with any relational database if it is well-structured.

The interface performs the following steps in order to be flexible with any new database regardless any previous knowledge about its structure:

1. Creating a connection with the SQL server, in my case it is MySQL Server.
2. The interface sends the following SQL query that returns all the schemes in the server:  
*SELECT SCHEMA\_NAME AS DATABASE\_NAME FROM INFORMATION\_SCHEMA.SCHEMATA*
3. Gathering all tables in each schema and store them in Data Frame to be used later. The query used to get the tables is:  
*SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE = 'BASE TABLE' AND TABLE\_SCHEMA='SCHEMA NAME'*
4. From the tables data frame, all the variables “Column names” in each table are collected and viewed as the interest list and other variable list. The query used is:  
*"SELECT \* FROM Database\_Name.Table\_Name LIMIT 1"*  
*This query returns a single entry from each table stored in a specific dataframe, then column names are taken and stored as a list*

After this step, no relationships or links between tables are established, but the interface holds mappings for the selected schema, tables in it, and the variables (column names) of each table. This method preserves low usage of Ram and storage since it only holds lists, and any action in the interface constructs its query from inputs and then get the needed dataset.

Note that in order not to send heavy queries just to collect column names, the interface used “*LIMIT 1*” in the query which returns only one row of the table requested.

Moreover, in order not to hold any unused data, steps 3 and 4 are only used for the selected schema. When schema changes, the interface automatically repeats step 3 and 4 for the new schema. In this way, the interface preserves minimum usage of system resources.

## Table View with additional data

This feature allows the user to view the result in tabular form. This might be simple, but due to the modularity of this interface, and the absence of prior knowledge of the database, the feature needs a way to join features from different tables within the schema. The procedure pseudo code is as following:

*Get the tables( $t1, t2$ ), primary key( $pk1, pk2$ ), and foreign key( $fk1, fk2$ ) for  $V1$  and  $V2$ .*

---

```

If t1 = t2: return data from the same table
Elsif v1 in t2: return data from t2
Elsif pk in fk2 or pk1=pk2: return data from joined tables
Elsif fk1 in fk2 or fk1 in pk2: return data from joined tables
Else:
    find_related_tables(t1,t2)
    return data using 3-tables join

```

the function `find_related_tables` searches for a relationship that joins `t1` and `t2` using `t3`. `t3` should hold keys from `t1` and `t2` in order to consider it a joining table. The following simplified psudo code represents how it works:

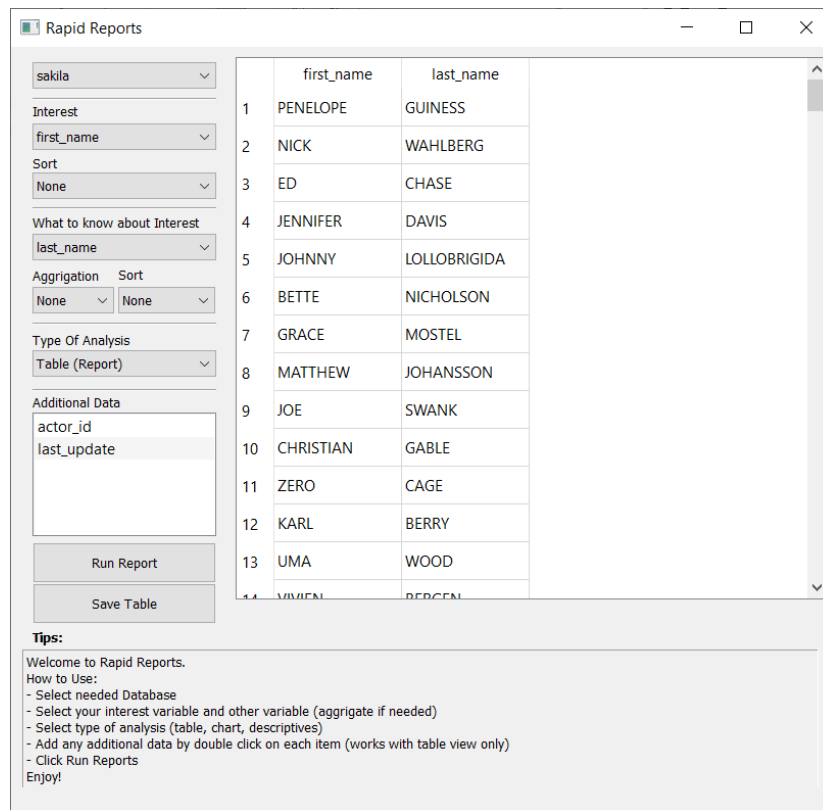
```

For t in schema_tables:
    Keys_table.append([list of keys, t])
#Tables that can connect with V1_ table using any of the keys in V1_table
T1= list all table names where keys_v1 in keys_table
#Tables that can connect with V1_ table using any of the keys in V1_table
T2= list all table names where keys_v2 in keys_table
find table_name where T1=T2
return table_name, matching variable

```

After joining tables and get needed content, “the additional data” area allows users to double click on each field and add it to the table view directly. The additional data are the variables in table 1 and table 2 where `V1` and `V2` exist respectively. Figure 4 shows the table view and additional data selection.





**Figure 4 Table View with Additional Data Selected**

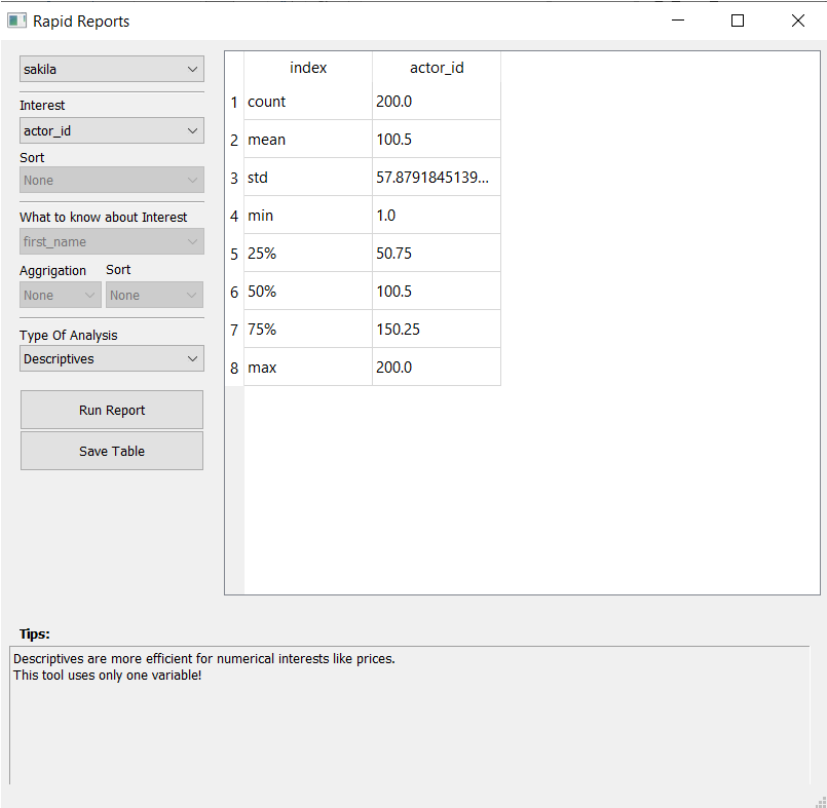
Table feature also supports saving the result to Excel file using Save Table button. This button has a validation that requires the user to run the report before saving it. It provides popup messages when user presses the save button before running the report, it would require him to press the report function. If the run report is pressed before, it will create a directory named “Output” that will have the excel files exported. The directory is C:\Output\Table\_(i).xlsx, where i is a counter that increases each time Save Table is pressed.

## Descriptive Statistics

This feature allows the user to study each variable separately. This feature can be used when choosing descriptive statistics from the analysis type list. The interface will automatically disable all features that are irrelevant to this analysis type. It will display the result in the table view, and the result differs depending on the type of the variable (Qualitative or Quantitative). If the variable is qualitative, it will provide the count of the datapoints, the number of unique values, the top value and its frequency in the dataset. On the other hand, if the variable is quantitative, the feature will return the count of datapoints, mean, standard deviation, minimum value, 25<sup>th</sup>, 50<sup>th</sup>, 75<sup>th</sup> percentiles, and the maximum value. Figure 5 shows the result of this feature.

Descriptive Statistics simply uses the built-in function in python DF.Describe(), which is powerful function that deals with all data types. However, the insights given for the qualitative variables is not

as strong as the ones given for quantitative variables. There is a potential for this interface to develop better statistics for qualitative variables.



**Figure 5 Descriptive Statistics**

Descriptive Statistics also supports Save Table. It will save the descriptive statistics of the selected variable to the excel file.

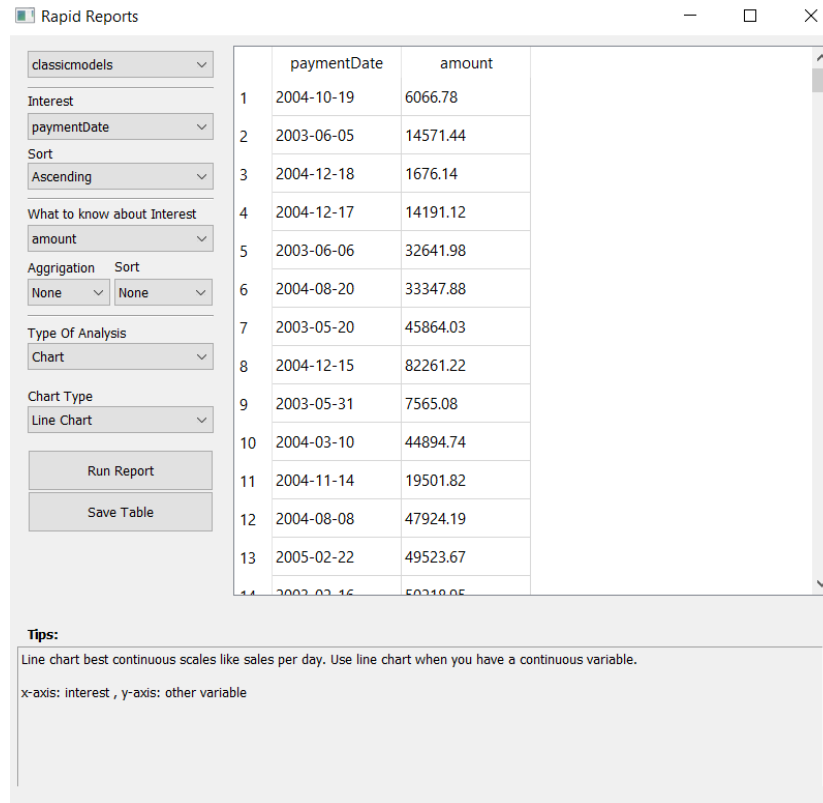
**Charts**

One of the most important tools in analysis in general and in business is charts. Management usually prefers charts since it represents trends in a very easy way to interpret. For example, if the management is trying to see sales trend, the best possible way is to draw a line chart, or if they won't to find the employee of this month then drawing a bar chart is the best. Beta Reports provides different types of plots, and provides hints on when to use a specific chart. For example, if the user selects a line chart, the interface will directly show this tip in the tips area:

*Line chart best continuous scales like sales per day. Use line chart when you have a continuous variable.*

*x-axis: interest, y-axis: other variable*

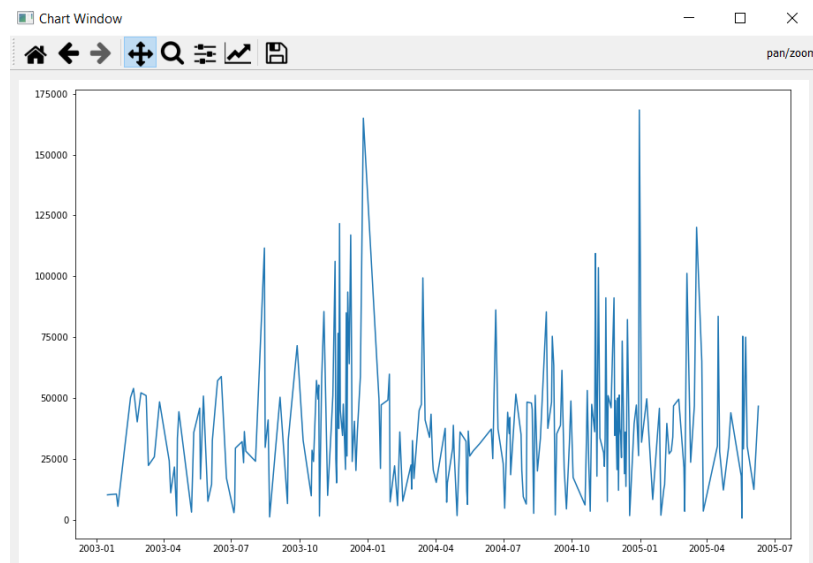
After selecting the chart and its variables, pressing run report will provide the user with the requested chart and in addition, it will show the table view of the output. Figure 6 represents the menu and the output table.



**Figure 6 Chart Menu and Output**

Figure 7 shows the chart requested by the user. The interface provides the user with a toolbar that allows him to do the following:

1. Moving Along Axis
2. Zoom in and out
3. Editing chart (Axis boundaries, colors, ...etc.)
4. Saving the chart



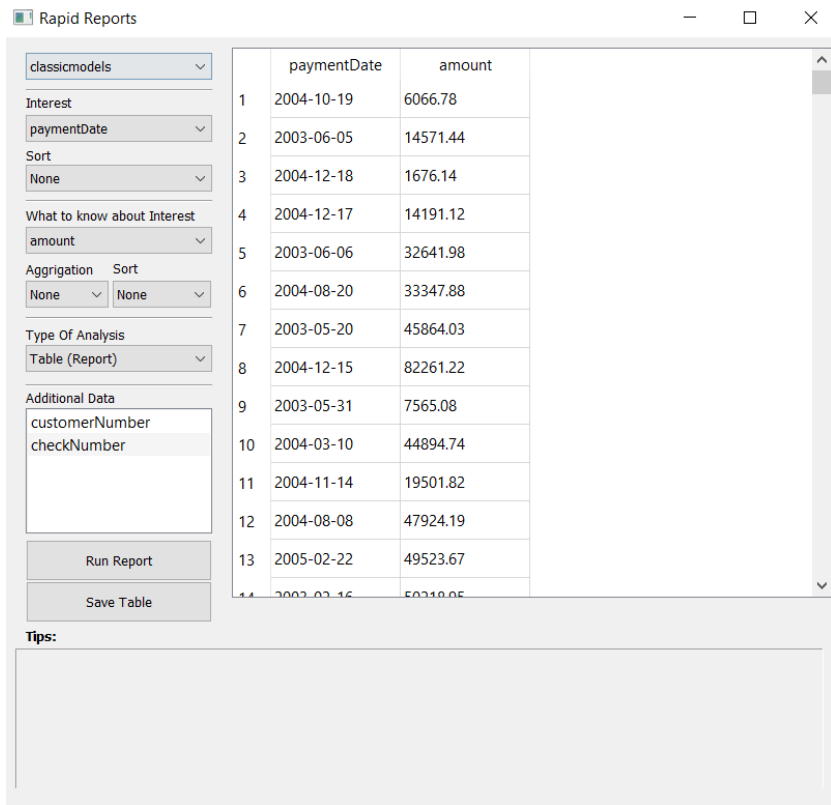
**Figure 7 Chart View**

Although the chart view has many features and flexibilities given to the user, it is only limited to one chart with only two variables. For example, the user can't track the trend of sales per day per branch in one chart, which might be one limitation for Beta Reports in this version. However, the interface is flexible for any improved features.

## Sorting and Aggregations

In addition to representing data in a tabular form without any modifications on the shape of the data, the interface is equipped with tools that enables the user to sort and aggregate the data.

Sorting is used to reorder the results ascending or descending. Moreover, the data can be aggregated by counts, sum, mean, min, and max. A combination of the two features might present better insights. For example, if I have each employee with orders he entered on the system, the regular form is shown in Figure 8.



The screenshot shows the 'Rapid Reports' application window. On the left is a sidebar with various controls:

- classicmodels** (dropdown)
- Interest** (dropdown)
- paymentDate** (dropdown)
- Sort** (dropdown, currently set to 'None')
- What to know about Interest** (dropdown, currently set to 'amount')
- Aggrigation** (dropdown, currently set to 'None')
- Sort** (dropdown, currently set to 'None')
- Type Of Analysis** (dropdown, currently set to 'Table (Report)')
- Additional Data** (text input area containing 'customerNumber' and 'checkNumber')
- Run Report** (button)
- Save Table** (button)

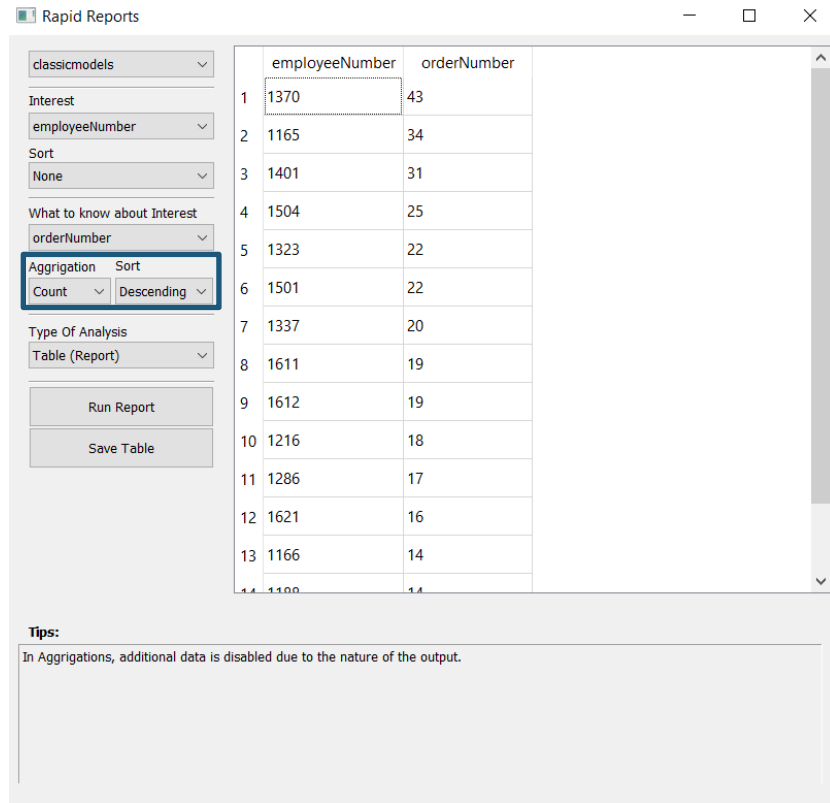
The main area displays a table with the following data:

	paymentDate	amount
1	2004-10-19	6066.78
2	2003-06-05	14571.44
3	2004-12-18	1676.14
4	2004-12-17	14191.12
5	2003-06-06	32641.98
6	2004-08-20	33347.88
7	2003-05-20	45864.03
8	2004-12-15	82261.22
9	2003-05-31	7565.08
10	2004-03-10	44894.74
11	2004-11-14	19501.82
12	2004-08-08	47924.19
13	2005-02-22	49523.67
14	2003-03-16	50310.05

At the bottom of the sidebar, there is a 'Tips:' section with a large empty text area.

**Figure 8 Regular Tabular Form of the Data**

However, if the manager is looking for the employees with highest number of orders, an aggregation on order numbers by count would be done and sorting them descending as shown in Figure 9.



**Figure 9 Using Aggregation and Sorting to Data**

As illustrated in the Figure 9, we can see that beside each employee number, the number of sales orders is aggregated. However, the interface was capable of joining the sales with employee number since there is a direct relationship, but if we try to get the employee name, the interface won't find any relationship because it needs to connect 4 tables together, which is not implemented at this phase. At any rate, such join can be implemented as a future improvement.

## Discussions and Conclusions

Beta Reports provides many features that attracts managers who need to take decisions that are data-driven, or simply monitor performance and gain direct insights.

Any software has its advantages that makes it stand among rivals. Beta Reports stands among its rivals with several advantages:

1. Many flexible ways to view insights (tables, charts and descriptive statistics).
2. Summarized tables using sorting and aggregation.
3. Automatic detection of tables relations (it can provide data from tables with 2<sup>nd</sup> degree relationship).
4. Database Flexibility: Ability to use on any well-structured relational database (no prior setup) .
5. Ability to modify charts and save the result.
6. High potential for improved features.
7. Responsive interface: once item is selected the output and screen layout changes automatically.

---

On the other hand, Beta Reports has some limitations:

1. Summarization and aggregation don't allow adding additional data to the table
2. Automatic relationship detection is considered only between 3 tables, which might limit the interface capability of finding further relationships
3. The interface requires proper definitions for primary and foreign keys. If not, relationships between tables won't be detected
4. Charts are limited to only two variables and one chart

However, the mentioned limitations can be considered as opportunities for improvements, and it is possible to get these improvements easily as an upgrade to the next version. For example, the automatic relationship function can be extended to higher relationship degree, or it can find all the possible relations and take the shortest one. The interface can also be developed to plot more than one plot, since matplotlib library supports subplots, and can support multiple lines to be drawn on the same chart.

On the other hand, the proper definition of the keys in tables is an essential part of the interface. One solution might include finding the columns with no duplicated entries and consider it primary key. However, this solution is risky since multiple columns might have unique entries. Also, this might only apply to primary keys. Foreign keys need further development to be found. However, data interfaces like Microsoft Power BI doesn't detect the keys if they're not defined, which indicates how much this process is complicated.

### **Personal Reflection**

In this project, I've experienced the programming of the user interface, how it's done on the front-end, and the back-end. Moreover, I've experienced the difficulties of retrieving data from the database, and how to manage queries, manipulating SQL and joining data from different tables.

Finally, I enjoyed coding this project and handling different problems and finding the best way to solve it. The project has developed my critical thinking and problem-solving skills.

### **References**

[1] James Richardson, Rita Sallam, Kurt Schlegel, Austin Kronz, Julian Sun 2020, Gartner, accessed 25 December 2020, <https://www.gartner.com/doc/reprints?id=1-1YBTIWVR&ct=200211&st=sb>

---

## Appendix

Appendix I Appendix II contains the plotter function used for charts.

### Appendix I

This section contains the main interface and its functions:

```
import pandas as pd
import os
from PyQt5 import QtCore, QtGui, QtWidgets
from plotter import Ui_Plotter
from connection import Login

class Ui_MainWindow(object):
    def open_window(self, d1, d2):
        self.window=QtWidgets.QMainWindow()
        self.pt= Ui_Plotter()

    self.pt.setupUi(self.window, d1, d2, self.chart_type.currentText())
    self.window.show()

    def setupUi(self, MainWindow, con):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(809, 750)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding,
QtWidgets.QSizePolicy.MinimumExpanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

    sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())

        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setToolButtonStyle(QtCore.Qt.ToolButtonIconOnly)
        MainWindow.setDockNestingEnabled(False)
        MainWindow.setUnifiedTitleAndToolBarOnMac(False)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.DB_list = QtWidgets.QComboBox(self.centralwidget)
        self.DB_list.setGeometry(QtCore.QRect(21, 15, 180, 26))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
```

---

```

        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.DB_list.sizePolicy().hasHeightForWidth())
        self.DB_list.setSizePolicy(sizePolicy)
        self.DB_list.setMinimumSize(QtCore.QSize(180, 26))
        self.DB_list.setMaximumSize(QtCore.QSize(180, 26))
        self.DB_list.setAccessibleName("")

self.DB_list.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
        self.DB_list.setObjectName("DB_list")
        self.Interest_list = QtWidgets.QComboBox(self.centralwidget)
        self.Interest_list.setGeometry(QtCore.QRect(21, 75, 180,
26))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.Interest_list.sizePolicy().hasHeightForWidth())
        self.Interest_list.setSizePolicy(sizePolicy)
        self.Interest_list.setMinimumSize(QtCore.QSize(180, 26))
        self.Interest_list.setMaximumSize(QtCore.QSize(180, 26))
        self.Interest_list.setAccessibleName("")

self.Interest_list.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
        self.Interest_list.setObjectName("Interest_list")
        self.related_list = QtWidgets.QComboBox(self.centralwidget)
        self.related_list.setGeometry(QtCore.QRect(21, 183, 180,
26))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.related_list.sizePolicy().hasHeightForWidth())

```



---

```

tForWidth())
    self.related_list.setSizePolicy(sizePolicy)
    self.related_list.setMinimumSize(QtCore.QSize(180, 26))
    self.related_list.setMaximumSize(QtCore.QSize(180, 26))
    self.related_list.setAccessibleName("")

self.related_list.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
    self.related_list.setObjectName("related_list")
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setGeometry(QtCore.QRect(21, 55, 180, 16))
    self.label.setObjectName("label")
    self.label_2 = QtWidgets.QLabel(self.centralwidget)
    self.label_2.setGeometry(QtCore.QRect(21, 164, 180, 16))
    self.label_2.setObjectName("label_2")
    self.label_3 = QtWidgets.QLabel(self.centralwidget)
    self.label_3.setGeometry(QtCore.QRect(21, 279, 180, 16))
    self.label_3.setObjectName("label_3")
    self.Analysis_type = QtWidgets.QComboBox(self.centralwidget)
    self.Analysis_type.setGeometry(QtCore.QRect(21, 297, 180,
26))
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
    sizePolicy.setHorizontalStretch(0)
    sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.Analysis_type.sizePolicy().hasHeightForWidth())
    self.Analysis_type.setSizePolicy(sizePolicy)
    self.Analysis_type.setAccessibleName("")

self.Analysis_type.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
    self.Analysis_type.setObjectName("Analysis_type")
    self.Analysis_type.addItem("")
    self.Analysis_type.addItem("")
    self.Analysis_type.addItem("")
    self.label_8 = QtWidgets.QLabel(self.centralwidget)
    self.label_8.setGeometry(QtCore.QRect(105, 215, 95, 16))
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,

```

---

```

QtWidgets.QSizePolicy.Fixed)

sizePolicy.setHeightForWidth(self.label_8.sizePolicy().hasHeightForWidth())
    self.label_8.setSizePolicy(sizePolicy)
    self.label_8.setObjectName("label_8")
    self.sorting_related =
QtWidgets.QComboBox(self.centralwidget)
    self.sorting_related.setGeometry(QtCore.QRect(105, 235, 95,
26))
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
    sizePolicy.setHorizontalStretch(0)
    sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.sorting_related.sizePolicy().hasHeightForWidth())
    self.sorting_related.setSizePolicy(sizePolicy)
    self.sorting_related.setAccessibleName("")

self.sorting_related.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
    self.sorting_related.setObjectName("sorting_related")
    self.sorting_related.addItem("")
    self.sorting_related.addItem("")
    self.sorting_related.addItem("")
    self.label_9 = QtWidgets.QLabel(self.centralwidget)
    self.label_9.setGeometry(QtCore.QRect(21, 106, 180, 16))
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)

sizePolicy.setHeightForWidth(self.label_9.sizePolicy().hasHeightForWidth())
    self.label_9.setSizePolicy(sizePolicy)
    self.label_9.setObjectName("label_9")
    self.sorting_interest =
QtWidgets.QComboBox(self.centralwidget)
    self.sorting_interest.setGeometry(QtCore.QRect(21, 123, 180,
26))
    sizePolicy =

```

---

```

QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
    sizePolicy.setHorizontalStretch(0)
    sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.sorting_interest.sizePolicy().hasHeightForWidth())
    self.sorting_interest.setSizePolicy(sizePolicy)
    self.sorting_interest.setAccessibleName("")

self.sorting_interest.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
    self.sorting_interest.setObjectName("sorting_interest")
    self.sorting_interest.addItem("")
    self.sorting_interest.addItem("")
    self.sorting_interest.addItem("")
    self.run_bt = QtWidgets.QPushButton(self.centralwidget)
    self.run_bt.setGeometry(QtCore.QRect(21, 485, 180, 50))
    self.run_bt.setMinimumSize(QtCore.QSize(180, 40))
    self.run_bt.setMaximumSize(QtCore.QSize(180, 40))
    self.run_bt.setObjectName("run_bt")
    self.Save = QtWidgets.QPushButton(self.centralwidget)
    self.Save.setGeometry(QtCore.QRect(21, 525, 180, 50))
    self.Save.setMinimumSize(QtCore.QSize(180, 40))
    self.Save.setMaximumSize(QtCore.QSize(180, 40))
    self.Save.setObjectName("Save Table")
    self.counter=0
    self.gridLayoutWidget =
QtWidgets.QWidget(self.centralwidget)
    self.gridLayoutWidget.setGeometry(QtCore.QRect(220, 10, 581,
531))
    self.gridLayoutWidget.setObjectName("gridLayoutWidget")
    self.gridLayout =
QtWidgets.QGridLayout(self.gridLayoutWidget)
    self.gridLayout.setContentsMargins(0, 0, 0, 0)
    self.gridLayout.setObjectName("gridLayout")
    self.tableWidget =
QtWidgets.QTableWidget(self.gridLayoutWidget)
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Expanding)
    sizePolicy.setHorizontalStretch(0)

```

---

```

        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.tableWidget.sizePolicy().hasHeight
ForWidth())
        self.tableWidget.setSizePolicy(sizePolicy)
        self.tableWidget.setTabletTracking(True)
        self.tableWidget setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.tableWidget setFrameShadow(QtWidgets.QFrame.Sunken)

self.tableWidget.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.A
djustToContents)

self.tableWidget.setEditTriggers(QtWidgets.QAbstractItemView.AllEdit
Triggers)

self.tableWidget.setSelectionMode(QtWidgets.QAbstractItemView.Extend
edSelection)

self.tableWidget.setSelectionBehavior(QtWidgets.QAbstractItemView.Se
lectItems)
        self.tableWidget.setObjectName("tableWidget")
        self.tableWidget.setColumnCount(0)
        self.tableWidget.setRowCount(0)
        self.gridLayout.addWidget(self.tableWidget, 0, 0, 1, 1)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 809, 26))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.actionDB1 = QtWidgets.QAction(MainWindow)
        self.actionDB1.setObjectName("actionDB1")
        self.actionExit = QtWidgets.QAction(MainWindow)
        self.actionExit.setObjectName("actionExit")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(21, 340, 180, 16))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)

```

---

```

        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.label_4.sizePolicy().hasHeightForWidth())
        self.label_4.setSizePolicy(sizePolicy)
        self.label_4.setMinimumSize(QtCore.QSize(180, 16))
        self.label_4.setMaximumSize(QtCore.QSize(180, 16))
        self.label_4.setObjectName("label_4")
        self.chart_type = QtWidgets.QComboBox(self.centralwidget)
        self.chart_type.setGeometry(QtCore.QRect(21, 358, 180, 26))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.chart_type.sizePolicy().hasHeightForWidth())
        self.chart_type.setSizePolicy(sizePolicy)
        self.chart_type.setMinimumSize(QtCore.QSize(180, 26))
        self.chart_type.setMaximumSize(QtCore.QSize(121, 26))

self.chart_type.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
        self.chart_type.setObjectName("chart_type")
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(21, 570, 180, 16))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.label_5.sizePolicy().hasHeightForWidth())
        self.label_5.setSizePolicy(sizePolicy)
        self.label_5.setObjectName("label_5")
        self.notes = QtWidgets.QLabel(self.centralwidget)
        self.notes.setGeometry(QtCore.QRect(11, 590, 780, 150))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)

```

---

```

        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.notes.sizePolicy().hasHeightForWidth())

        self.notes.setSizePolicy(sizePolicy)
        self.notes setFrameShape(QtWidgets.QFrame.Panel)
        self.notes setFrameShadow(QtWidgets.QFrame.Sunken)
        self.notes.setLineWidth(1)
        self.notes.setMidLineWidth(0)
        self.notes.setText("")
        self.notes.setTextFormat(QtCore.Qt.AutoText)

self.notes.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)
        self.notes.setObjectName("notes")
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        self.notes.setWordWrap(True)
        self.label_5.setFont(font)
        self.aggrigation = QtWidgets.QComboBox(self.centralwidget)
        self.aggrigation.setGeometry(QtCore.QRect(21, 235, 80, 26))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.aggrigation.sizePolicy().hasHeightForWidth())
        self.aggrigation.setSizePolicy(sizePolicy)

self.aggrigation.setSizeAdjustPolicy(QtWidgets.QComboBox.AdjustToMinimumContentsLengthWithIcon)
        self.aggrigation.setObjectName("aggrigation")
        self.label_6 = QtWidgets.QLabel(self.centralwidget)
        self.label_6.setGeometry(QtCore.QRect(21, 216, 180, 16))
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)

```

---

```

sizePolicy.setHeightForWidth(self.label_6.sizePolicy().hasHeightForWidth())
    self.label_6.setSizePolicy(sizePolicy)
    self.label_6.setObjectName("label_6")
    self.label_7 = QtWidgets.QLabel(self.centralwidget)
    self.label_7.setGeometry(QtCore.QRect(21, 340, 180, 16))
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)

sizePolicy.setHeightForWidth(self.label_7.sizePolicy().hasHeightForWidth())
    self.label_7.setSizePolicy(sizePolicy)
    self.label_7.setObjectName("label_7")
    self.line = QtWidgets.QFrame(self.centralwidget)
    self.line.setGeometry(QtCore.QRect(21, 50, 180, 3))
    self.line.setFrameShape(QtWidgets.QFrame.HLine)
    self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
    self.line.setObjectName("line")
    self.line_2 = QtWidgets.QFrame(self.centralwidget)
    self.line_2.setGeometry(QtCore.QRect(21, 157, 180, 3))
    self.line_2.setFrameShape(QtWidgets.QFrame.HLine)
    self.line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
    self.line_2.setObjectName("line_2")
    self.line_3 = QtWidgets.QFrame(self.centralwidget)
    self.line_3.setGeometry(QtCore.QRect(21, 270, 180, 3))
    self.line_3.setFrameShape(QtWidgets.QFrame.HLine)
    self.line_3.setFrameShadow(QtWidgets.QFrame.Sunken)
    self.line_3.setObjectName("line_3")
    self.line_4 = QtWidgets.QFrame(self.centralwidget)
    self.line_4.setGeometry(QtCore.QRect(21, 335, 180, 3))
    self.line_4.setFrameShape(QtWidgets.QFrame.HLine)
    self.line_4.setFrameShadow(QtWidgets.QFrame.Sunken)
    self.line_4.setObjectName("line_4")
    self.listView = QtWidgets.QListWidget(self.centralwidget)
    self.listView.setGeometry(QtCore.QRect(21, 358, 180, 121))
    sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
    sizePolicy.setHorizontalStretch(0)
    sizePolicy.setVerticalStretch(0)

```

---

```

sizePolicy.setHeightForWidth(self.listView.sizePolicy().hasHeightForWidth())
    self.listView.setSizePolicy(sizePolicy)
    self.listView.setProperty("showDropIndicator", False)
    self.listView.setAlternatingRowColors(True)

self.listView.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)
    self.listView.setSelectionRectVisible(True)
    self.listView.setObjectName("listView")
    self.retranslateUi(MainWindow)
    QtCore.QMetaObject.connectSlotsByName(MainWindow)
    self.connection=con
    self.databases= pd.read_sql_query("select schema_name as database_name from information_schema.schemata", self.connection)
    self.databases=[self.databases.database_name.iloc[i] for i in range(len(self.databases)) if
self.databases.database_name.iloc[i] not in
['mysql', 'information_schema', 'performance_schema', 'sys']]
    self.label_4.hide()
    self.chart_type.hide()

self.vars=pd.DataFrame(data=None, columns=['variables', 'tables'])
    self.DB_list.currentIndexChanged.connect(self.get_interests)
    for db in self.databases:
        self.DB_list.addItem(db)
    self.DB_list.setCurrentIndex(0)
    self.current_db=self.DB_list.currentText()
    self.tables=pd.read_sql_query("SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA='"+self.current_db+"'", self.connection)
    self.get_interests(0)
    self.get_related_vars(0)
    self.run_bt.clicked.connect(self.analysis)
    self.supported_charts=['Bar Chart', 'Line Chart', 'Scatter Plot', 'Pie Chart', 'Histogram']
    for chart in self.supported_charts:
        self.chart_type.addItem(chart)

self.chart_type.currentIndexChanged.connect(self.special_actions)

self.aggrigation_set=['None', 'Sum', 'Count', 'Average', 'Min', 'Max']

```



---

```

        for s in self.aggrigation_set:
            self.aggrigation.addItem(s)

self.aggrigation.currentIndexChanged.connect(self.special_actions)

self.Analysis_type.currentIndexChanged.connect(self.special_actions)

self.Interest_list.currentIndexChanged.connect(self.interest_changed
)

self.related_list.currentIndexChanged.connect(self.additional)
    self.listView.itemActivated.connect(self.get_add_data)
    self.notes.setText('Welcome to Rapid Reports. \nHow to Use:
\n- Select needed Database \n- Select your interest variable and
other variable (aggrigate if needed) \n- Select type of analysis
(table, chart, descriptives) \n- Add any additional data by double
click on each item (works with table view only) \n- Click Run
Reports \nEnjoy! ')
    self.Save.clicked.connect(self.save_table)
    self.data_table=pd.DataFrame()

def save_table(self,index):
    if self.data_table.shape[0]:
        file = "C:/Output/"
        file_name="Table_"+str(self.counter)+'.xlsx'
        if not os.path.exists(file):
            os.makedirs(file)
        self.data_table.to_excel(file+file_name)
        self.counter+=1
        msg= QtWidgets.QMessageBox()
        msg.setWindowTitle('Notification')
        msg.setText('Data Exported to: \n'+file+file_name)
        msg.setIcon(QtWidgets.QMessageBox.Information)
        x=msg.exec_()
    else:
        msg= QtWidgets.QMessageBox()
        msg.setWindowTitle('Error Message')
        msg.setText('No data to export, Press Run Report first')
        msg.setIcon(QtWidgets.QMessageBox.Information)
        x=msg.exec_()

def interest_changed(self,index):

```

---

```

        self.current_interest=self.Interest_list.currentText()
        self.current_related=self.related_list.currentText()
        self.get_related_vars(0)
        self.additionals(0)

    def additionals(self,index):
        self.current_interest=self.Interest_list.currentText()
        self.current_related=self.related_list.currentText()
        if index == -1:
            self.get_related_vars(0)
        self.current_interest=self.Interest_list.currentText()
        self.current_related=self.related_list.currentText()
        self.t1=list(self.vars[self.vars['variables']==
self.Interest_list.currentText()][ 'tables'])[0]
        self.t2=list(self.vars[self.vars['variables']==
self.related_list.currentText()][ 'tables'])[0]
        self.c1=list(pd.read_sql_query("SHOW COLUMNS FROM
"+self.current_db+"."+self.t1,self.connection)[ 'Field'])
        self.c2=list(pd.read_sql_query("SHOW COLUMNS FROM
"+self.current_db+"."+self.t2,self.connection)[ 'Field'])
        c=[]
        c.extend(self.c1)
        c.extend(self.c2)
        c=set(c)
        if self.current_interest ==self.current_related:
            c.remove(self.current_interest)
        else:
            c.remove(self.current_interest)
            c.remove(self.current_related)
        self.listView.clear()
        for col in c:
            self.listView.addItem(col)

    def get_add_data (self):
        self.current_interest=self.Interest_list.currentText()
        self.current_related=self.related_list.currentText()
        self.get_data(self.current_interest,self.current_related)
        t1=list(self.vars[self.vars['variables']==
self.current_interest][ 'tables'])[0]
        t2=list(self.vars[self.vars['variables']==
self.current_related][ 'tables'])[0]
        c1=list(pd.read_sql_query("SHOW COLUMNS FROM

```

---

```

"+self.current_db+"."+t1,self.connection) ['Field'])
    c2=list(pd.read_sql_query("SHOW COLUMNS FROM
"+self.current_db+"."+t2,self.connection) ['Field'])
    buffer=self.listView.selectedItems()
    item_data=[]
    for item in buffer:
        v=item.text()
        if v in c1: item_data.append([v,t1])
        elif v in c2: item_data.append([v,t2])
        else: item_data.append(['No Data','No Data'])
    item_data.append([self.current_interest,t1])
    item_data.append([self.current_related,t2])
    table_map=pd.DataFrame(data=item_data
,columns=['variable','table'])
    var1=",".join(table_map[table_map['table']==t1]['variable'])
    var2=",".join(table_map[table_map['table']==t2]['variable'])
    data1=pd.read_sql_query("SELECT "+var1+" FROM
"+self.current_db+"."+t1,self.connection)
    data2=pd.read_sql_query("SELECT "+var2+" FROM
"+self.current_db+"."+t2,self.connection)
    data=pd.concat([data1,data2],axis=1)
    data=data.loc[:,~data.columns.duplicated()]
    cols=list(data.columns)
    if self.current_interest ==self.current_related:
        cols.remove(self.current_interest)
    else:
        cols.remove(self.current_interest)
        cols.remove(self.current_related)
    new_cols=[self.current_interest,self.current_related]
    new_cols.extend(cols)
    self.data_table = data[new_cols].drop_duplicates()
    self.sorter()
    self.load_data(self.data_table)

    def get_related_vars(self, index):
        interest=self.Interest_list.currentText()
        if len(self.Interest_list.currentText()):
interest=self.Interest_list.currentText()

interest=self.vars[self.vars['variables']==self.Interest_list.curren
tText()]
        self.related_vars=self.vars.drop(interest.index)

```

---

```

        self.related_list.clear()
        for rv in list(self.related_vars['variables']):
            self.related_list.addItem(rv)
        self.current_related=self.related_list.currentText()
        self.related_list.setCurrentIndex(0)

    def get_interests (self, index):
        self.Interest_list.blockSignals(True)
        self.related_list.blockSignals(True)

self.vars=pd.DataFrame(data=None,columns=['variables','tables'])
        self.current_db=self.DB_list.currentText()
        self.tables=pd.read_sql_query("SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND
TABLE_SCHEMA='"+self.current_db+"'", self.connection)
        col_tables=[]
        cols=[]
        for table in self.tables.iloc[:,0]:
            temp=pd.read_sql_query("SELECT * FROM
"+self.DB_list.currentText()+ "." +table+" LIMIT 1",
self.connection).columns
            cols.extend(temp)
            for i in range(len(temp)):
                col_tables.append(table)
        self.vars['variables']=cols
        self.vars['tables']=col_tables
        self.vars.drop_duplicates(subset=['variables'],inplace=True)
        self.Interest_list.clear()
        interests=list(self.vars['variables'])
        for col in interests:
            self.Interest_list.addItem(col)
        self.Interest_list.setCurrentIndex(0)
        self.get_related_vars(0)
        self.additionals(0)
        self.Interest_list.blockSignals(False)
        self.related_list.blockSignals(False)
        self.current_interest=self.Interest_list.currentText()
        self.current_related=self.related_list.currentText()

    def sorter(self):
        sort_dic={'Ascending':True,'Descending':False,'None':''}
        int_flag=sort_dic[self.sorting_interest.currentText()]

```

---

```

        rel_flag=sort_dic[self.sorting_related.currentText()]
        if self.sorting_interest.currentText() != 'None':

self.data_table=self.data_table.sort_values(by=[self.current_interes
t],ascending=int_flag)
        if self.sorting_related.currentText() != 'None':

self.data_table=self.data_table.sort_values(by=[self.current_related
],ascending=rel_flag)

    def load_data(self, Data):
        self.tableWidget.setRowCount(Data.shape[0])
        self.tableWidget.setColumnCount(Data.shape[1])
        self.tableWidget.setHorizontalHeaderLabels(Data.columns)
        for col in range(Data.shape[1]):
            for row in range(Data.shape[0]):

self.tableWidget.setItem(row,col,QtWidgets.QTableWidgetItem(str(Data
.iloc[row,col])))

    def find_related_tables(self,t1,t2):
        t1_k=pd.read_sql_query(("SHOW KEYS FROM
"+self.current_db+"."+t1), self.connection)

t1_k=t1_k.loc[:,['Column_name']].rename(columns={'Column_name':'key'
})
        t2_k=pd.read_sql_query(("SHOW KEYS FROM
"+self.current_db+"."+t2), self.connection)

t2_k=t2_k.loc[:,['Column_name']].rename(columns={'Column_name':'key'
})
        key_table=pd.DataFrame(data=None, columns=['key','table'])
        for table in self.tables.iloc[:,0]:
            t=pd.read_sql_query(("SHOW KEYS FROM
"+self.current_db+"."+table), self.connection)

t=t.loc[:,['Column_name','Table']].rename(columns={'Column_name':'ke
y','Table':'table'})
            key_table=key_table.append(t,ignore_index=True)

t1_related_tables=key_table[key_table['key'].isin(t1_k['key'])]
```

```

t2_related_tables=key_table[key_table['key'].isin(t2_k['key'])]

t1_related_tables=t1_related_tables[t1_related_tables['table']!=t1].
drop_duplicates(subset=['key'])

t2_related_tables=t2_related_tables[t2_related_tables['table']!=t2].
drop_duplicates(subset=['key'])
    matching_table=
pd.DataFrame(data=None, columns=['key', 'table'])

matching_table=matching_table.append(t1_related_tables[t1_related_tables['table'].isin(t2_related_tables['table'])])

matching_table=matching_table.append(t2_related_tables[t2_related_tables['table'].isin(t1_related_tables['table'])])
    if matching_table.shape[0]==0:
        msg= QtWidgets.QMessageBox()
        msg.setWindowTitle('Error Message')
        msg.setText('No relationship found between variables! \n
Please choose another combination')
        msg.setIcon(QtWidgets.QMessageBox.Information)
        x=msg.exec_()
        return ',',',',''
    else:
        return matching_table.drop_duplicates()

def get_data(self,x,y):
    t1=list(self.vars[self.vars['variables']==x]['tables'])[0]
    pk1=list(pd.read_sql_query(("SHOW KEYS FROM
"+self.DB_list.currentText()+". "+t1+" WHERE Key_name = 'PRIMARY'"),
self.connection)['Column_name'])
    fk1=list(pd.read_sql_query("SHOW KEYS FROM
"+self.DB_list.currentText()+". "+t1+" WHERE Key_name <> 'PRIMARY'",
self.connection)['Column_name'])
    t2=list(self.vars[self.vars['variables']==y]['tables'])[0]
    pk2=list(pd.read_sql_query("SHOW KEYS FROM
"+self.DB_list.currentText()+". "+t2+" WHERE Key_name = 'PRIMARY'",
self.connection)['Column_name'])
    fk2=list(pd.read_sql_query("SHOW KEYS FROM
"+self.DB_list.currentText()+". "+t2+" WHERE Key_name <> 'PRIMARY'",
self.connection)['Column_name'])
    v1=x

```

---

```

        v2=y
        if t1==t2:
            self.data_table=pd.read_sql_query("SELECT "+v1+", "+v2+"
FROM "+self.current_db+"."+t1+"",self.connection).drop_duplicates()
        elif v1 in
list(self.vars[self.vars['tables']==t2]['variables']):
            self.data_table=pd.read_sql_query("SELECT "+v1+", "+v2+"
FROM "+self.current_db+"."+t2+"",self.connection).drop_duplicates()
            elif (pk1[0] in fk2) or (pk1[0]==pk2[0]):
                self.data_table=pd.read_sql_query("SELECT
"+t1+"."+v1+", "+t2+"."+v2+" FROM
"+self.current_db+"."+t1+", "+self.current_db+"."+t2+" WHERE
"+t1+"."+pk1[0]+"="+t2+"."+pk1[0],self.connection).drop_duplicates()
            else:
                #if fk1 is empty or not found in direct relationship
                flag=0
                if len(fk1):
                    for fk in fk1:
                        if flag==0:
                            if (fk in fk2) or (fk in pk2):

self.data_table=pd.read_sql_query("SELECT
"+t1+"."+v1+", "+t2+"."+v2+" FROM
"+self.current_db+"."+t1+", "+self.current_db+"."+t2+" WHERE
"+t1+"."+fk+"="+t2+"."+fk,self.connection).drop_duplicates()
                            flag=1
                        else:
                            t3_d=self.find_related_tables(t1,t2)
                            if t3_d.shape[0]<2:
                                print('no k in r1')
                                pass
                            else:
                                t3=t3_d.iloc[0,1]
                                t1_t3_k=t3_d.iloc[0,0]
                                t2_t3_k=t3_d.iloc[1,0]

self.data_table=pd.read_sql_query("SELECT
"+t1+"."+v1+", "+t2+"."+v2+" FROM "+self.current_db+"."+t1+" ,
"+self.current_db+"."+t2+" , "+self.current_db+"."+t3
                                                                    +"
WHERE
"+self.current_db+"."+t1+"."+t1_t3_k+"="+self.current_db+"."+t3+"."+

```

---

```

t1_t3_k
                                                                 "+"
AND "+"
self.current_db+"."+t3+"."+t2_t3_k+"="+self.current_db+"."+t2+"."+t2
_t3_k , self.connection ).drop_duplicates()
                                flag=1
    else:
        t3_d=self.find_related_tables(t1,t2)
        if t3_d.shape[0]<2:
            print('no k in r1')
            pass
        else:
            t3=t3_d.iloc[0,1]
            t1_t3_k=t3_d.iloc[0,0]
            t2_t3_k=t3_d.iloc[1,0]
            self.data_table=pd.read_sql_query("SELECT
"+t1+"."+v1+", "+t2+"."+v2+" FROM "+self.current_db+"."+t1+" ,
"+self.current_db+"."+t2+" , "+self.current_db+"."+t3
                                                                 "+" WHERE
"+self.current_db+"."+t1+"."+t1_t3_k+"="+self.current_db+"."+t3+"."+
t1_t3_k
                                                                 "+" AND "+"
self.current_db+"."+t3+"."+t2_t3_k+"="+self.current_db+"."+t2+"."+t2
_t3_k , self.connection ).drop_duplicates()

    def table_view (self,x,y):
        for i in reversed(range(self.gridLayout.count())):
            self.gridLayout.itemAt(i).widget().deleteLater()
        self.tableWidget =
QtWidgets.QTableWidget(self.gridLayoutWidget)
        self.tableWidget.setObjectName("tableWidget")
        self.tableWidget.setColumnCount(0)
        self.tableWidget.setRowCount(0)
        self.gridLayout.addWidget(self.tableWidget, 0, 0, 1, 1)
        if len(self.listView.selectedIndexes())>0:
            self.sorter()
            self.load_data(self.data_table)
        elif self.aggrigation.currentText()=='None':
            self.get_data(x,y)
            self.sorter()
            self.load_data(self.data_table)
        else:

```



---

```

        self.agg_functions(x,y)
        self.sorter()
        self.load_data(self.data_table)

def special_actions(self,index):
    self.run_bt.move(21, 485) #move back button to its place
    self.Save.move(21,525)
    self.label_4.hide()
    self.line_4.show()
    self.chart_type.hide()
    self.related_list.setEnabled(True) # Enabling related
    self.aggrigation.setEnabled(True) # Enabling aggrigation
    self.notes.setText('')
    self.listView.show()
    self.label_7.show()
    self.sorting_interest.setEnabled(True)
    self.sorting_related.setEnabled(True)
    self.Save.show()
    if self.Analysis_type.currentText()=='Descriptives':
        self.run_bt.move(21,340)
        self.Save.move(21,380)
        self.aggrigation.setEnabled(False) # disabling
aggrigation
        self.related_list.setEnabled(False) # disabling related
        self.label_7.hide()
        self.line_4.hide()
        self.listView.hide()
        self.sorting_interest.setEnabled(False)
        self.sorting_related.setEnabled(False)
        self.notes.setText('Descriptives are more efficient for
numerical interests like prices. \nThis tool uses only one
variable!')
    elif self.Analysis_type.currentText()=='Chart':
        self.run_bt.move(21,400)
        self.Save.move(21,440)
        self.label_7.hide()
        self.line_4.hide()
        self.listView.hide()
        self.label_4.show()
        self.chart_type.show()
        if self.chart_type.currentText() == 'Histogram':
            self.related_list.setEnabled(False) # disabling

```

---

related

```
        self.aggrigation.setEnabled(False)
        self.notes.setText('Histogram is used to present
distribution and relationships of a single variable over a set of
categories like distribution of grades on a school exam . \n \nThis
tool uses only one variable!')
        elif self.chart_type.currentText() == 'Bar Chart':
            self.notes.setText('Bar chart compare values for
different categories or compare value changes over a period of time
for a single category. \nUse if the number of categories is quite
small,or if one of your data dimensions is time. \n \nx-axis:
interest , y-axis: other variable')
        elif self.chart_type.currentText() == 'Line Chart':
            self.notes.setText('Line chart best continuous
scales like sales per day. Use line chart when you have a continuous
variable. \n \nx-axis: interest , y-axis: other variable')
        elif self.chart_type.currentText() == 'Scatter Plot':
            self.notes.setText("Scatter charts are primarily
used for correlation and distribution analysis. Good for showing the
relationship between two different variables where one correlates to
another (or doesn't). Don't use aggrigations or variables with
categories. \n \nx-axis: interest , y-axis: other variable")
        elif self.chart_type.currentText() == 'Pie Chart':
            self.notes.setText("Pie Chart is useful for
comparing categories. A pie chart typically represents numbers in
percentages, used to visualize a part to whole relationship or a
composition. use aggrigations like count of sold units per sales
point. \n \nx-axis: interest , y-axis: other variable")
        elif self.Analysis_type.currentText() == 'Table (Report)':
            if self.aggrigation.currentText() != 'None':
                self.listView.hide()
                self.label_7.hide()
                self.run_bt.move(21,340)
                self.Save.move(21,380)
                self.notes.setText('In Aggrigations, additional data
is disabled due to the nature of the output.')
            else:
                self.run_bt.move(21, 485)
                self.Save.move(21,525)
                self.label_7.show()
                self.listView.show()
```

---

```

def best_chart(self, x, y):
    if self.aggrigation.currentText() == 'None':
        self.get_data(x, y)
        self.sorter()
        self.load_data(self.data_table)
    else:
        self.agg_functions(x, y)
        self.sorter()
        self.load_data(self.data_table)

self.open_window(list(self.data_table.iloc[:, 0]), list(self.data_table.iloc[:, 1]))

def descriptive (self, x):
    for i in reversed(range(self.gridLayout.count())):
        self.gridLayout.itemAt(i).widget().deleteLater()
    self.tableWidget =
QtWidgets.QTableWidget(self.gridLayoutWidget)
    self.tableWidget.setObjectName("tableWidget")
    self.tableWidget.setColumnCount(0)
    self.tableWidget.setRowCount(0)
    self.gridLayout.addWidget(self.tableWidget, 0, 0, 1, 1)
    t=list(self.vars[self.vars['variables']==x]['tables'])[0]
    data=pd.read_sql_query("SELECT "+x+" FROM
"+self.current_db+"."+t, self.connection)
    self.data_table=data.describe().reset_index()
    self.load_data(self.data_table)

def analysis (self, index):
    self.current_interest=self.Interest_list.currentText()
    self.current_related=self.related_list.currentText()
    if self.Analysis_type.currentText() == 'Table (Report)':

self.table_view(self.current_interest, self.current_related)
        elif self.Analysis_type.currentText() == 'Chart':

self.best_chart(self.current_interest, self.current_related)
        elif self.Analysis_type.currentText() == 'Descriptives':
            self.descriptive(self.current_interest)

def agg_functions (self, interest, related):
    self.get_data(interest, related)

```

---

```

        if self.aggrigation.currentText() == 'Sum':
            agg_data=self.data_table.groupby([interest]).sum()
        elif self.aggrigation.currentText() == 'Count':
            agg_data=self.data_table.groupby([interest]).count()
        elif self.aggrigation.currentText() == 'Average':

agg_data=self.data_table.groupby([interest]).mean().round(4)
        elif self.aggrigation.currentText() == 'Min':
            agg_data=self.data_table.groupby([interest]).min()
        elif self.aggrigation.currentText() == 'Max':
            agg_data=self.data_table.groupby([interest]).max()
        self.data_table=agg_data.reset_index()

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Rapid
Reports"))
        self.DB_list.setCurrentText(_translate("MainWindow", "Select
Table"))
        self.DB_list.setPlaceholderText(_translate("MainWindow",
"Select Table"))
        self.Interest_list.setCurrentText(_translate("MainWindow",
"Select Interest"))

self.Interest_list.setPlaceholderText(_translate("MainWindow",
"Select Interest"))
        self.related_list.setCurrentText(_translate("MainWindow",
"Select Info"))

self.related_list.setPlaceholderText(_translate("MainWindow",
"Select Info"))
        self.label.setText(_translate("MainWindow", "Interest"))
        self.label_2.setText(_translate("MainWindow", "What to know
about Interest"))
        self.label_3.setText(_translate("MainWindow", "Type Of
Analysis"))
        self.Analysis_type.setCurrentText(_translate("MainWindow",
"Select Info"))

self.Analysis_type.setPlaceholderText(_translate("MainWindow",
"Select Info"))
        self.Analysis_type.setItemText(0, _translate("MainWindow",

```

```

"Table (Report)")
    self.Analysis_type.setItemText(1, _translate("MainWindow",
"Chart"))
    self.Analysis_type.setItemText(2, _translate("MainWindow",
"Descriptives"))
    self.run_bt.setText(_translate("MainWindow", "Run Report"))
    self.Save.setText(_translate("MainWindow", "Save Table"))
    self.actionDB1.setText(_translate("MainWindow", "DB1"))
    self.actionExit.setText(_translate("MainWindow", "Exit"))
    self.label_4.setText(_translate("MainWindow", "Chart Type"))
    self.label_5.setText(_translate("MainWindow", "Tips:"))
    self.label_6.setText(_translate("MainWindow",
"Aggrigation"))
    self.label_7.setText(_translate("MainWindow", "Additional
Data"))
    self.label_8.setText(_translate("MainWindow", "Sort"))
    self.label_9.setText(_translate("MainWindow", "Sort"))
    self.tableWidget.setSortingEnabled(True)
    self.sorting_related.setCurrentText(_translate("MainWindow",
"Select Info"))

self.sorting_related.setPlaceholderText(_translate("MainWindow",
"Select Info"))
    self.sorting_related.setItemText(0, _translate("MainWindow",
"None"))
    self.sorting_related.setItemText(1, _translate("MainWindow",
"Ascending"))
    self.sorting_related.setItemText(2, _translate("MainWindow",
"Descending"))

self.sorting_interest.setCurrentText(_translate("MainWindow",
"Select Info"))

self.sorting_interest.setPlaceholderText(_translate("MainWindow",
"Select Info"))
    self.sorting_interest.setItemText(0,
_translate("MainWindow", "None"))
    self.sorting_interest.setItemText(1,
_translate("MainWindow", "Ascending"))
    self.sorting_interest.setItemText(2,
_translate("MainWindow", "Descending"))

```

---

```
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    login= Login()

    if login.exec_() == QtWidgets.QDialog.Accepted:
        MainWindow = QtWidgets.QMainWindow()
        ui = Ui_MainWindow()
        ui.setupUi(MainWindow, login.connection)
        MainWindow.show()
        sys.exit(app.exec_())
    else:
        sys.exit()
```

---

## Appendix II

This section contains the plotter function:

```
from PyQt5 import QtCore, QtGui, QtWidgets
import numpy as np
from matplotlib.backends.qt_compat import QtCore, QtWidgets,
is_pyqt5
from matplotlib.backends.backend_qt5agg import (
    FigureCanvas, NavigationToolbar2QT as NavigationToolbar)
from matplotlib.figure import Figure

class Ui_Plotter(object):

    def setupUi(self, OtherWindow, d1, d2, plot_type):
        OtherWindow.setObjectName("OtherWindow")
        OtherWindow.resize(809, 609)
        OtherWindow.setGeometry(QtCore.QRect(500, 200, 809, 609))
        self.centralwidget=QtWidgets.QWidget(OtherWindow)
        self.centralwidget.setObjectName("centralwidget")
        layout = QtWidgets.QVBoxLayout(self.centralwidget)
        static_canvas = FigureCanvas(Figure())
        OtherWindow.addToolBar(NavigationToolbar(static_canvas,
OtherWindow))
        OtherWindow.setLayout(layout)
        layout.addWidget(static_canvas)
        self._static_ax = static_canvas.figure.subplots()
        if plot_type == 'Bar Chart':
            self._static_ax.bar(d1, d2)
        elif plot_type == 'Line Chart':
            self._static_ax.plot(d1, d2)
        elif plot_type == 'Scatter Plot':
            self._static_ax.scatter(d1, d2)
        elif plot_type == 'Pie Chart':
            self._static_ax.pie(d2, labels=d1, autopct='%1.1f%%')
        elif plot_type == 'Histogram':
            self._static_ax.hist(d1)
        OtherWindow.setCentralWidget(self.centralwidget)
        self.retranslateUi(OtherWindow)
        QtCore.QMetaObject.connectSlotsByName(OtherWindow)

    def retranslateUi(self, OtherWindow):
        _translate = QtCore.QCoreApplication.translate
```

---

```
        OtherWindow.setWindowTitle(_translate("OtherWindow", "Chart  
Window"))
```

```
if __name__ == "__main__":  
    import sys  
    app = QtWidgets.QApplication(sys.argv)  
    OtherWindow = QtWidgets.QMainWindow()  
    ui = Ui_Plotter()  
    ui.setupUi(OtherWindow)  
    OtherWindow.show()  
    sys.exit(app.exec_())
```

### Appendix III

This section contains the code of connection screen:

```
from PyQt5 import QtCore, QtGui, QtWidgets  
import mysql.connector as mysql  
  
class Login(QtWidgets.QDialog):  
    def __init__(self, parent=None):  
        super(Login, self).__init__(parent)  
  
        self.bt_con = QtWidgets.QPushButton(self)  
        self.bt_con.setGeometry(QtCore.QRect(110, 170, 90, 30))  
        self.bt_con.setObjectName("Connection")  
        self.bt_disc = QtWidgets.QPushButton(self)  
        self.bt_disc.setGeometry(QtCore.QRect(210, 170, 90, 30))  
        self.bt_disc.setObjectName("Connection")  
        self.user = QtWidgets.QLineEdit(self)  
        self.user.setGeometry(QtCore.QRect(80, 40, 241, 31))  
        self.user.setObjectName("user")  
        self.label = QtWidgets.QLabel(self)  
        self.label.setGeometry(QtCore.QRect(40, 45, 41, 21))  
        self.label.setObjectName("label")  
        self.label_2 = QtWidgets.QLabel(self)  
        self.label_2.setGeometry(QtCore.QRect(10, 90, 71, 21))  
        self.label_2.setObjectName("label_2")  
        self.password = QtWidgets.QLineEdit(self)  
        self.password.setGeometry(QtCore.QRect(80, 85, 241, 31))  
        self.password.setObjectName("password")  
        self.label_3 = QtWidgets.QLabel(self)
```



---

```

self.label_3.setGeometry(QtCore.QRect(40, 135, 41, 21))
self.label_3.setObjectName("label_3")
self.host = QtWidgets.QLineEdit(self)
self.host.setGeometry(QtCore.QRect(80, 130, 241, 31))
self.setWindowTitle('Connection')
self.host.setObjectName("host")
self.label.setText("User:")
self.label_2.setText("Password:")
self.label_3.setText("Host:")
self.bt_con.setText("Connect")
self.bt_disc.setText("Cancel")

self.bt_con.clicked.connect(self.db_con)
self.bt_disc.clicked.connect(self.cancel)
self.username=''
self.key=''
self.host_db=''
self.connection=''

def cancel(self, index):
    self.reject()
def db_con(self):
    self.username=self.user.text()
    self.key=self.password.text()
    self.host_db=self.host.text()
    if len(self.username) and len(self.key) and
len(self.host_db):
        try:
            self.connection=mysql.connect(
                user=self.username,
                password=self.key,
                host=self.host_db)
            self.accept()
        except:
            msg= QtWidgets.QMessageBox()
            msg.setWindowTitle('Error')
            msg.setText('Incorrect Connection Parameters')
            msg.setIcon(QtWidgets.QMessageBox.Information)
            x=msg.exec_()
    else:
        msg= QtWidgets.QMessageBox()

```

```
msg.setWindowTitle('Error')
msg.setText('Please fill all the fields')
msg.setIcon(QtWidgets.QMessageBox.Information)
x=msg.exec_()
```

## Appendix IV

Attached files:

File Name	Description
Project.py	Main interface body.
Plotter.py	Chart view function.
connection.py	Connection Screen.
Database_sample.sql	SQL dump that contains 4 different schemas as a sample database for the project. This file to be ran on MySQL database if you don't have any other database