## CHAPTER 1: PROBLEM FORMULATION

# Chapter - 1

### 1.1 About the Organization

Hindustan Petroleum Corporation Limited is a Maharatna CPSU and (HPCL) is a Government Company within the meaning of Section 617 of the Companies Act 1956. It is registered with Corporate Identification Number (CIN): L23201MH1952GOI008858

The Registered Office of the Corporation is at Petroleum House, 17 Jamshedji Tata Road, Churchgate, Mumbai 400020.

HPCL is an amalgamation of the erstwhile foreign oil companies ESSO and Caltex, which were taken over by the Government of India in 1974 and 1976 respectively.

HPCL is a Central Public Sector Undertaking, with a subscribed capital of Rs. 339.33 Crores.

The shares are listed on BSE/NSE and are actively traded.

HPCL is one of the largest integrated Public Sector Undertaking, engaged in the business of refining Crude Oil and marketing of various petroleum products like Asphalt, Diesel, Kerosene, LPG, Lube Oils, Petrol, branded products like ATF (Aviation Turbine Fuel), Power, Turbojet, Naphtha, throughout India and at select foreign countries. Some of these products are exported to other countries.

HPCL owns and operates two refineries situated at Mumbai (West coast of India) and Visakhapatnam (East Coast of India)

HPCL has focused on its business throughout India by segmenting its business outlook into Retail(Petrol Pumps), LPG, Industries & Commercial (Bulk Fuels supplies to industries, ships), Lubes, Aviation, Refineries etc, with support from the shared services like Company Secretary, Finance, Human Resources, Legal, Public Relations, etc.

HPCL has 7 Retail and 7 LPG Zonal offices at major Cities, in addition to 90 Regional Offices, 37 major Terminals/Installations/Tap Off Points and an elaborate Pan India

Infrastructure network comprising Aviations Service Facilities (ASF), Auto LPG Pumps, CNG outlets, Depots, LPG Bottling Plants, LPG Import Facilities, Lube Blending Plants, Petrol Pumps, HP Gas LPG Distributors, SKO/LDO Distributors, Lube CFAs, etc.

HPCL is managed by a Board of Directors. The Board of Directors consists of a maximum of 15 Directors, including the Chairman & Managing Director. The Chairman & Managing Director is the head of the Corporation. The Board comprises of Wholetime Directors also called Functional Directors – Director (Marketing), Director (Refineries), Director (Human Resources), Director (Finance).
In addition, part time Directors representing Government of India, through Ministry of Petroleum & Natural Gas, and part time Independent Directors, also called Maharatna Directors are on the Board. All these Directors are nominated by the Government of India.

**1.2 Introduction about the Problem**

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process. It involves summarizing the main characteristics of a dataset, often with visual methods, to uncover patterns, spot anomalies, and test hypotheses. The problem arises because raw data is often messy, unstructured, and filled with noise. Without proper exploration, important insights may be missed, leading to poor decision-making and inaccurate predictive models. EDA helps in understanding the underlying structure of the data, guiding further analysis and ensuring that any conclusions drawn are based on a solid understanding of the data.

**1.3 Present State of the Art**

The current state of the art in EDA includes advanced techniques such as interactive data visualization tools (e.g., Tableau, Power BI), statistical methods, and machine learning algorithms. These tools and methods allow data analysts to delve deeper into datasets, enabling them to explore large and complex datasets efficiently. The integration of artificial intelligence and machine learning has further enhanced EDA, allowing for automated pattern detection and anomaly identification. Moreover, programming languages like Python and R offer powerful libraries (e.g., Pandas, Matplotlib, Seaborn) that facilitate comprehensive data exploration, making EDA more accessible and effective.

### 1.4 Need of Computerization

The need for computerization in EDA arises from the sheer volume and complexity of modern datasets. Manual data analysis is not only time-consuming but also prone to errors. Computerized EDA tools automate the process of data cleaning, transformation, and visualization, making it possible to handle large datasets efficiently. Additionally, computerized tools allow for the application of complex statistical techniques and machine learning algorithms that would be impractical to perform manually. This automation enhances the accuracy and speed of analysis, allowing analysts to focus on interpreting results and making data-driven decisions.

### 1.5 Proposed Software / Project

The proposed project aims to develop an EDA tool that integrates the latest advancements in data analysis and visualization. This tool will be designed to handle diverse datasets, providing users with intuitive interfaces for data exploration. It will include features for data cleaning, transformation, and visualization, along with advanced statistical analysis and machine learning integration. The tool will be userfriendly, catering to both novice and experienced data analysts, and will support a wide range of data formats. By leveraging cutting-edge technologies, this software will streamline the EDA process, making it more efficient and accessible.

### 1.6 Importance of the Work

The importance of this work lies in its potential to revolutionize the way data is explored and understood. In a data-driven world, the ability to quickly and accurately analyze data is crucial for businesses, researchers, and policymakers. The proposed EDA tool will empower users to make better decisions by providing deeper insights into their data. It will also democratize data analysis, making it accessible to those without advanced technical skills. Ultimately, this work will contribute to more informed decision-making across various fields, from finance and healthcare to marketing and engineering.

## CHAPTER2 SYSTEM ANALYSIS

# Chapter - 2

## 2.1 Feasibility Study

### 2.1.1 Technical Feasibility

The technical feasibility of performing Exploratory Data Analysis (EDA) on audit observations using Streamlit and Python is strong. Python is a robust programming language with extensive libraries like Pandas, NumPy, Matplotlib, and K-Means that support data manipulation, analysis, and visualization. Streamlit provides a seamless platform for creating interactive web applications with minimal coding.

### 2.1.2 Economical Feasibility

Economically, the proposed EDA project is feasible as it leverages open-source software (Python, Streamlit, etc.), which eliminates the need for expensive licenses. The development cost is mainly associated with the time and resources needed for coding, testing, and deployment. Given that the project can be executed with existing hardware and software resources, the financial investment is minimal.

### 2.1.3 Operational Feasibility

From an operational standpoint, the project is feasible as it enhances the current workflow of auditors by providing them with powerful tools for data exploration and visualization. The interactive dashboards and real-time data analysis capabilities offered by the Streamlit application will streamline the auditing process, making it more efficient and effective.

### 2.1.4 Other Feasibility Dimensions

Other feasibility dimensions include the scalability and maintainability of the application. The use of Python and Streamlit ensures that the application can be easily scaled to handle larger datasets or additional functionalities as needed. The modular design allows for future enhancements without requiring significant rework. Moreover, the open-source nature of the software components used ensures that the application can be maintained and updated with minimal costs, making it sustainable in the long term.

### 2.2 Analysis Methodology

The analysis methodology for this project involves a structured approach to Exploratory Data Analysis (EDA). It begins with data collection and preparation, including data cleaning and transformation to ensure the dataset is suitable for analysis. The next step involves performing EDA using descriptive statistics and data visualization techniques to summarize and understand the data. This includes calculating summary statistics, visualizing data distributions, and identifying patterns and anomalies. Finally, the insights gained from EDA are integrated into the Streamlit application, where interactive dashboards and visualizations are built for real-time data exploration and decisionmaking.

## 2.3 Choice of the Platforms

### 2.3.1 Software Used

The primary software used for this project includes:

- **Python**: The core programming language used for data manipulation, analysis, and application development.
- **Pandas**: A Python library for data manipulation and analysis, used extensively for handling and processing datasets.
- **NumPy**: A library for numerical computations, used for handling large arrays and matrices of data.
- **Matplotlib and Plotly**: Visualization libraries in Python used for creating static plots, graphs, and charts.
- **Streamlit**: A Python library used to create interactive web applications quickly and efficiently, enabling users to visualize and interact with data in real-time.

### 2.3.2 Hardware Used

The hardware requirements for this project are minimal. A standard computer with the following specifications will suffice:

- **Processor**: A multi-core processor (e.g., Intel i5 or equivalent) to handle the computational tasks efficiently.
- **RAM**: At least 8GB of RAM to manage data processing and running multiple applications simultaneously.
- **Storage**: Sufficient storage (preferably SSD) to store datasets and application files, with a recommended capacity of at least 256GB.
- **Internet Connectivity**: Reliable internet access for downloading software libraries, accessing data sources, and deploying the Streamlit application on a web server or cloud platform.

## CHAPTER 3 : SYSTEM DESIGN

# Chapter – 3

### 3.1 Design Methodology

The design methodology for the Exploratory Data Analysis (EDA) application using Streamlit and Python follows an iterative and modular approach. This approach ensures that each component of the system is developed, tested, and refined in stages, allowing for continuous improvement and adaptability. The key steps in the design methodology include:

1. **Requirement Analysis**: Identifying and documenting the functional and nonfunctional requirements of the EDA application, focusing on user needs, data handling, and visualization capabilities.

2. **System Architecture Design**: Creating a high-level architecture that outlines the major components of the system, including the data processing pipeline, the Streamlit interface, and the database.

3. **Component Design**: Breaking down the system into smaller modules, such as data collection, data cleaning, visualization, and user interface design. Each module is designed with specific input, processing, and output requirements.

4. **Prototyping**: Developing a basic prototype of the application using Python and Streamlit to validate design concepts and gather user feedback.

5. **Iterative Refinement**: Continuously refining the design based on user feedback and testing, ensuring that the final product meets all requirements and performs optimally.

6. **Final Implementation**: Integrating all modules into a cohesive system, followed by extensive testing and validation to ensure the application functions as intended.

### 3.2 Database Design

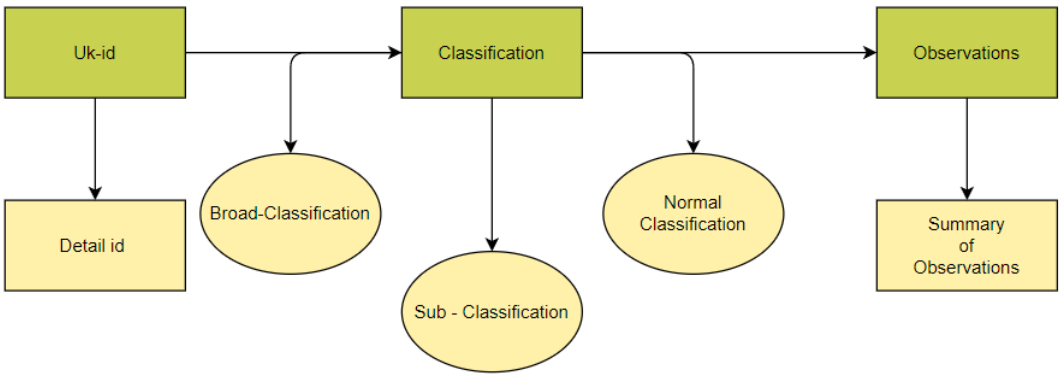
The database design for the EDA application focuses on efficiently storing, retrieving, and managing audit observation data. The design includes both logical and physical aspects, ensuring that the data is structured in a way that supports quick access and analysis.

### 3.2.1 Entity-Relationship Diagram (ERD)

The Entity-Relationship Diagram (ERD) represents the data model of the EDA application, illustrating how data entities are related within the system. Key entities might include **AuditObservations**, **Classifications**, **IDs, etc**. The ERD defines the relationships between these entities, such as:

- **One-to-Many**
- **One-to-One**

Attributes for each entity, such as **UK-ID**, **Detail-ID**, **Date**, **ObservationDetails**, and are also defined in the ERD.



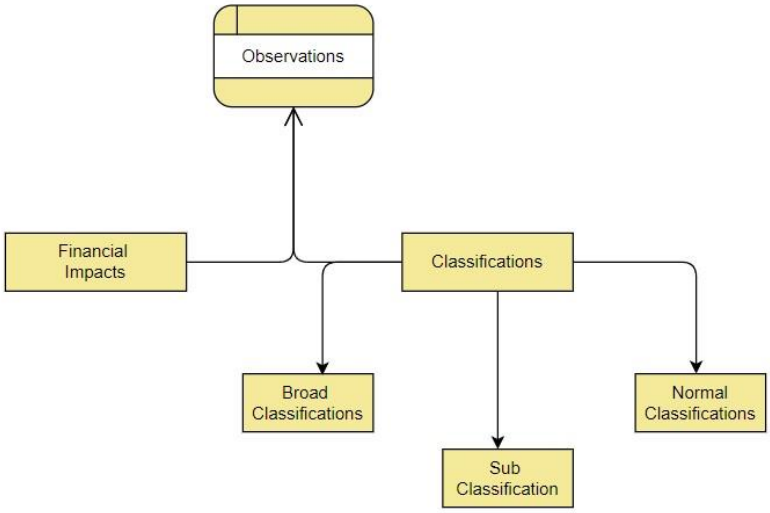

**3.1.2.1**

### 3.2.2 Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) illustrates the flow of information within the EDA application, showing how data moves from input to output through various processes. Key processes include:

- **Data Input**: Capturing raw audit observations from various sources, such as databases or spreadsheets.

- **Data Cleaning**: Processes that remove duplicates, handle missing values, and standardize data formats.
- **Data Storage**: Storing cleaned and transformed data in the database.
- **Data Analysis**: Processes that perform EDA, including descriptive statistics, visualization, and pattern detection.
- **Data Output**: Generating reports and visualizations, which are then presented to users through the Streamlit interface.




**3.2.2.1**

### 3.3 Input Design

The input design for the EDA application focuses on creating intuitive and user-friendly forms and interfaces that allow users to enter and interact with data efficiently. The design considers the following aspects:

1. **Data Entry Forms**: Developing forms within the Streamlit application for users to input audit observations and related data. These forms include fields for entering text, selecting dates, choosing from dropdown menus, and uploading files.
2. **Validation Checks**: Implementing validation rules to ensure that the data entered by users is accurate and complete. This includes checks for mandatory fields, data type

validation (e.g., ensuring that dates are entered in the correct format), and logical validation (e.g., ensuring that observation dates are not in the future).

3. **Interactive Inputs**: Designing interactive components such as sliders, checkboxes, and radio buttons that allow users to filter and sort data dynamically. For example, users might adjust a slider to view audit observations within a specific date range or use checkboxes to filter observations by department.

4. **User Feedback**: Providing immediate feedback to users during data entry, such as error messages for invalid input or confirmation messages for successful submissions. This ensures a smooth user experience and reduces the likelihood of data entry errors.
5. **Data Integration**: Ensuring that the data entered through the input forms is seamlessly integrated into the database, allowing for real-time analysis and visualization within the Streamlit application.

**CHAPTER 4 : TESTING AND IMPLEMENTATION**

# Chapter - 4

### 4.1 Working on Dataset :-

### 4.1.1 Loading The Dataset

```
table1 = pd.read_csv('observations.csv')
```

```
table2 = pd.read_csv('parent_audit_reports.csv')
```

### Merge the Two Tables

```
import pandas as pd

# Read the csv files
table1 = pd.read_csv('observations.csv')
table2 = pd.read_csv('parent_audit_reports.csv')

# Merge the tables on the 'id' column
merged_table = pd.merge(table1, table2, on='uk_id')

# Print the merged table
print(merged_table)
```

### Reading and Describing the dataset

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | detail_id | broad_classificat | financial_impact | icfr_category | nature | observation_no | risk_category | sub_classificatio | summary_of_obs | uk_id | classification |
| 2 | 4,291 | Governance & C | | Inventory | Repetitive | 1 | Acceptable | Non-compliance | Delay in Operati | 10,340 | Non-compliance |
| 3 | 4,293 | Operational Effic | | Inventory | Repetitive | 2 | Acceptable | Abnormal Gain/I | Gain/Loss entrie | 10,340 | Inventory Manag |
| 4 | 4,296 | Financial Manag | 20.85 lakhs | Inventory | Unique | 3 | Acceptable | Own Consumpti | HSD excess proc | 10,340 | Accuracy and rel |
| 5 | 4,298 | Governance & C | | Inventory | Repetitive | 4 | Acceptable | Lack of Control i | Open RTGP | 10,340 | Non-compliance |
| 6 | 4,300 | Financial Manag | | Expenditure | Repetitive | 5 | Acceptable | Related to PCD | Review of PCD | 10,340 | Accuracy and rel |
| 7 | 4,303 | Governance & C | | Inventory | Repetitive | 6 | Acceptable | Non-compliance | Open SOP Debit | 10,340 | Non-compliance |
| 8 | 4,304 | Procurement an | | Expenditure | Repetitive | 7 | Acceptable | Open Purchase ( | Open Purchase ( | 10,340 | Identification of |
| 9 | 4,306 | Governance & C | 3.36 lakhs | Revenue | Repetitive | 8 | Acceptable | Non-compliance | Pending recover | 10,340 | Non-compliance |
| 10 | 4,307 | Governance & C | 2.01 lakhs | Revenue | Repetitive | 9 | Acceptable | Non-compliance | Pending recover | 10,340 | Non-compliance |
| 11 | 4,308 | Governance & C | | Inventory | Repetitive | 10 | Acceptable | Approvals for Pi | Approval for Pip | 10,340 | Non-compliance |

### 4.1.2 Information of Table

```
table1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194 entries, 0 to 193
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   detail_id               194 non-null    int64 
 1   broad_classification    194 non-null    object
 2   financial_impact        62 non-null     object
 3   icfr_category           194 non-null    object
 4   nature                  194 non-null    object
 5   observation_no          194 non-null    int64 
 6   risk_category           194 non-null    object
 7   sub_classification      194 non-null    object
 8   summary_of_observation  194 non-null    object
 9   uk_id                   194 non-null    int64 
 10  classification          194 non-null    object
dtypes: int64(3), object(8)
memory usage: 16.8+ KB
```

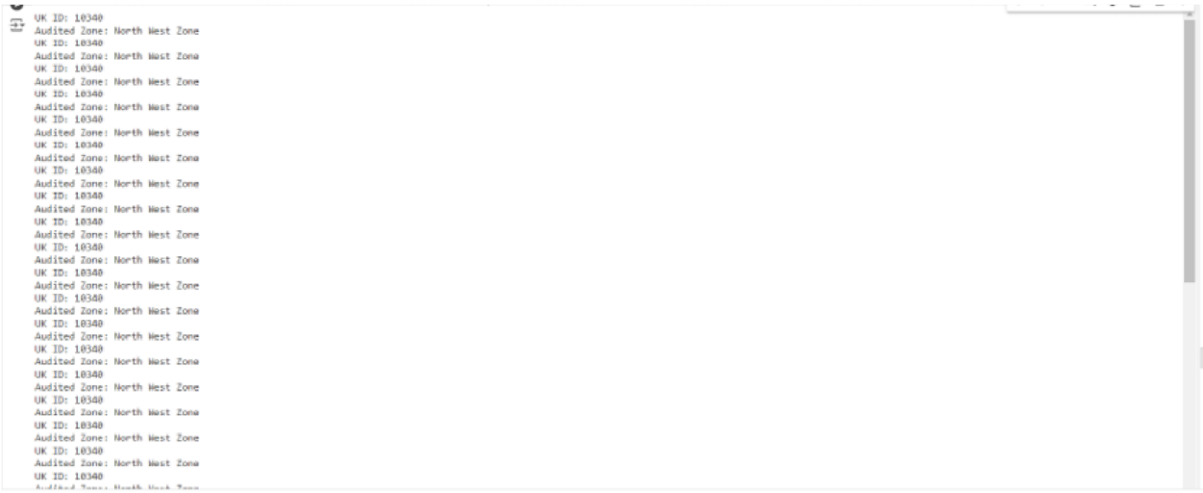
### Information of Dataframe

```
[ ]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231 entries, 0 to 230
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   detail_id               194 non-null    float64
 1   broad_classification    194 non-null    object 
 2   financial_impact        62 non-null     object 
 3   icfr_category           194 non-null    object 
 4   nature                  194 non-null    object 
 5   observation_no          194 non-null    float64
 6   risk_category           194 non-null    object 
 7   sub_classification      194 non-null    object 
 8   summary_of_observation  194 non-null    object 
 9   uk_id                   231 non-null    int64  
 10  classification          194 non-null    object 
 11  audit_period_end        37 non-null     object 
 12  audit_period_start      37 non-null     object 
 13  audit_team_zone         37 non-null     object 
 14  audited_zone            37 non-null     object 
 15  location                37 non-null     object 
 16  sbu                     37 non-null     object 
dtypes: float64(2), int64(1), object(14)
memory usage: 30.8+ KB
```

**Displaying the UK ID based on Particular North West Zone**

```
import pandas as pd

# Load the data from the two CSV files, verify the file paths
table1 = pd.read_csv('observations.csv') # Update with the correct file path
table2 = pd.read_csv('parent_audit_reports.csv') # Update with the correct file path

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Filter the data to only include the North West zone
north_west_data = merged_df[merged_df['audited_zone'].str.contains('North West')]

# Display each sub-classification in the 'audited_zone' column
for index, row in north_west_data.iterrows():
    print("UK ID:", row['uk_id'])
    print("Audited Zone:", row['audited_zone'])
```



**Displaying the UK ID based on Particular SouthZone**

```
import pandas as pd

# Load the data from the two CSV files, verify the file paths
table1 = pd.read_csv('observations.csv') # Update with the correct file path
table2 = pd.read_csv('parent_audit_reports.csv') # Update with the correct file path

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Filter the data to only include the North West zone
north_west_data = merged_df[merged_df['audited_zone'].str.contains('South Zone')]

# Display each sub-classification in the 'audited_zone' column
for index, row in north_west_data.iterrows():
    print("UK ID:", row['uk_id'])
    print("Audited Zone:", row['audited_zone'])
```

```
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10490
Audited Zone: South Zone
UK ID: 10495
Audited Zone: South Zone
UK ID: 10495
Audited Zone: South Zone
UK ID: 10495
```

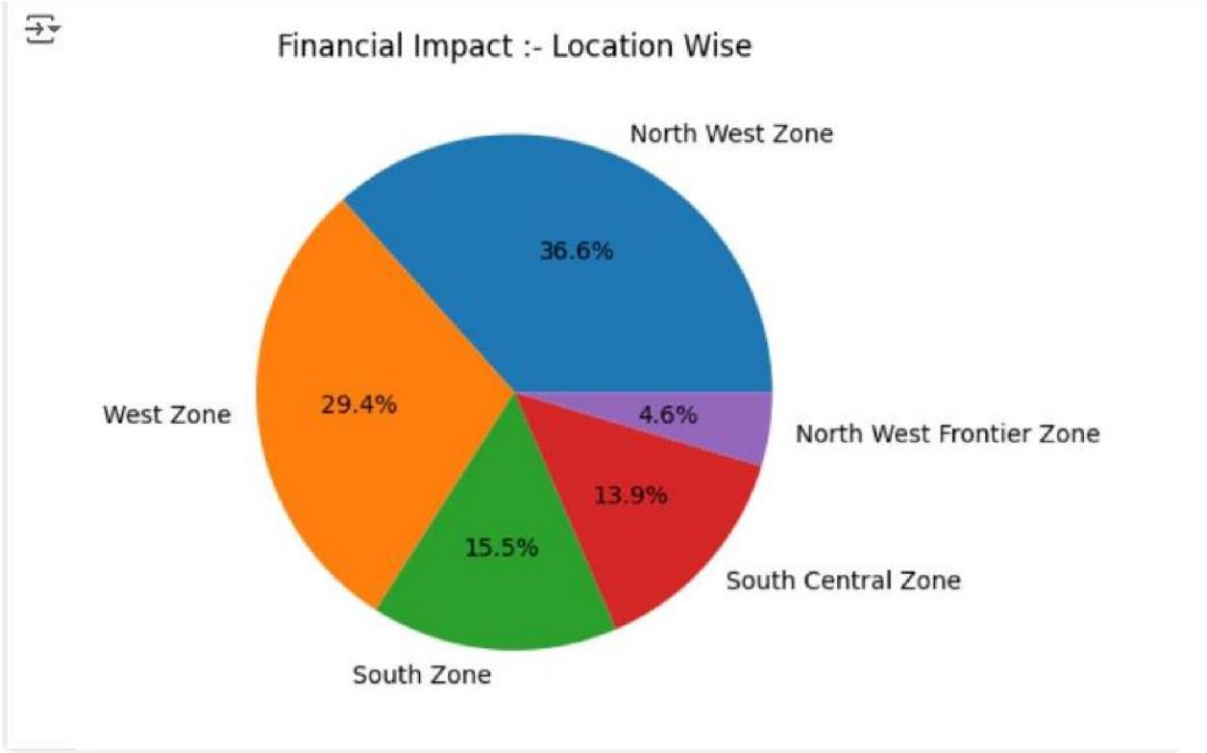
### 4.1.3 Cleaning of HTML Tags from the CSV file and Converting it to a New File

```
[ ]  import csv
     import re

     with open('observations.csv', 'r') as csvfile:
         reader = csv.reader(csvfile)

         with open('newobservations.csv', 'w', newline='') as outfile:
             writer = csv.writer(outfile)

             for row in reader:
                 cleaned_row = [re.sub(r'<.*?>', '', cell) for cell in row]
                 writer.writerow(cleaned_row)
```

## 4.2 DATA VISUALISATION

### Location wise Financial Impact

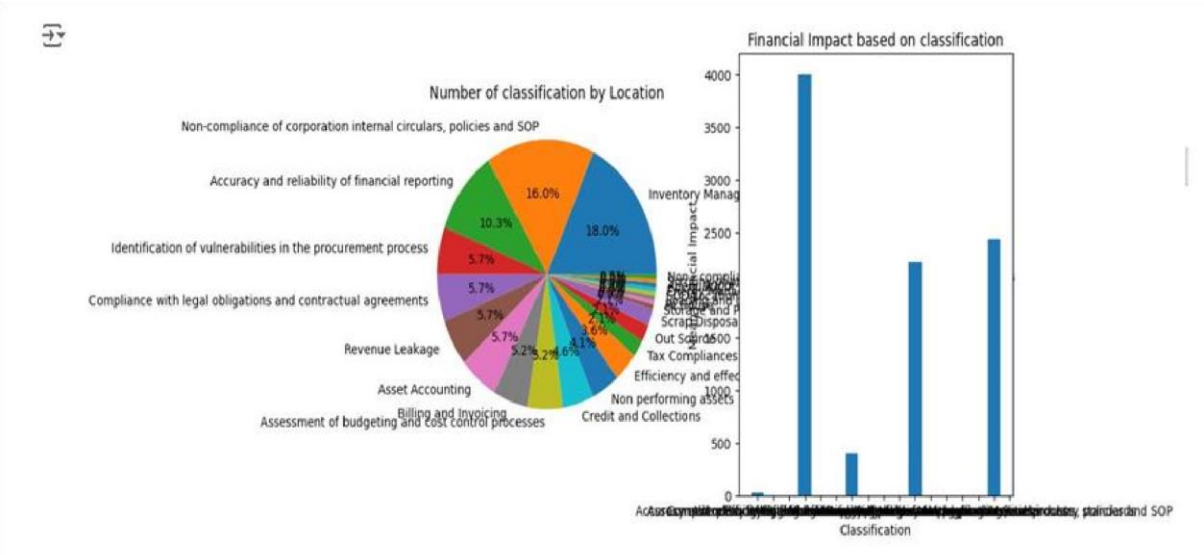
```
+ Code + Text

[ ] import pandas as pd
    import matplotlib.pyplot as plt

    # Merge the two DataFrames on the 'uk_id' column
    merged_df = pd.merge(table1, table2, on='uk_id')

    # Attempt to convert 'financial_impact' to numeric, coercing errors to NaN
    merged_df['financial_impact'] = pd.to_numeric(merged_df['financial_impact'], errors='coerce')

    # Get the number of observations for each location
    location_counts = merged_df['audited_zone'].value_counts()

    # Create a figure and axis object
    fig, ax = plt.subplots()

    # Create a pie chart
    ax.pie(location_counts, labels=location_counts.index, autopct='%1.1f%%')

    # Set title
    ax.set_title('Financial Impact :- Location Wise')

    # Show the plot
    plt.show()
```



**Location Wise Classifications and Financial Impact based on Classifications**

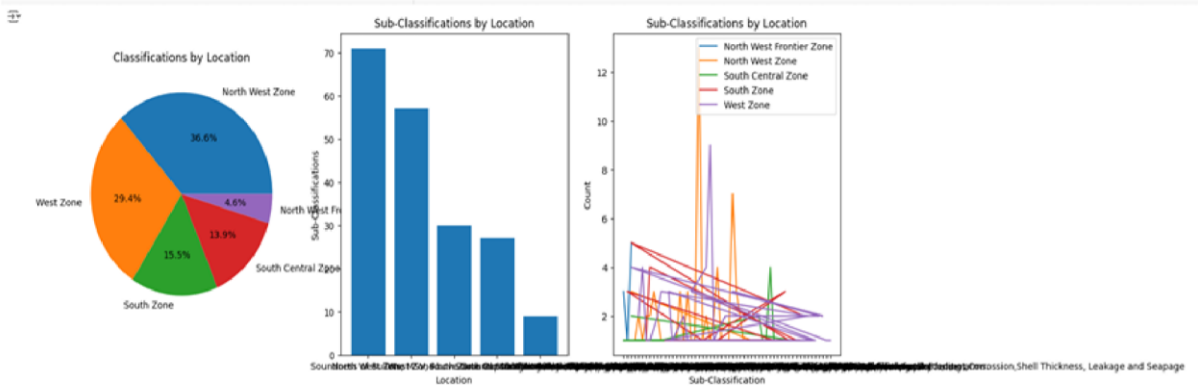
+ Code + Text

```
# Read the csv files
table1 = pd.read_csv('observations.csv') # Load data into table1
table2 = pd.read_csv('parent_audit_reports.csv') # Load data into table2

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'financial_impact' to numeric, coercing errors to NaN
merged_df['financial_impact'] = pd.to_numeric(merged_df['financial_impact'], errors='coerce')

# Get the number of classification for each location
location_counts = merged_df['classification'].value_counts()

# Calculate the mean of a numeric column (e.g., 'financial_impact') for each location
# Ensure the column you choose is numeric
location_impact = merged_df.groupby('classification')['financial_impact'].mean().reset_index()

# Create a figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

# Create a pie chart for location counts
ax1.pie(location_counts, labels=location_counts.index, autopct='%1.1f%%')
ax1.set_title('Number of classification by Location')

ax2.bar(location_impact['classification'], location_impact['financial_impact'])
ax2.set_title('Financial Impact based on classification')
ax2.set_xlabel('Classification')
ax2.set_ylabel('Mean Financial Impact')

# Show the plot
plt.show()
```



## Location wise Different Classifications

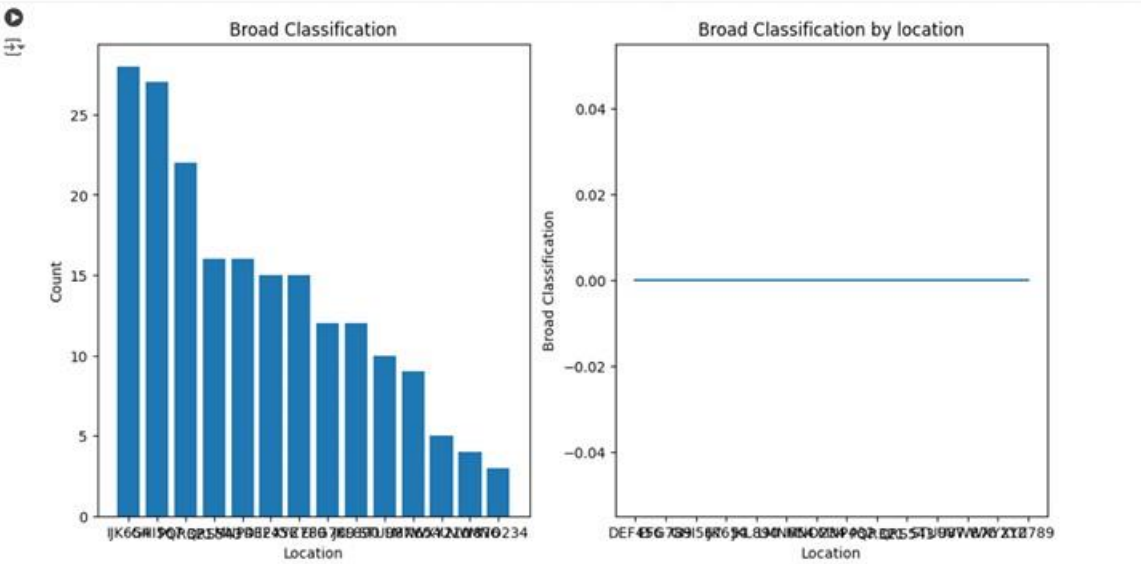
```
import pandas as pd
import matplotlib.pyplot as plt

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Get the number of observations for each location
location_counts = merged_df['audited_zone'].value_counts()

# Get the sub-classifications for each location
location_subclass = merged_df['audited_zone'].value_counts().reset_index(name='sub_classification')
location_subclass.columns = ['audited_zone', 'sub_classification']

# Get the sub-classifications for each location
subclass_counts = merged_df.groupby(['audited_zone', 'sub_classification'])['sub_classification'].count().reset_index(name='subclass_counts')

# Create a figure with three subplots
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 6))

# Create a pie chart for location counts
ax1.pie(location_counts, labels=location_counts.index, autopct='%1.1f%%')
ax1.set_title('Classifications by Location')

ax2.bar(location_subclass['audited_zone'], location_subclass['sub_classification'])
ax2.set_title('Sub-Classifications by Location')
ax2.set_xlabel('Location')
ax2.set_ylabel('Sub-Classifications')

for zone in subclass_counts['audited_zone'].unique():
    subclass_data = subclass_counts[subclass_counts['audited_zone'] == zone]
    ax3.plot(subclass_data['sub_classification'], subclass_data['subclass_counts'], label=zone)
ax3.set_title('Sub-Classifications by Location')
ax3.set_xlabel('Sub-Classification')
ax3.set_ylabel('Count')
ax3.legend()

# Show the plot
plt.show()
```



**Location Wise Broad Classifications**

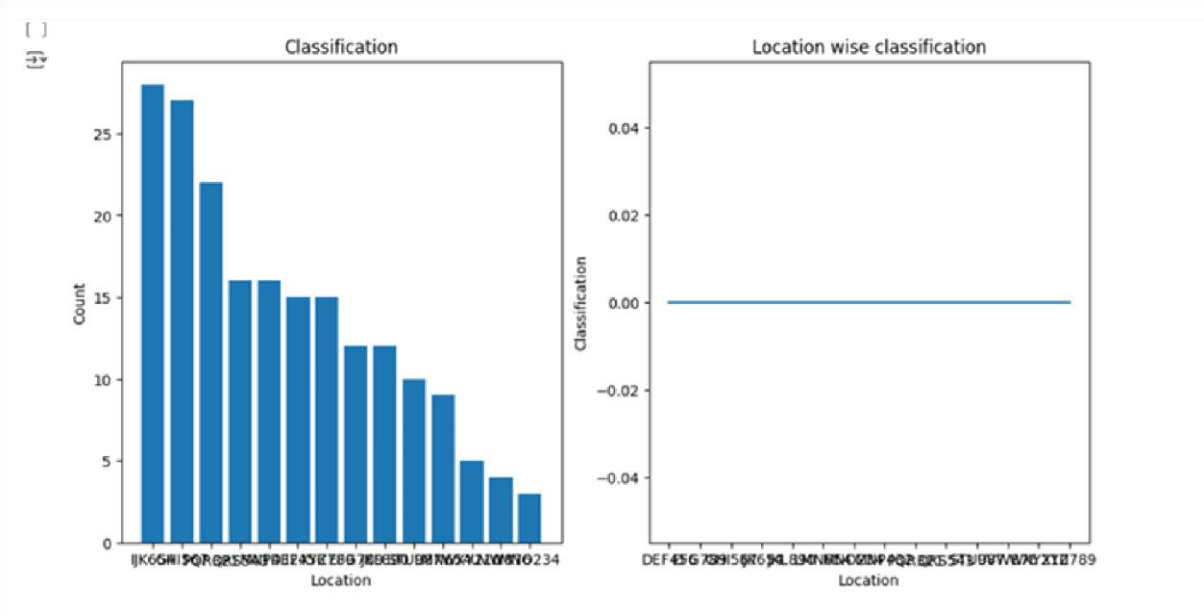
+ Code + Text

```
import pandas as pd
import matplotlib.pyplot as plt

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'broad_classification' to numeric, coercing errors to NaN
merged_df['broad_classification'] = pd.to_numeric(merged_df['broad_classification'], errors='coerce')

# Get the number of observations for each location
location_counts = merged_df['location'].value_counts()

# Get the broad classification by audited zone
broad_classification_by_zone = merged_df.groupby('location')['broad_classification'].sum().reset_index()

# Create a figure and axis object
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Create a bar chart
ax1.bar(location_counts.index, location_counts.values)
ax1.set_title('Broad Classification')
ax1.set_xlabel('Location')
ax1.set_ylabel('Count')

# Create a line chart
ax2.plot(broad_classification_by_zone['location'], broad_classification_by_zone['broad_classification'])
ax2.set_title('Broad Classification by location')
ax2.set_xlabel('Location')
ax2.set_ylabel('Broad Classification')

# Show the plot
plt.show()
```



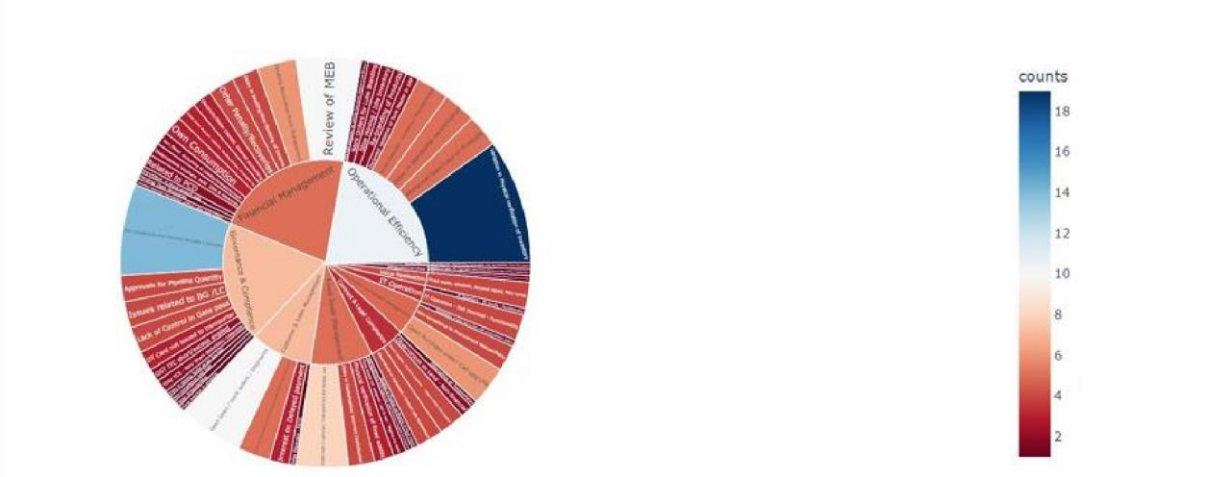
**Location Wise Classifications**

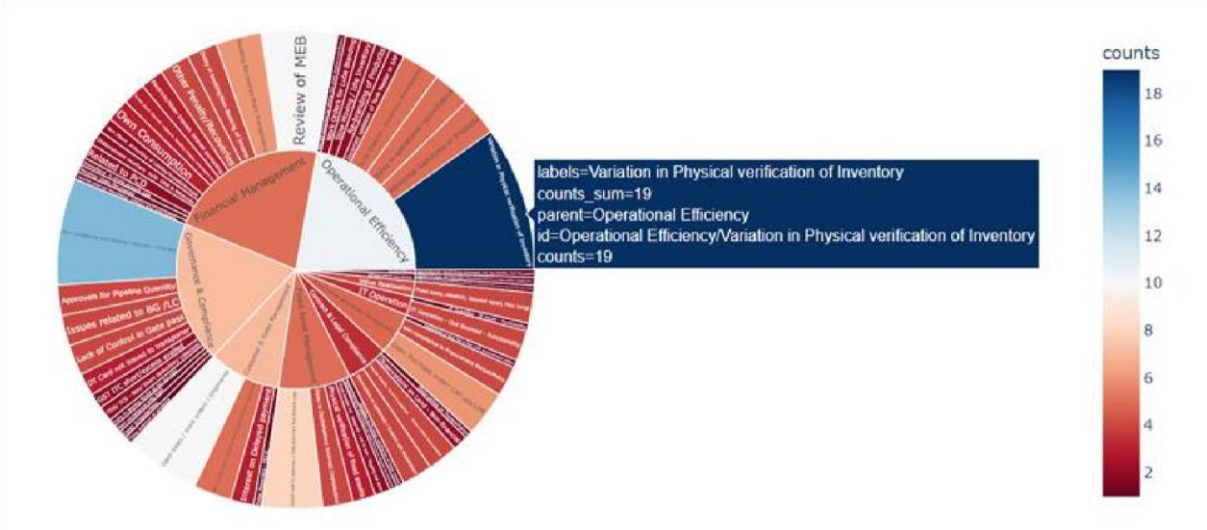
```
+ Code + Text

import pandas as pd
import matplotlib.pyplot as plt

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'broad_classification' to numeric, coercing errors to NaN
merged_df['classification'] = pd.to_numeric(merged_df['classification'], errors='coerce')

# Get the number of observations for each location
location_counts = merged_df['location'].value_counts()

# Get the broad classification by audited zone
classification_by_zone = merged_df.groupby('location')['classification'].sum().reset_index()

# Create a figure and axis object
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Create a bar chart
ax1.bar(location_counts.index, location_counts.values)
ax1.set_title('Classification')
ax1.set_xlabel('Location')
ax1.set_ylabel('Count')

# Create a line chart
ax2.plot(classification_by_zone['location'],classification_by_zone['classification'])
ax2.set_title('Location wise classification')
ax2.set_xlabel('Location')
ax2.set_ylabel('Classification')

# Show the plot
plt.show()
```



**Group-by Broad-Classification and Sub-Classification**

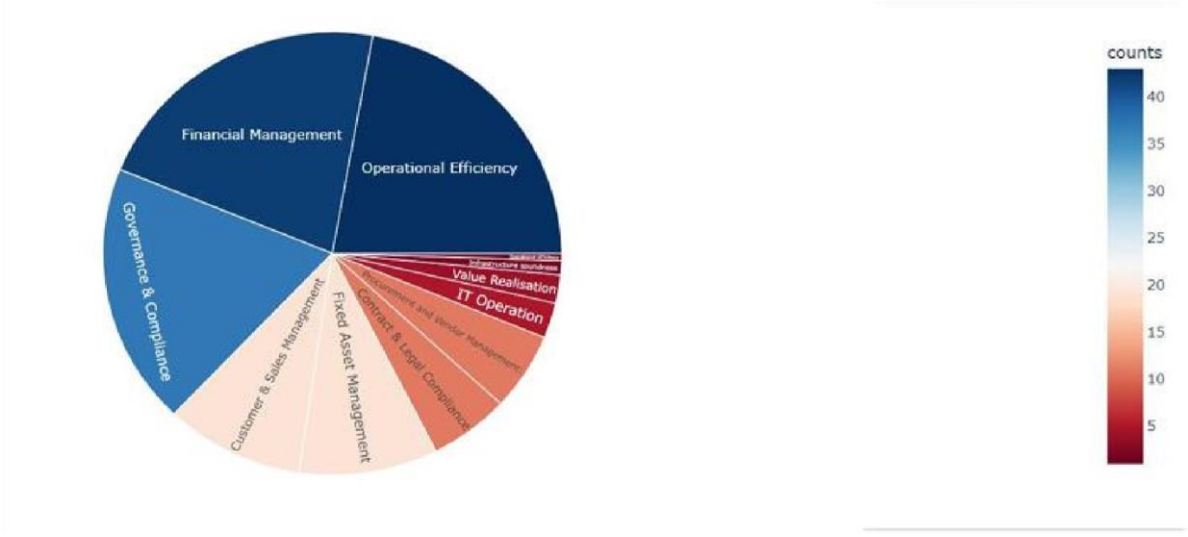
```
import plotly.express as px
import pandas as pd

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'broad_classification' to numeric, coercing errors to NaN
merged_df['audited_zone'] = pd.to_numeric(merged_df['audited_zone'], errors='coerce')

# Group by both 'broad_classification' and 'sub_classification' and count
classification_display = merged_df.groupby(['broad_classification', 'sub_classification']).size().reset_index(name='counts')

# Create a sunburst chart
fig = px.sunburst(classification_display, path=['broad_classification', 'sub_classification'], values='counts',
                  color='counts', color_continuous_scale='RdBu')

# Show the plot
fig.show()
```

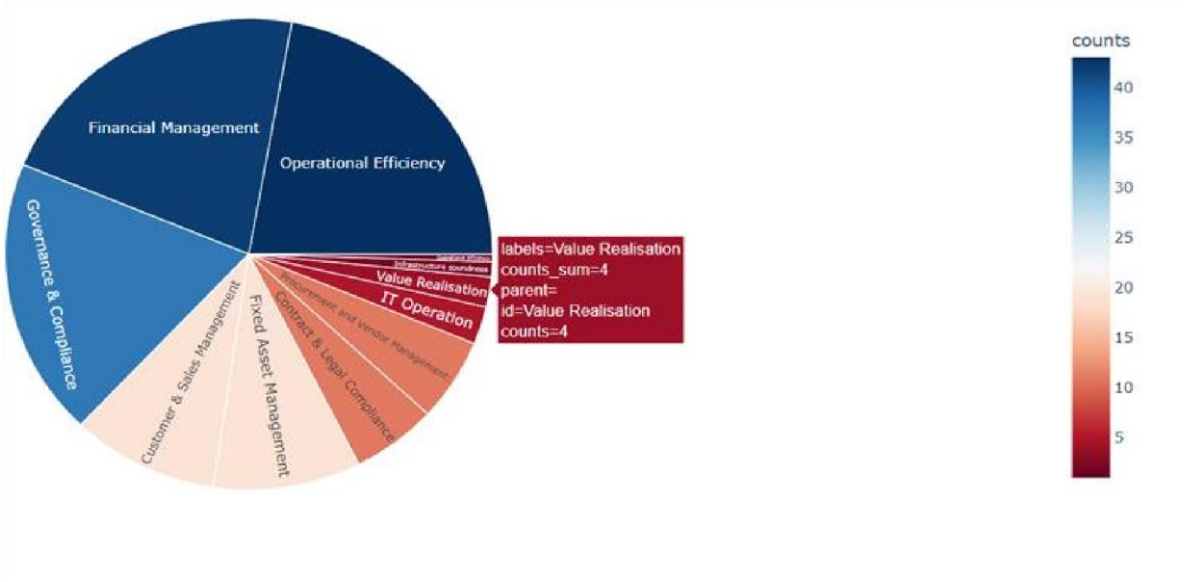


**Group by 'broad_classification' and count**

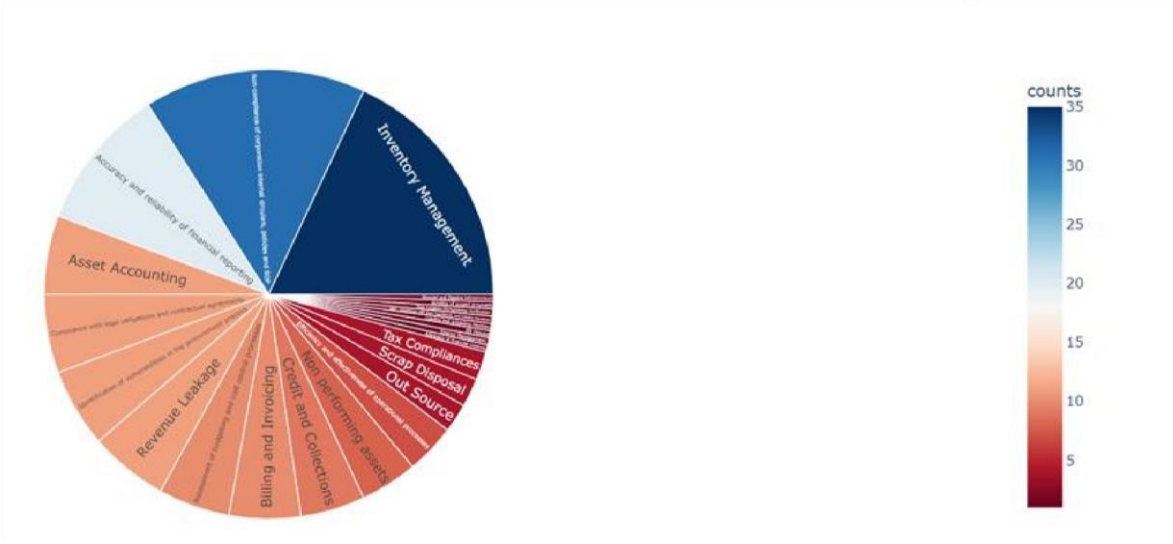
```
import plotly.express as px
import pandas as pd

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'broad_classification' to numeric, coercing errors to NaN
merged_df['audited_zone'] = pd.to_numeric(merged_df['audited_zone'], errors='coerce')

# Group by'broad_classification'and count
classification_display = merged_df.groupby('broad_classification').size().reset_index(name='counts')

# Create a sunburst chart
fig = px.sunburst(classification_display, path=['broad_classification'], values='counts',
                  color='counts', color_continuous_scale='RdBu')

# Show the plot
fig.show()
```

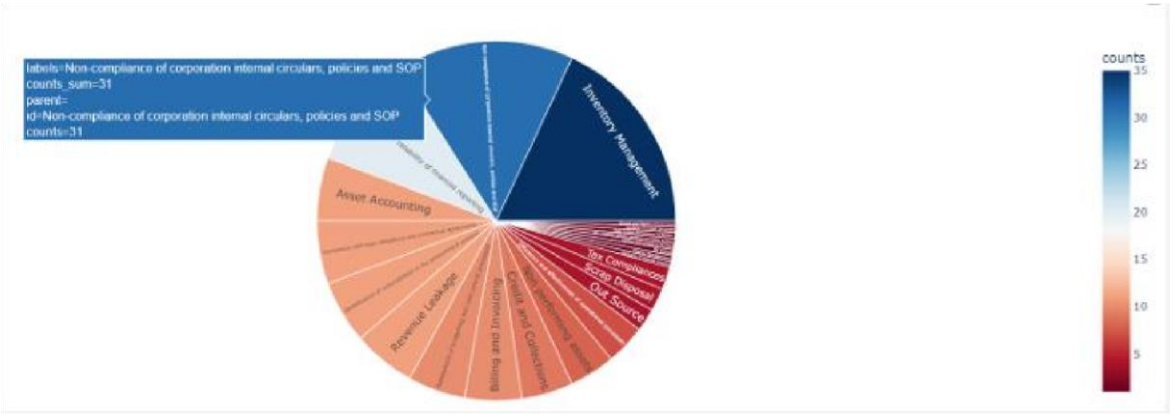


**Group by Classification and Count**

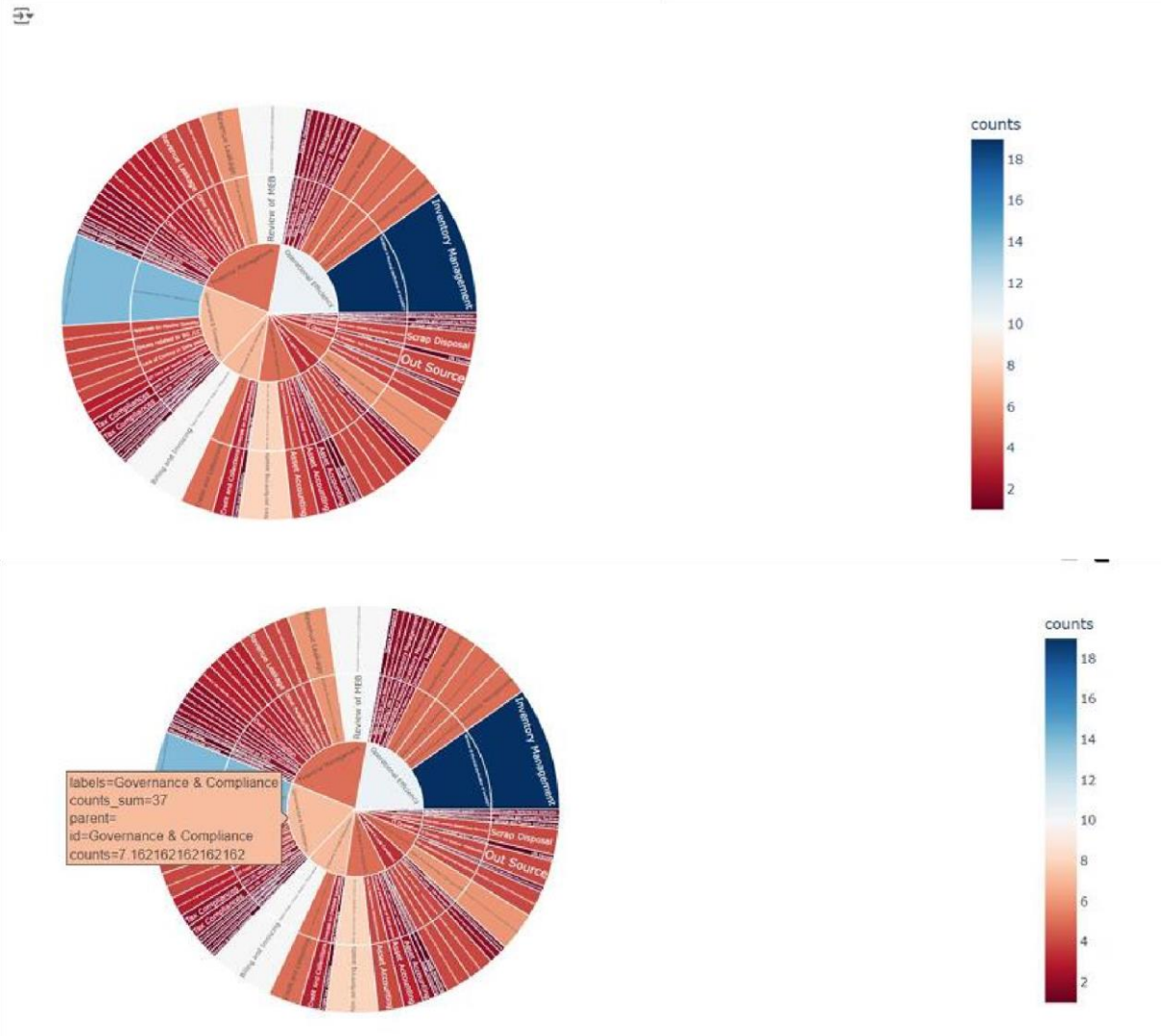
```
import plotly.express as px
import pandas as pd

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'broad_classification' to numeric, coercing errors to NaN
merged_df['audited_zone'] = pd.to_numeric(merged_df['audited_zone'], errors='coerce')

# Group by both 'broad_classification' and 'sub_classification' and count
classification_display = merged_df.groupby(['classification']).size().reset_index(name='counts')

# Create a sunburst chart
fig = px.sunburst(classification_display, path=['classification'], values='counts',
                  color='counts', color_continuous_scale='RdBu')

# Show the plot
fig.show()
```



**Group by all 'broad_classification' and 'sub_classification' and ‘classification’ and count**

```
import plotly.express as px
import pandas as pd

# Merge the two DataFrames on the 'uk_id' column
merged_df = pd.merge(table1, table2, on='uk_id')

# Attempt to convert 'broad_classification' to numeric, coercing errors to NaN
merged_df['audited_zone'] = pd.to_numeric(merged_df['audited_zone'], errors='coerce')

# Group by both 'broad_classification' and 'sub_classification' and count
classification_display = merged_df.groupby(['broad_classification', 'sub_classification','classification']).size().reset_index(name='counts')

# Create a sunburst chart
fig = px.sunburst(classification_display, path=['broad_classification', 'sub_classification','classification'], values='counts',
                  color='counts', color_continuous_scale='RdBu')

# Show the plot
fig.show()
```



**Clustering of Data using BERT (Bidirectional Encoder Representations from Transformers)**

First install the library for clustering

```
[ ]  !pip install numpy
     !pip install scikit-learn
     !pip install matplotlib
```

**CODE**

```
import pandas as pd
import torch
from transformers import BertTokenizer, BertModel
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Load your dataset into a Pandas DataFrame
df = pd.read_excel('observations.xlsx')

# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_cls_sentence(sentence):
    # Tokenize input sentence and convert to tensor
    inputs = tokenizer.encode_plus(
        sentence,
        add_special_tokens=True,
        max_length=512,
        truncation=True,
        return_attention_mask=True,
        return_tensors='pt'
    )
    input_ids = inputs['input_ids']
    attention_mask = inputs['attention_mask']

    # Pass input through BERT model and extract embeddings for [CLS] token
    with torch.no_grad():
```

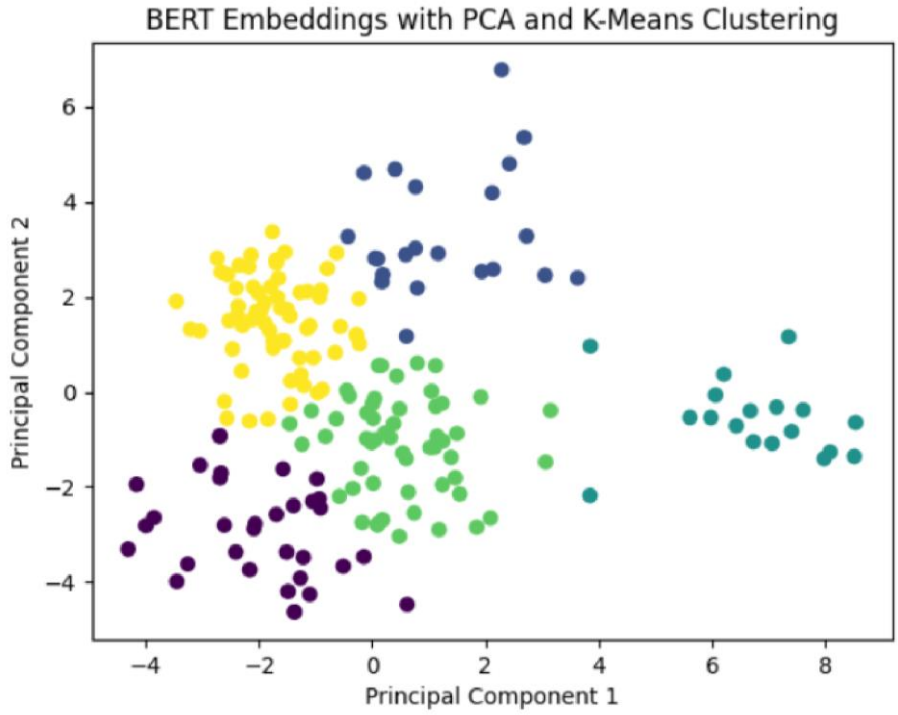
```
    # Pass input through BERT model and extract embeddings for [CLS] token
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        cls_embedding = outputs.last_hidden_state[:, 0, :]

    return cls_embedding.flatten()

# Create a list to store the BERT embeddings
bert_embeddings = []

# Start timer
st = time.time()

# Iterate over the text data and extract BERT embeddings
for sentence in df['summary_of_observation']:
    bert_embedding = get_cls_sentence(sentence)
    bert_embeddings.append(bert_embedding.numpy())

# Convert the list of embeddings to a NumPy array
X_cls_bert = np.vstack(bert_embeddings)

# Print elapsed time
et = time.time()
print("Elapsed time: {:.2f} seconds".format(et - st))

# Apply PCA to reduce dimensionality
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_cls_bert)

# Perform K-Means clustering
kmeans = KMeans(n_clusters=5)  # adjust the number of clusters as needed
kmeans.fit(X_pca)
labels = kmeans.labels_
```

```
# Plot the reduced embeddings using a scatter plot with different colors for each cluster
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('BERT Embeddings with PCA and K-Means Clustering')
plt.show()

# Add the BERT embeddings as a new column to the DataFrame
df['cls_bert'] = bert_embeddings

print(df['cls_bert'])
```

OUTPUT

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggi
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or
  warnings.warn(
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 2.31kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 5.74MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 22.9MB/s]
config.json: 100% 570/570 [00:00<00:00, 32.5kB/s]
model.safetensors: 100% 440M/440M [00:02<00:00, 274MB/s]
Elapsed time: 43.30 seconds
```

```
Elapsed time: 43.30 seconds
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The
  super()._check_params_vs_input(X, default_n_init=10)
```



**Making of Dashboard via Streamlit**

**To make a Dashboard I am using STREAMLIT**

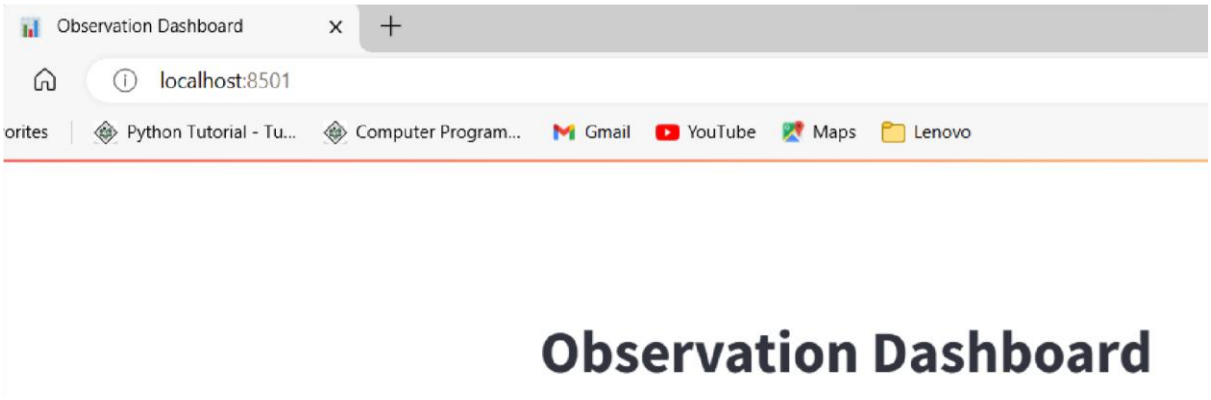
```
pip install streamlit
```

CODE

dashboard.py > ...

```python
import pandas as pd
import plotly.express as px
import streamlit as st
import os
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Streamlit page configuration
st.set_page_config(page_title="Observation Dashboard", page_icon=":bar_chart:")

st.title("Observation Dashboard")
```

OUTPUT



Load data

```
# Load data
file_path = os.path.join(r'C:\Users\lenovo\Downloads\observations.xlsx')
df1 = pd.read_excel(file_path)

file_path1 = os.path.join(r'C:\Users\lenovo\Downloads\parent_audit_reports.xlsx')
df2 = pd.read_excel(file_path1)

# Merge the tables on the 'uk_id' column
merged_table = pd.merge(df1, df2, on='uk_id')
```

C: > Users > lenovo > Downloads > newobservations.xlsx

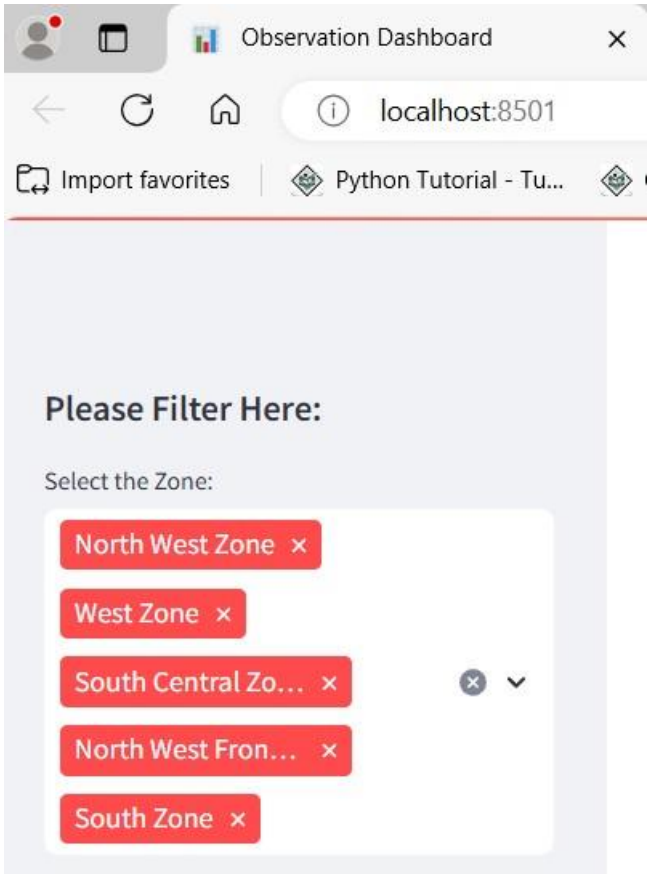
| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | detail_id | broad_classificat | financial_impact | icfr_category | nature | observation_no | risk_category | sub_classificatio | summary_of_obs | uk_id | classification |
| 2 | 4,291 | Governance & C | | Inventory | Repetitive | 1 | Acceptable | Non-compliance | Delay in Operati | 10,340 | Non-compliance |
| 3 | 4,293 | Operational Effic | | Inventory | Repetitive | 2 | Acceptable | Abnormal Gain/I | Gain/Loss entrie | 10,340 | Inventory Manag |
| 4 | 4,296 | Financial Manag | 20.85 lakhs | Inventory | Unique | 3 | Acceptable | Own Consumpti | HSD excess proc | 10,340 | Accuracy and rel |
| 5 | 4,298 | Governance & C | | Inventory | Repetitive | 4 | Acceptable | Lack of Control i | Open RTGP | 10,340 | Non-compliance |
| 6 | 4,300 | Financial Manag | | Expenditure | Repetitive | 5 | Acceptable | Related to PCD | Review of PCD | 10,340 | Accuracy and rel |
| 7 | 4,303 | Governance & C | | Inventory | Repetitive | 6 | Acceptable | Non-compliance | Open SOP Debit | 10,340 | Non-compliance |
| 8 | 4,304 | Procurement an | | Expenditure | Repetitive | 7 | Acceptable | Open Purchase ( | Open Purchase ( | 10,340 | Identification of |
| 9 | 4,306 | Governance & C | 3.36 lakhs | Revenue | Repetitive | 8 | Acceptable | Non-compliance | Pending recover | 10,340 | Non-compliance |
| 10 | 4,307 | Governance & C | 2.01 lakhs | Revenue | Repetitive | 9 | Acceptable | Non-compliance | Pending recover | 10,340 | Non-compliance |

**Making of Sidebar For Filtering Zones**

CODE

```
# Sidebar for filtering
st.sidebar.header("Please Filter Here:")
audited_zone = st.sidebar.multiselect(
    "Select the Zone:",
    options=merged_table["audited_zone"].unique(),
    default=merged_table["audited_zone"].unique()
)

# Define df_selection based on audited_zone filter
df_selection = merged_table.query("audited_zone == @audited_zone")
```

OUTPUT

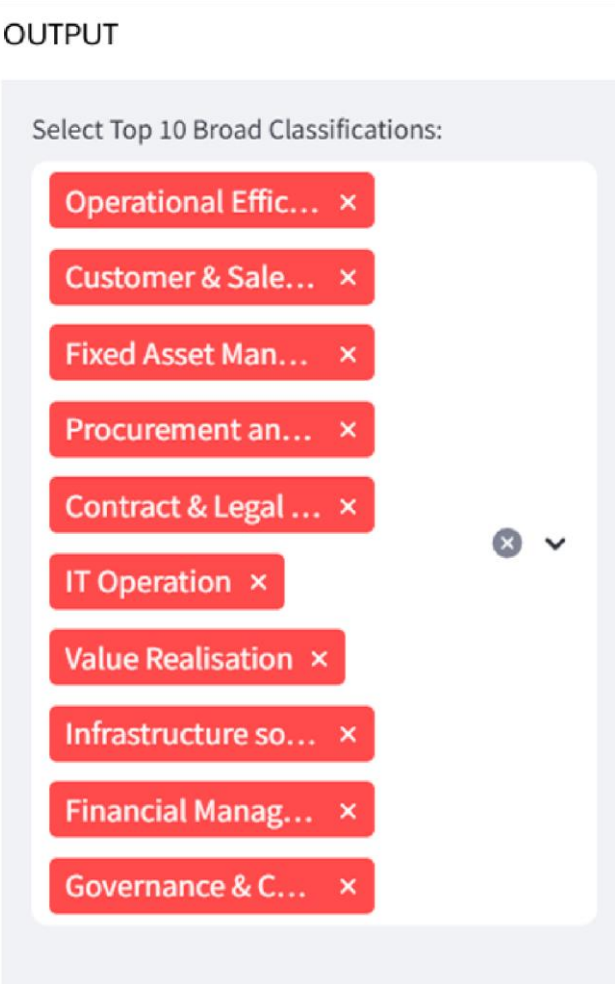


**Making of Sidebar For Filtering Broad Classifications**

CODE

```
# Calculate the top 10 broad classifications by count of observations
top_broad_classifications = df_selection['broad_classification'].value_counts().nlargest(10).index.tolist()

# Sidebar for selecting broad classifications
broad_classification_selection = st.sidebar.multiselect(
    "Select Top 10 Broad Classifications:",
    options=top_broad_classifications,
    default=top_broad_classifications
)

# Filter df_selection based on the selected broad classifications
df_selection = df_selection.query("broad_classification in @broad_classification_selection")
```

OUTPUT



**Summary of Observations for Selected Broad Classifications**

CODE

```
# Display summary of observations for selected broad classifications
summary = df_selection.groupby('broad_classification')['summary_of_observation'].count().reset_index()
summary.columns = ['Broad Classification', 'Number of Observations']
st.write("**Summary of Observations for Selected Broad Classifications:**")
st.dataframe(summary)
```

OUTPUT

**Summary of Observations for Selected Broad Classifications:**

| | Broad Classification | Number of Observations |
|---|---|---|
| 0 | Contract & Legal Compliance | 11 |
| 1 | Customer & Sales Management | 19 |
| 2 | Financial Management | 42 |
| 3 | Fixed Asset Management | 19 |
| 4 | Governance & Compliance | 37 |
| 5 | IT Operation | 5 |
| 6 | Infrastructure soundness | 2 |
| 7 | Operational Efficiency | 43 |
| 8 | Procurement and Vendor Management: | 11 |
| 9 | Value Realisation | 4 |

**Displaying Bar Chart for Financial Impacts based on Zones**

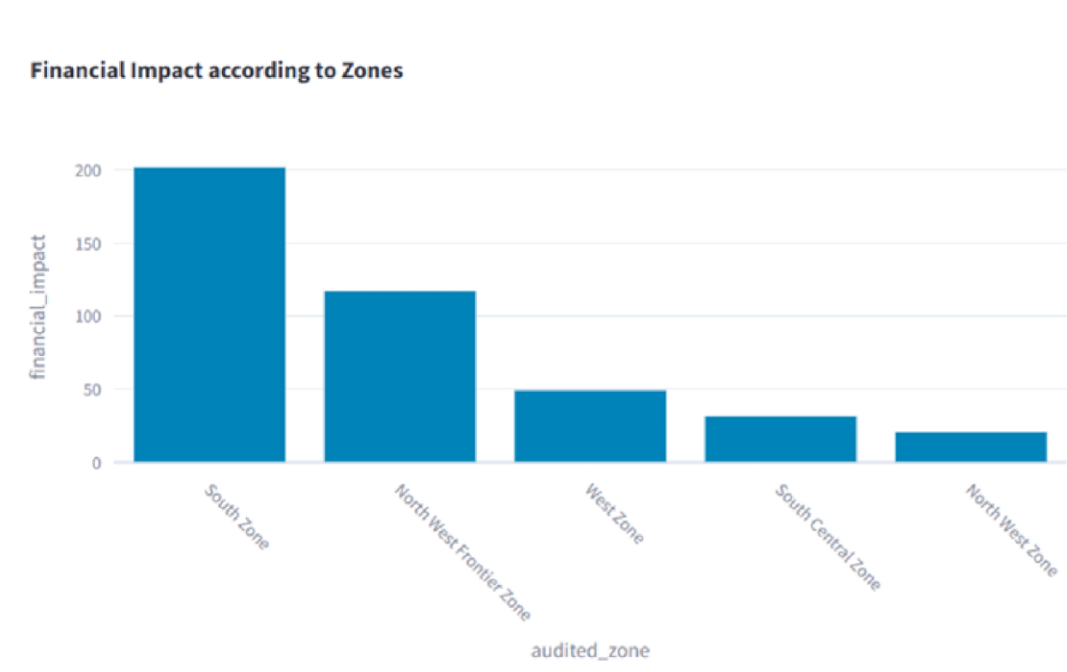
CODE

dashboard.py | parent_audit_reports.xlsx | observations.xlsx | newobservations.xlsx

dashboard.py > ...

```
65  # Sort the dataframe in descending order of financial impact
66  zone_financial_impact = zone_financial_impact.sort_values(by='financial_impact', ascending=False)
67
68  # Create the bar chart
69  fig = px.bar(
70      zone_financial_impact,
71      x="audited_zone",
72      y="financial_impact",
73      orientation="v",  # vertical bar chart
74      title="<b>Financial Impact according to Zones</b>",
75      color_discrete_sequence=["#0083B8"] * len(zone_financial_impact),
76      template="plotly_white",
77  )
78
79  fig.update_layout(
80      plot_bgcolor="rgba(0,0,0,0)",
81      xaxis=dict(
82          showgrid=False,
83          tickangle=45  # rotate x-axis labels by 45 degrees
84      ),
85  )
86
87  # Convert financial_impact to numeric data type
88  df_selection['financial_impact'] = df_selection['financial_impact'].apply(extract_numeric)
89
90  # Handle NaN values
91  df_selection.dropna(subset=['financial_impact'], inplace=True)
92
```

OUTPUT



**Creating WordCloud of SUB CLASSIFICATIONS**

CODE

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
# Create word cloud
text = ' '.join(df_selection['sub_classification'].tolist())
wordcloud = WordCloud(width=800, height=400, max_font_size=110).generate(text)
```

```
st.write("**Word Cloud of SUB CLASSIFICATIONS**")
fig_wordcloud, ax = plt.subplots(figsize=(10, 8))
ax.imshow(wordcloud, interpolation='bilinear')
ax.axis('off')
st.pyplot(fig_wordcloud)
```

OUTPUT

Word Cloud of SUB CLASSIFICATIONS

**Location wise Number of Observations**

CODE

```python
# Group by location and count the number of observations
location_summary = df_selection.groupby('location')['summary_of_observation'].count().reset_index()
location_summary.columns = ['Location', 'Number of Observations']

# Create the bar chart
fig_bar = px.bar(
    location_summary,
    x="Location",
    y="Number of Observations",
    template="plotly_white",
)

fig_bar.update_layout(
    plot_bgcolor="rgba(0,0,0,0)",
    xaxis=dict(
        showgrid=False,
        tickangle=45  # rotate x-axis labels by 45 degrees
    ),
)

# Display the bar chart on the Streamlit dashboard
st.write("**Summary of Observations by Location**")
st.plotly_chart(fig_bar, use_container_width=True)
```

OUTPUT

**Broad Classification Summary – Counts and Financial Impacts** CODE

```
dashboard.py > ...
124
125  import plotly.express as px
126
127  # Group by broad_classification and calculate count and financial impact
128  broad_classification_summary = df_selection.groupby(['audit_period_start', 'audit_period_end', 'broad_classification']).agg({
129      'summary_of_observation': 'count',
130      'financial_impact': 'sum'
131  }).reset_index()
132
133  # Rename columns
134  broad_classification_summary.columns = ['Audit Period Start', 'Audit Period End', 'Broad Classification', 'Count', 'Financial Impact']
135
136  # Melt the DataFrame for plotting
137  melted_df_count = broad_classification_summary[['Audit Period Start', 'Audit Period End', 'Broad Classification', 'Count']].melt(id_vars=['Audit Period Start', 'Audit Per
138  melted_df_financial = broad_classification_summary[['Audit Period Start', 'Audit Period End', 'Broad Classification', 'Financial Impact']].melt(id_vars=['Audit Period Sta
139
```

OUTPUT


Broad Classification Summary - Financial Impacts
Broad Classification
Contract & Legal Compliance
Financial Management
Fixed Asset Management
Governance & Compliance
Operational Efficiency
Procurement and Vendor Management:
Customer & Sales Management
Value Realisation
Infrastructure soundness
IT Operation
count
0
0.5
1
1.5
2
2.5
3
01-01-2023 00:00
01-02-2023 00:00
01-11-2022 00:00
01-12-2022 00:00
31-10-2023 00:00
31-12-2023 00:00
31-01-2024 00:00
30-09-2023 00:00
01-04-2022 00:00
01-04-2023 00:00
01-06-2022 00:00
31-03-2023 00:00
30-06-2023 00:00
12-09-2022 00:00
14-09-2023 00:00
Financial Impact

**CHAPTER 5 : CONCLUSION AND REFERENCES**

# Chapter – 5

## 5.1 Conclusion

The Exploratory Data Analysis (EDA) application developed using Streamlit and Python provides a powerful and user-friendly tool for analyzing audit observations. Through the integration of data manipulation, statistical analysis, and visualization techniques, the application allows auditors to gain deeper insights into their data, identify patterns and anomalies, and make data-driven decisions. The project demonstrates the effectiveness of modern data analysis tools in improving the accuracy and efficiency of audit processes. Overall, the EDA application is a significant step forward in leveraging technology to enhance audit capabilities, ensuring that audits are both thorough and insightful.

## 5.2 System Specifications

### 5.2.1 Hardware Requirements

The hardware requirements for running the EDA application are minimal, making it accessible to a wide range of users. The recommended hardware specifications include:

- **Processor**: A multi-core processor, such as Intel i5 or equivalent, to handle the computational tasks involved in data processing and visualization.
- **RAM**: At least 8GB of RAM to support efficient data manipulation and the operation of multiple applications simultaneously.
- **Storage**: A solid-state drive (SSD) with a capacity of at least 256GB to store the dataset, application files, and temporary processing data.
- **Display**: A monitor with a resolution of at least 1920x1080 pixels for clear visualization of data and application interfaces.
- **Internet Connectivity**: Reliable internet access is necessary for downloading software libraries, accessing data sources, and deploying the Streamlit application on a web server or cloud platform.

### 5.2.2 Software Requirements

The software requirements for the EDA application include the following:

- **Operating System**: The application is compatible with major operating systems, including Windows, macOS, and Linux.
- **Python**: Version 3.7 or higher is required to run the application and its associated libraries.
- **Python Libraries**: Essential libraries include Pandas for data manipulation, NumPy for numerical operations, Matplotlib and Seaborn for data visualization, and Streamlit for developing the web application interface.
- **Web Browser**: A modern web browser (such as Chrome, Firefox, or Edge) is required to access and interact with the Streamlit application.

## 5.3 Limitations of the System

Despite its capabilities, the EDA application has certain limitations:

- **Scalability**: While the application is designed to handle moderate-sized datasets, performance may degrade with extremely large datasets or complex analyses, necessitating more powerful hardware or optimization techniques.
- **Customization**: The application is tailored for specific use cases related to audit observations, and may require significant customization for other types of data or analysis tasks.
- **Real-Time Analysis**: Although the application supports real-time data visualization, it is not optimized for high-frequency data streams or real-time processing at a large scale.
- **Dependency on Python**: The application relies heavily on Python and its libraries, which may present challenges for users unfamiliar with the language or environment.

## 5.4 Future Scope for Modification

There are several areas where the EDA application can be enhanced in the future:

- **Scalability Improvements**: Implementing advanced data processing techniques or integrating with big data platforms (e.g., Hadoop, Spark) could improve the application's ability to handle larger datasets.
- **Machine Learning Integration**: Adding machine learning capabilities for predictive analysis or automated anomaly detection could expand the functionality and utility of the application.
- **Customization Options**: Enhancing the user interface to allow for more flexible customization of analysis parameters and visualization types would make the application more adaptable to different use cases.
- **Mobile Compatibility**: Developing a mobile-friendly version of the application would increase accessibility, allowing users to interact with the data and visualizations on the go.
- **Advanced Security Features**: Implementing robust security features, such as data encryption and user authentication, would make the application more suitable for sensitive data environments.

**5.5 References/Bibliography**

- https://colab.research.google.com/

- https://www.analyticsvidhya.com/

- Google

- Youtube

- H. petroleum Corp. Ltd., "About Hindustan petroleum corp. ltd.

**CHAPTER 6 : ANNEXURES**

# Chapter - 6

### A-1 Data Dictionary

The Data Dictionary is a comprehensive reference that defines and describes all the data elements used in the EDA application. It includes detailed information on each variable, such as the variable name, data type, format, allowed values, and a description of its purpose. For example, entries might include variables like

**UK-ID** (integer, unique identifier for each audit observation), **Date** (date, the date the observation was recorded). The Data Dictionary ensures that all data elements are consistently used and understood throughout the development and analysis process.

### A-2 Test Reports

Test Reports document the testing phase of the EDA application, detailing the results of various tests performed to ensure the system functions correctly. This section includes descriptions of the test cases, the expected outcomes, and the actual results observed during testing. Tests may cover areas such as data validation, user interface functionality, performance under load, and integration between different system components. For example, a test report might verify that the data visualization module correctly generates histograms and scatter plots without errors. The reports provide evidence of the system's reliability and help identify areas that may require further refinement.

### A-3 Sample Inputs

Sample Inputs showcase examples of the data that users would enter into the EDA application. These inputs are provided to demonstrate how the system processes various types of data and to ensure that the input forms and validation checks function as intended. For instance, sample inputs might include audit observation records with fields like date, department, auditor name, observation details, and anomaly status. Providing sample inputs helps in verifying that the application can handle different data scenarios and that users can easily input the required information without errors.

**A-4 Sample Outputs**

Sample Outputs illustrate the results generated by the EDA application after processing the input data. These outputs might include statistical summaries, data visualizations (e.g., histograms, box plots, scatter plots), and generated reports that highlight key findings from the analysis. For example, a sample output might show a bar chart summarizing the frequency of different types of anomalies across departments or a heatmap depicting correlations between various audit metrics. These examples help in validating that the system produces accurate and meaningful results, aligning with user expectations and the system's objectives.

**A-5 Coding (Optional)**

The Coding section, though optional, provides excerpts or full code listings of the Python scripts used to develop the EDA application. This may include code for data processing, visualization, and the Streamlit application interface. For example, this section might feature code snippets that demonstrate how data is cleaned and transformed using Pandas, how visualizations are created using Matplotlib and Seaborn, or how the Streamlit application is structured to create interactive dashboards. Including the coding details can be valuable for developers who need to understand or modify the application's functionality, offering insights into the implementation of key features.