INDEX

Practical 1

Implement Data Loading, Storage and File Formats. Read data and store them in text format.

To understand the process of data loading, storage, and handling various file formats, and to implement these operations in a programmatic manner for reading and storing data in text format.

Objective:

- Learn the significance of data loading, storage, and file formats in data processing.
- Understand how to read data from files and store it efficiently.
- Implement methods for handling data in text format (e.g., .txt, .csv, .json)
- Ensure scalability, reliability, and accuracy in data handling operations.

Dataset used:

We have used inbuilt python dictionary to create a dataset, and then with the help of pandas library we have save the file in various format.

Code:

```
import pandas as pd

#sample studet data a list of dictionaries

students = [
    {"id":1 , "name":"Alice" , "age":20 , "marks":85},
    {"id":2 , "name":"Bob" , "age":22 , "marks":78},
    {"id":3 , "name":"Charlie" , "age":21 , "marks":92},
]

#Convert to Dataframe

df = pd.DataFrame(students)

#Display the data

print("Original Dataframe: \n", df)
```

Original Dataframe:

	id	name	age	marks
0	1	Alice	20	85
1	2	Bob	22	78
2	3	Charlie	21	92

```
#Save Dataframe to CSV
df.to_csv("students.csv" , index=False)

#Read back the CSV file
df_csv = pd.read_csv("students.csv")
print("\n Data loaded from CSV: \n ",df_csv)
```

Data loaded from CSV:

	id	name	age	marks
0	1	Alice	20	85
1	2	Bob	22	78
2	3	Charlie	21	92

```
# Save Dataframe to json file
df.to_json("students.json" , orient="records", indent=4)

#Read JSON back
df_json = pd.read_json("students.json")
print("\n Data loaded from JSON: \n", df_json)
```

Data loaded from JSON:

	id	name	age	marks
0	1	Alice	20	85
1	2	Bob	22	78
2	3	Charlie	21	92

```
#write to a .txt file line by line
with open("students.txt" , "w") as f:
  for index , row in df.iterrows():
    line = f"{row['id']} , {row['name']} , {row['age']} , {row['marks']}\n"
    f.write(line)

#Raed the text file
with open("students.txt" , "r") as f:
  lines = f.readlines()
  print("\n Data loaded from text file: \n")
  for line in lines:
    print(line.strip())
```

Data loaded from text file:

1 , Alice , 20 , 85
2 , Bob , 22 , 78
3 , Charlie , 21 , 92

Practical 2

Implement the code to interact with Web APIs and to perform web scrapping

The aim is to implement Python code that can interact with web APIs and perform web scraping. These functionalities enable automation for extracting structured and unstructured data from the web. This data can be used for applications such as data analysis, machine learning, and content aggregation.

Libraries Used:

- BeautifulSoup (from bs4): For parsing HTML and XML documents.
- requests: For sending HTTP requests and retrieving HTML content.

Dataset used:

We have used the 'http://quotes.toscrape.com/' website to scrape the data. Along that we have also used the OpenWeatherMap API to scape weather information.

Code:

```
import requests
from bs4 import BeautifulSoup

# Step 1: Send an HTTP request to the website
url = 'http://quotes.toscrape.com/'
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Step 2: Parse the HTML content of the page with BeautifulSoup
    soup = BeautifulSoup(response.text, 'html.parser')

    # Step 3: Find all the quotes and authors
    quotes_data = []  # To store the quotes and authors

    # In the website, quotes are inside <span> tags with class 'text' and authors in <small> tags with class 'author'
    quotes = soup.find_all('span', class_='text')
    authors = soup.find_all('small', class_='author')

    # Step 4: Loop through the quotes and authors, and store them in a dictionary
    for i in range(len(quotes)):
        quote_text = quotes[i].text
        author = authors[i].text
        quotes_data.append({'quote': quote_text, 'author': author})

    # Step 5: Display the scraped data
    for entry in quotes_data:
        print(f"Quote: {entry['quote']}")
        print(f"Author: {entry['author']}")
        print('---')
else:
    print(f"Failed to retrieve the webpage. Status code: {response.status_code}")
```

Quote: “The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”
Author: Albert Einstein

Quote: “It is our choices, Harry, that show what we truly are, far more than our abilities.”
Author: J.K. Rowling

Quote: “There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”
Author: Albert Einstein

Quote: “The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid.”
Author: Jane Austen

Quote: “Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring.”
Author: Marilyn Monroe

Quote: “Try not to become a man of success. Rather become a man of value.”
Author: Albert Einstein

Quote: “It is better to be hated for what you are than to be loved for what you are not.”
Author: André Gide

Quote: “I have not failed. I've just found 10,000 ways that won't work.”
Author: Thomas A. Edison

Quote: “A woman is like a tea bag; you never know how strong it is until it's in hot water.”
Author: Eleanor Roosevelt

Quote: “A day without sunshine is like, you know, night.”
Author: Steve Martin

```
import requests
# Replace with your actual OpenWeatherMap API key
API_KEY ='8879b0a6acf716373b5d1dbe7b911960'
city = 'London'
# Adding country code to avoid ambiguity
# Define the API endpoint and include your API key
url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"
# Send a GET request to the API
response = requests.get(url)
# Check if the request was successful
if response.status_code == 200:
    data = response.json()
    print(f"City: {data['name']}")
    print(f"Temperature: {data['main']['temp']}°C")
    print(f"Weather: {data['weather'][0]['description']}")
else:
    print(f"Failed to retrieve data. Status code: {response.status_code}, Reason: {response.reason}")
```

City: London
Temperature: 5.86°C
Weather: overcast clouds

```
# Replace with your actual OpenWeatherMap API key
API_KEY = '8879b0a6acf716373b5d1dbe7b911960'
city = 'London'  # Adding country code to avoid ambiguity
# Define the API endpoint and include your API key
url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"
# Send a GET request to the API
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    data = response.json()
    print(f"City: {data['name']}")
    print(f"Temperature: {data['main']['temp']}°C")
    print(f"Weather: {data['weather'][0]['description']}")
else:
    print(f"Failed to retrieve data. Status code: {response.status_code}, Reason: {response.reason}")
```

City: London
Temperature: 5.86°C
Weather: overcast clouds

```
# Replace with your actual OpenWeatherMap API key
API_KEY = '8879b0a6acf716373b5d1dbe7b911960'
city = 'Mumbai'  # Adding country code to avoid ambiguity
# Define the API endpoint and include your API key
url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"
# Send a GET request to the API
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    data = response.json()
    print(f"City: {data['name']}")
    print(f"Temperature: {data['main']['temp']}°C")
    print(f"Weather: {data['weather'][0]['description']}")
else:
    print(f"Failed to retrieve data. Status code: {response.status_code}, Reason: {response.reason}")
```

City: Mumbai
Temperature: 21.99°C
Weather: haze

Practical 3

Demonstrate Data Cleaning and Preparation.

To clean and prepare raw data for analysis by identifying and handling missing, inconsistent, and irrelevant data, ensuring it is accurate and complete for generating meaningful insights.

- Identify data quality issues: Detect missing, inconsistent, or duplicate entries in the dataset.
    - Missing values can occur due to human error, system issues, or incomplete data collection.
    - Missing values are identified as blanks, NaNs, or nulls.
- Handle missing data: Apply appropriate techniques to address missing values (e.g., imputation, removal).
    - Imputation: Filling missing values using statistical methods like mean, median, or mode.
    - Removal: Eliminating rows or columns with excessive missing values.

Dataset used:

For the study purpose we have use the Titanic dataset which is very known dataset in the data science community. This dataset is related to voyage from Southampton England to New York, in a way of transits it met mishap and it's was recorded has one of the awful events.

Code:

```
#Practical 4:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, LabelEncoder

# Step 1: Load the Titanic Dataset

titanic = sns.load_dataset('titanic')

# Step 2: Inspect the Dataset

print("First 5 rows of the Titanic dataset:")

print(titanic.head())
```

First 5 rows of the Titanic dataset:

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

```
# Check for missing values
print("\nMissing values in each column:")
print(titanic.isnull().sum())
```

```
Missing values in each column:
survived       0
pclass         0
sex            0
age          177
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
deck         688
embark_town    2
alive          0
alone          0
dtype: int64
```

```
# Check data types
print("\nData types and basic statistics:")
print(titanic.info())
```

```
Data types and basic statistics:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4),
object(5)
memory usage: 80.7+ KB
None
```

```
# Check stastical summary
print("\n statistical Summary:")
print(titanic.describe())
# Check shape of dataset
print("\n Shape of dataset:")
print(titanic.shape)
```

statistical Summary:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Shape of dataset:
(891, 15)

```
# Step 3: Handle Missing Values
# Fill missing 'age' values with the median of the 'age' column
titanic['age'].fillna(titanic['age'].median(), inplace=True)

# Fill missing 'embarked' values with the most common value (mode)
titanic['embarked'].fillna(titanic['embarked'].mode()[0], inplace=True)

# Drop 'deck' column due to a large number of missing values
titanic.drop(columns=['deck'], inplace=True)

# Fill missing 'embark_town' with the mode
titanic['embark_town'].fillna(titanic['embark_town'].mode()[0], inplace=True)

# Drop any remaining rows with missing values
titanic.dropna(inplace=True)
print("\nMissing values after cleaning:")
print(titanic.isnull().sum())
```

Missing values after cleaning:

```
survived      0
pclass        0
sex           0
age           0
sibsp         0
parch         0
fare          0
embarked      0
class         0
who           0
adult_male    0
embark_town   0
alive         0
alone         0
dtype: int64
```

Practical 4

Implement Data wrangling on a data set.

The aim of data wrangling is to clean, structure, and enrich raw data into a usable format for analysis. This process ensures that the data is accurate, consistent, and suitable for generating insights or building machine learning models.

Data wrangling involves several systematic steps to prepare raw data for further analysis.

These steps include:

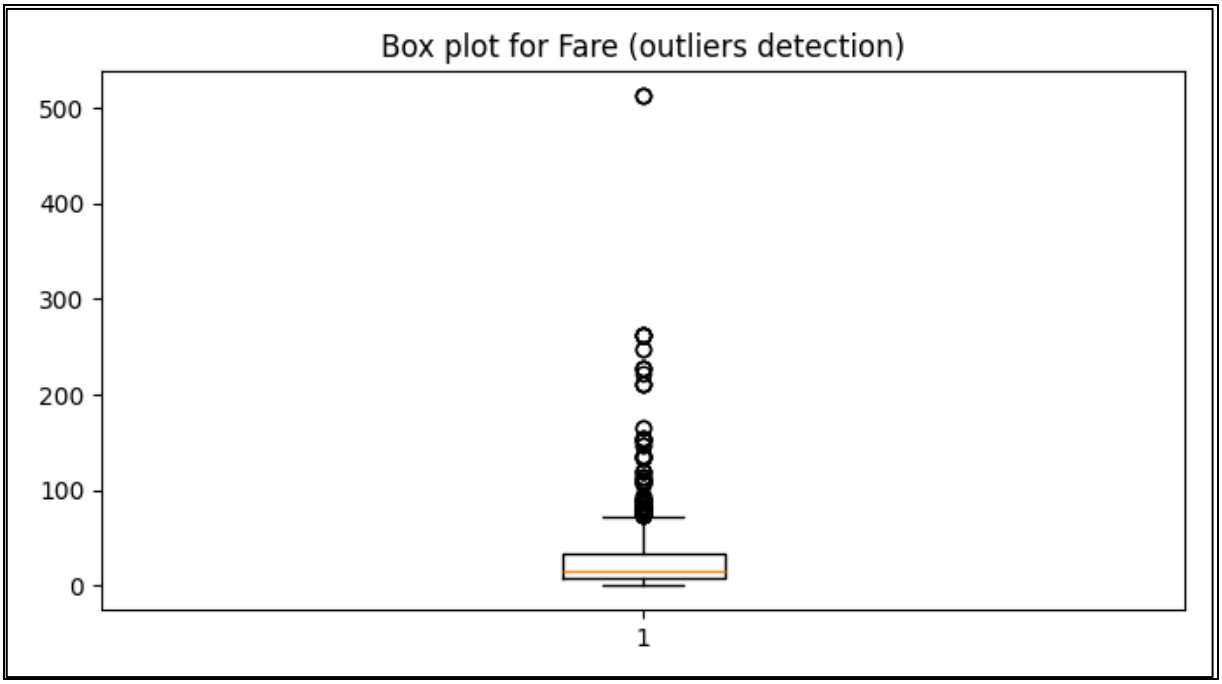
- Data Acquisition
- Exploratory Data Analysis (EDA)
- Data Cleaning
- Data Transformation
- Data Integration
- Feature Engineering
- Data Validation
- Storage and Export

A cleaned and structured dataset free from missing values, duplicates, and inconsistencies.
Enhanced data usability for statistical analysis and predictive modelling.
Increased confidence in the reliability and quality of the results derived from the data.

Code:

```
# Step 1: Handle Categorical Variables
# Convert 'sex' and 'embarked' into numerical labels using LabelEncoder
label_encoder = LabelEncoder()
titanic['sex'] = label_encoder.fit_transform(titanic['sex'])
titanic['embarked'] = label_encoder.fit_transform(titanic['embarked'])

# Convert 'who' into binary (man: 1, woman: 0)
titanic['who'] = titanic['who'].apply(lambda x: 1 if x == 'man' else 0)

# Step 2: Feature Engineering
# Create a new feature: 'family_size' = 'sibsp' + 'parch' + 1
titanic['family_size'] = titanic['sibsp'] + titanic['parch'] + 1

# Step 3: Remove Duplicates
titanic_cleaned = titanic.drop_duplicates()

# Step 4: Handle Outliers
# Inspecting outliers in the 'fare' column
plt.figure(figsize=(8,4))
plt.boxplot(titanic_cleaned['fare'])
plt.title("Box plot for Fare (outliers detection)")
plt.show()
```

Output:




```
# Cap outliers in 'fare' to the 99th percentile
fare_cap = titanic_cleaned['fare'].quantile(0.99)
titanic_cleaned['fare'] = np.where(titanic_cleaned['fare'] > fare_cap, fare_cap, titanic_cleaned['fare'])

print("\nFare column statistics after handling outliers:")
print(titanic_cleaned['fare'].describe())
```

OutPut:

Fare column statistics after handling outliers:
count 775.000000
mean 33.907613
std 45.401205
min 0.000000
25% 8.050000
50% 15.900000
75% 34.197900
max 262.375000
Name: fare, dtype: float64
/tmp/ipython-input-966917491.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
titanic_cleaned['fare'] = np.where(titanic_cleaned['fare'] > fare_cap, fare_cap, titanic_cleaned['fare'])

```
# Step 5: Normalize Numerical Data
# Normalize 'age' and 'fare' using StandardScaler
scaler = StandardScaler()
titanic_cleaned[['age', 'fare']] = scaler.fit_transform(titanic_cleaned[['age', 'fare']])
```

OutPut:

```
/tmp/ipython-input-417614019.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  titanic_cleaned[['age', 'fare']] = scaler.fit_transform(titanic_cleaned[['age', 'fare']])
```

```
print("\nFirst 5 rows of the cleaned and wrangled data:")
print(titanic_cleaned.head())
```

OutPut:

```
First 5 rows of the cleaned and wrangled data:
   survived  pclass  sex       age  sibsp  parch      fare  embarked  class  \
0         0       3    1 -0.551060      1      0 -0.587536         2  Third
1         1       1    0  0.611945      1      0  0.823763         0  First
2         1       3    0 -0.260308      0      0 -0.572659         2  Third
3         1       1    0  0.393881      1      0  0.423002         2  First
4         0       3    1  0.393881      0      0 -0.569904         2  Third

   who  adult_male  embark_town alive  alone  family_size
0    1        True  Southampton    no  False            2
1    0       False    Cherbourg   yes  False            2
2    0       False  Southampton   yes   True            1
3    0       False  Southampton   yes  False            2
4    1        True  Southampton    no   True            1
```

```
# Step 6: Save the cleaned data to a CSV file
titanic_cleaned.to_csv('titanic_cleaned.csv', index=False)
print("\nCleaned dataset saved to 'titanic_cleaned.csv'")
```

OutPut:

```
Cleaned dataset saved to 'titanic_cleaned.csv'
```

Practical 5

Demonstrate the handling of missing data and string manipulation.

To demonstrate techniques for handling missing data and string manipulation in datasets to ensure data quality and consistency.

Objectives:

- Understand the importance of handling missing data and its impact on analysis.
- Learn various methods to identify and address missing values.
- Explore string manipulation techniques to clean and preprocess text data.
- Enhance data quality for accurate and reliable insights.

Dataset used: For the demonstration purpose on how to handle missing data, along with string manipulation we have created a toy dataset frame using pandas. Next we have also imported the penguin data that comes preinstalled with Seaborn library.

Code:

```
import pandas as pd

#1. Create a Toy Dataset

data = {"Product": ["Laptop", "Tablet", "Smartphone", "Monitor", None],
   "Price": [1200, None, 800, 300, 150],
   "Category": ["Electronics", "Electronics", None, "Accessories", "Accessories"],
   "Description": ["High-end laptop", "Compact and versatile", "Feature-packed smartphone", None,
"Affordable monitor"]
}
#Create a DataFrame
df = pd.DataFrame(data)
print("Original Dataset:")
print(df)
#Handling Missing Data Fill missing Product names with "Unknown"
df["Product"].fillna("Unknown", inplace=True)
```

Original Dataset:

	Product	Price	Category	Description
0	Laptop	1200.0	Electronics	High-end laptop
1	Tablet	NaN	Electronics	Compact and versatile
2	Smartphone	800.0	None	Feature-packed smartphone
3	Monitor	300.0	Accessories	None
4	None	150.0	Accessories	Affordable monitor

```
# 2. Handling Missing Data Fill missing Product names with "Unknown"
df["Product"].fillna("Unknown", inplace=True)

#Fill missing Prices with the median price
df["Price"].fillna(df["Price"].median(), inplace=True)

#Fill missing Categories with "Miscellaneous"
df["Category"].fillna("Miscellaneous", inplace=True)

#Fill missing Descriptions with a default value
df["Description"].fillna("No description available", inplace=True)
print("\nDataset After Handling Missing Data:")
print(df)
```

Dataset After Handling Missing Data:

	Product	Price	Category	Description
0	Laptop	1200.0	Electronics	High-end laptop
1	Tablet	550.0	Electronics	Compact and versatile
2	Smartphone	800.0	Miscellaneous	Feature-packed smartphone
3	Monitor	300.0	Accessories	No description available
4	Unknown	150.0	Accessories	Affordable monitor

```
#3. String Manipulation
#Add a new column "Short Description" with the first 10 characters of the Description
df["Short Description"] = df["Description"].str[:10]

#Convert the Category column to uppercase
df["Category"] = df["Category"].str.upper()

#Replace spaces with hyphens in the Product column
df["Product"] = df["Product"].str.replace(" ", "-", regex=False)
print("\nDataset After String Manipulation:")
print(df)
```

Dataset After String Manipulation:

	Product	Price	Category	Description \
0	Laptop	1200.0	ELECTRONICS	High-end laptop
1	Tablet	550.0	ELECTRONICS	Compact and versatile
2	Smartphone	800.0	MISCELLANEOUS	Feature-packed smartphone
3	Monitor	300.0	ACCESSORIES	No description available
4	Unknown	150.0	ACCESSORIES	Affordable monitor

	Short Description
0	High-end l
1	Compact an
2	Feature-pa
3	No descrip
4	Affordable

#4. Save the Cleaned and Transformed Dataset

```
cleaned_file_path = "cleaned_data.csv"
df.to_csv(cleaned_file_path, index=False)
print(f"\nCleaned data saved as CSV at:" ,{cleaned_file_path})
```

Cleaned data saved as CSV at: {'cleaned_data.csv'}

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#Load the Penguins dataset from seaborn
penguins = sns.load_dataset("penguins")
# Display the first few rows of the dataset
print("Original Dataset:")
print(penguins.head())
```

Original Dataset:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	\
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
3	Adelie	Torgersen	NaN	NaN	NaN	
4	Adelie	Torgersen	36.7	19.3	193.0	

	body_mass_g	sex
0	3750.0	Male
1	3800.0	Female
2	3250.0	Female
3	NaN	NaN
4	3450.0	Female

```
# Exploratory Data Analysis (EDA)
 # Check dataset info
print("\nDataset Info:")
print(penguins.info())

 #Describe numerical columns
print("\nStatistical Summary of Numerical Columns:")
print(penguins.describe())

 #Check for missing values
print("\nMissing Data:")
print(penguins.isnull().sum())

 #Visualizations
 #Distribution of bill length
plt.figure(figsize=(8, 6))
sns.histplot(penguins['bill_length_mm'], kde=True, bins=20, color="blue")
plt.title("Distribution of Bill Length")
plt.xlabel("Bill Length (mm)")
plt.ylabel("Frequency")
plt.show()
```

```
#Boxplot for flipper length by species
plt.figure(figsize=(8, 6))
sns.boxplot(x="species", y="flipper_length_mm", data=penguins, palette="Set2")
plt.title("Flipper Length by Species")
plt.xlabel("Species")
plt.ylabel("Flipper Length (mm)")
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(penguins.select_dtypes('float64').corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

Dataset Info:

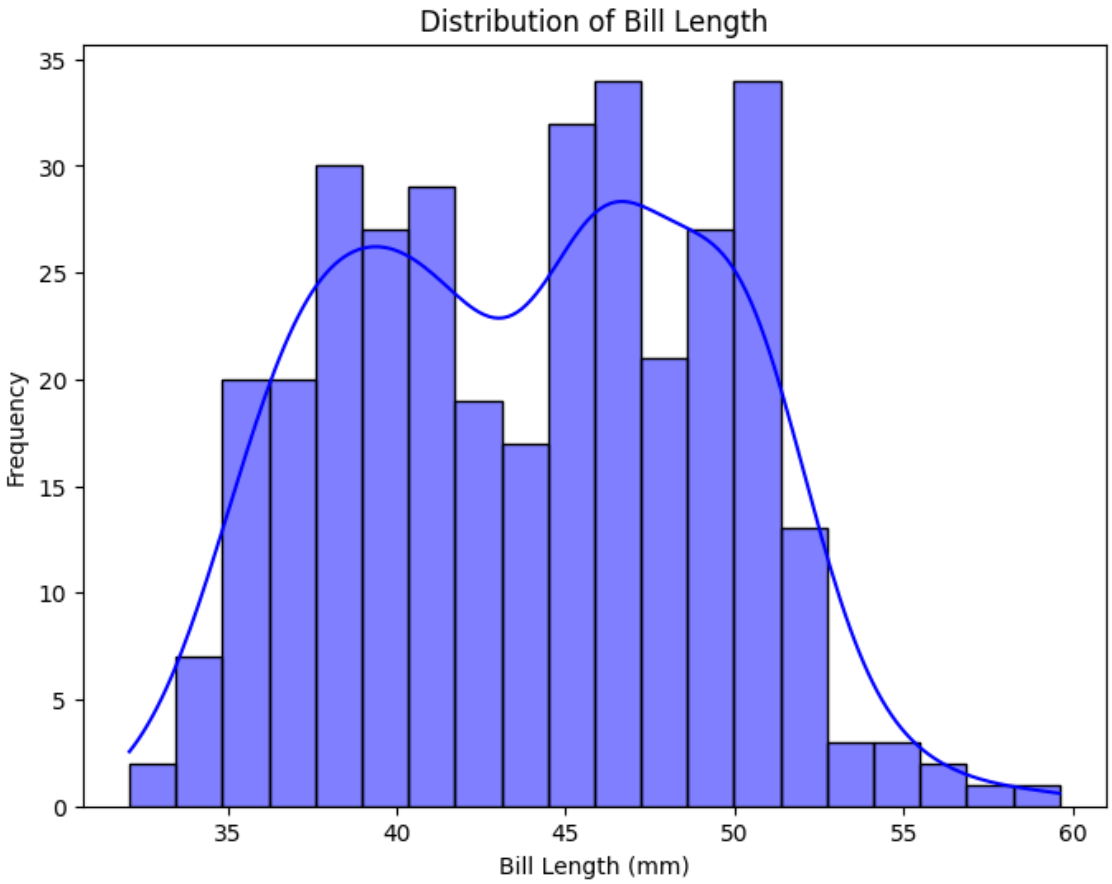
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   species           344 non-null    object
 1   island            344 non-null    object
 2   bill_length_mm    342 non-null    float64
 3   bill_depth_mm     342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g       342 non-null    float64
 6   sex               333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
None
```

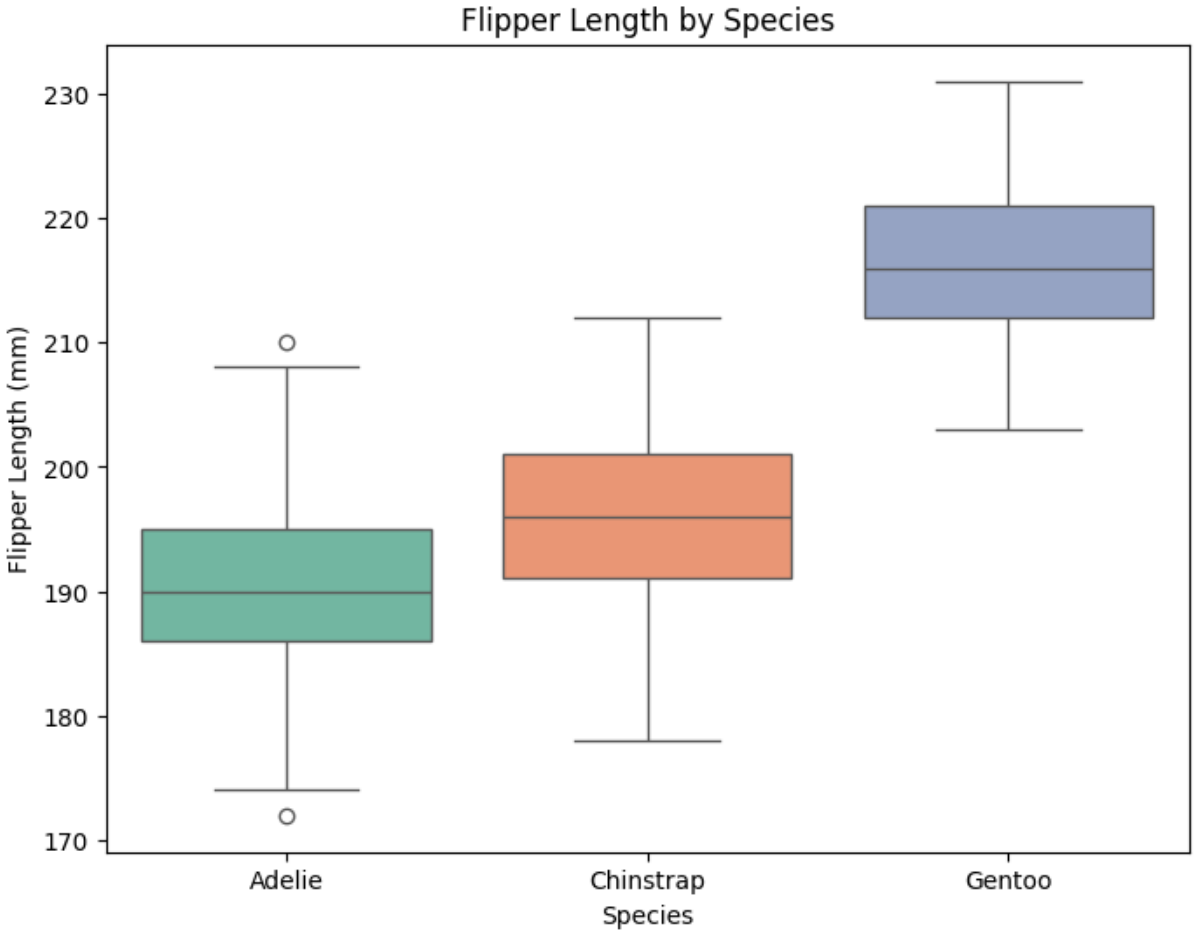
Statistical Summary of Numerical Columns:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	342.000000	342.000000	342.000000	342.000000
mean	43.921930	17.151170	200.915205	4201.754386
std	5.459584	1.974793	14.061714	801.954536
min	32.100000	13.100000	172.000000	2700.000000
25%	39.225000	15.600000	190.000000	3550.000000
50%	44.450000	17.300000	197.000000	4050.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

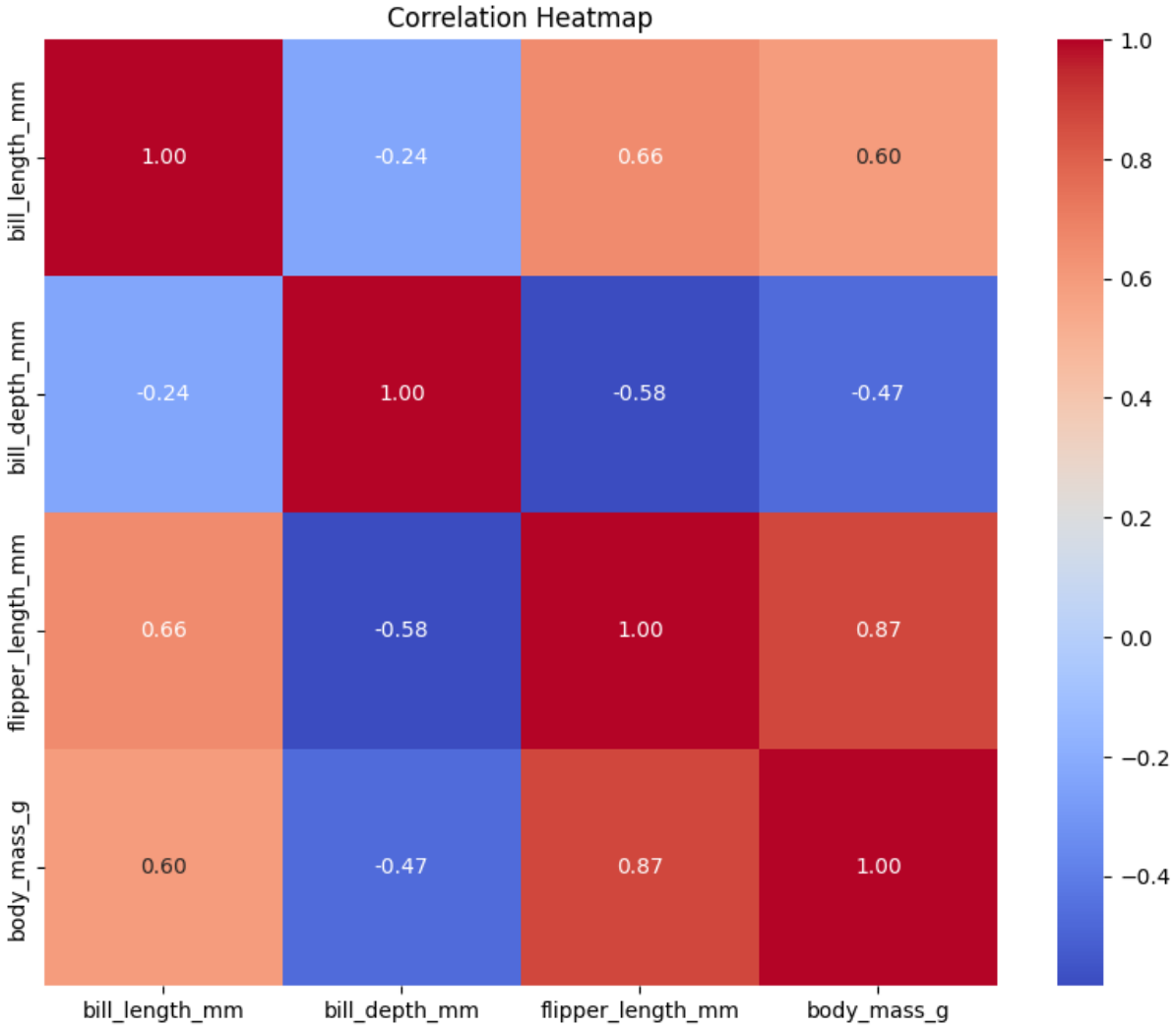
Missing Data:

```
species            0
island             0
bill_length_mm     2
bill_depth_mm      2
flipper_length_mm  2
body_mass_g        2
sex               11
dtype: int64
```


Distribution of Bill Length
Frequency
Bill Length (mm)
0
5
10
15
20
25
30
35
35
40
45
50
55
60


Flipper Length by Species
Flipper Length (mm)
170
180
190
200
210
220
230
Adelie
Chinstrap
Gentoo
Species

Output:




```
# Handle Missing Data

#Fill missing numerical values with their respective column means
penguins["bill_length_mm"].fillna(penguins["bill_length_mm"].mean(), inplace=True)
penguins["bill_depth_mm"].fillna(penguins["bill_depth_mm"].mean(), inplace=True)
penguins["flipper_length_mm"].fillna(penguins["flipper_length_mm"].mean(), inplace=True)
penguins["body_mass_g"].fillna(penguins["body_mass_g"].median(), inplace=True)

#Fill missing categorical values with "Unknown"
penguins["sex"].fillna("Unknown", inplace=True)
print("\nDataset After Handling Missing Data:")
print(penguins.isnull().sum())
```

Dataset After Handling Missing Data:

species 0
island 0
bill_length_mm 0
bill_depth_mm 0
flipper_length_mm 0
body_mass_g 0
sex 0

```
# String Manipulation

 #Create a new column by concatenating species and island
penguins["species_island"] = penguins["species"] + " - " + penguins["island"]

# Convert species names to uppercase
penguins["species"] = penguins["species"].str.upper()

# Extract the first three letters of the island name as a new column
penguins["island_code"] = penguins["island"].str[:3]
print("\nDataset After String Manipulation:")
print(penguins.head())
```

Dataset After String Manipulation:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	\
0	ADELIE	Torgersen	39.10000	18.70000	181.000000	
1	ADELIE	Torgersen	39.50000	17.40000	186.000000	
2	ADELIE	Torgersen	40.30000	18.00000	195.000000	
3	ADELIE	Torgersen	43.92193	17.15117	200.915205	
4	ADELIE	Torgersen	36.70000	19.30000	193.000000	

	body_mass_g	sex	species_island	island_code
0	3750.0	Male	Adelie - Torgersen	Tor
1	3800.0	Female	Adelie - Torgersen	Tor
2	3250.0	Female	Adelie - Torgersen	Tor
3	4050.0	Unknown	Adelie - Torgersen	Tor
4	3450.0	Female	Adelie - Torgersen	Tor

```
#Save the Cleaned and Transformed Dataset
cleaned_file_path = "cleaned_penguins.csv"
penguins.to_csv(cleaned_file_path, index=False)
print(f"\nCleaned dataset saved as CSV at: {cleaned_file_path}")
```

Cleaned dataset saved as CSV at: cleaned_penguins.csv

```
# Grouped Analysis

# Group by species and calculate mean flipper length
grouped_data = penguins.groupby("species")["flipper_length_mm"].mean().reset_index()
print("\nMean Flipper Length by Species:")
print(grouped_data)

# Save grouped data as Excel
grouped_file_path = "grouped_penguins.xlsx"
grouped_data.to_excel(grouped_file_path, index=False)
print(f"\nGrouped data saved as Excel at: {grouped_file_path}")
```

Mean Flipper Length by Species:

	species	flipper_length_mm
0	ADELIE	190.025758
1	CHINSTRAP	195.823529
2	GENTOO	217.055768

Grouped data saved as Excel at: grouped_penguins.xlsx

Practical 6

Create common charts with title, labels and descriptions using Tableau.

Theory:

There are several ways text can appear on a sheet. You can show titles and captions on any sheet. Another option is to Add tooltips to marks. A legend card appears in the worksheet when you encode marks by dropping them on the Color or Size cards. It's also possible to Add Annotations for a mark, point, or area in the view.

These text elements can be customized for text properties like size, color, alignment, and font, as well as element properties like shading and borders.

1. Bar Chart:

a) Open Tableau and connect to your data source.
b) Drag the dimension you want to represent on the x-axis to Columns.
c) Drag the measure you want to represent on the y-axis to Rows.
d) Tableau will automatically create a bar chart.
e) Customize the chart by adding titles, labels, and descriptions in the "Titles" and "Annotations" sections.

2. Line Chart:

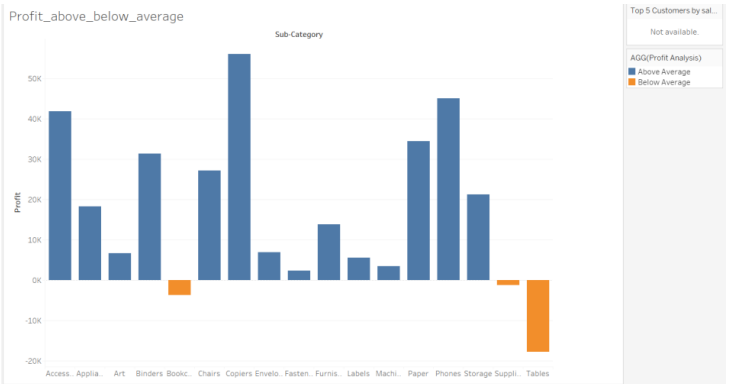
a) Follow steps 1-3 from the Bar Chart instructions.
b) Instead of a bar chart, Tableau will create a line chart.
c) You can customize the chart in the same way as the Bar Chart, adding titles, labels, and descriptions.

3. Pie Chart:

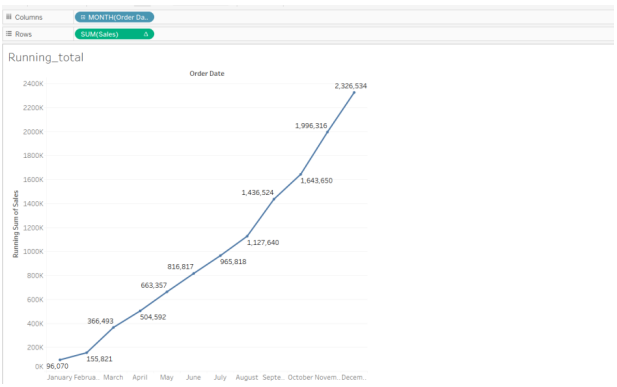
a) Follow steps 1-3 from the Bar Chart instructions.
b) Drag the dimension you want to represent on the pie slices to the "Color" shelf.
c) Tableau will automatically create a pie chart.
d) Customize the chart as needed, adding titles, labels, and descriptions.

Output:

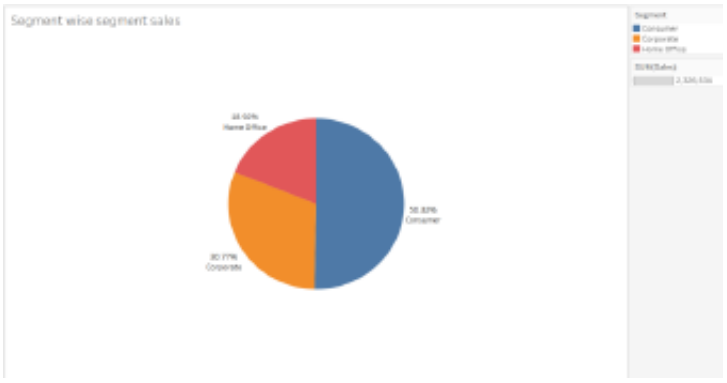
Bar Chart:



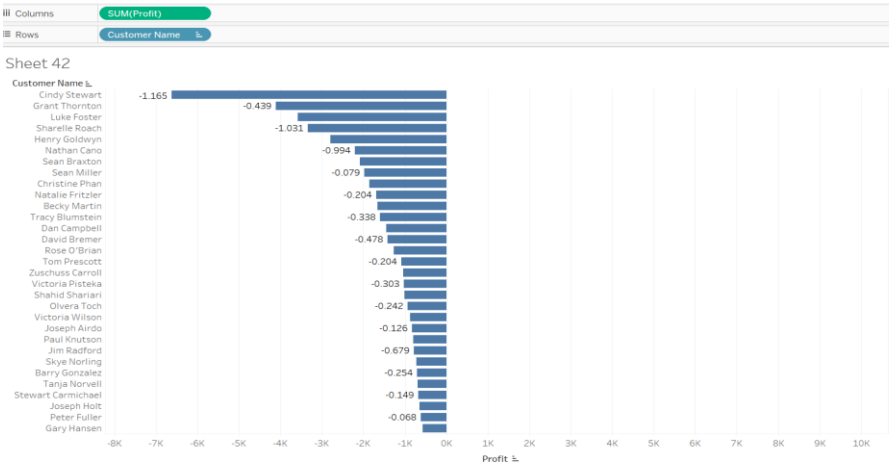
Line Chart:



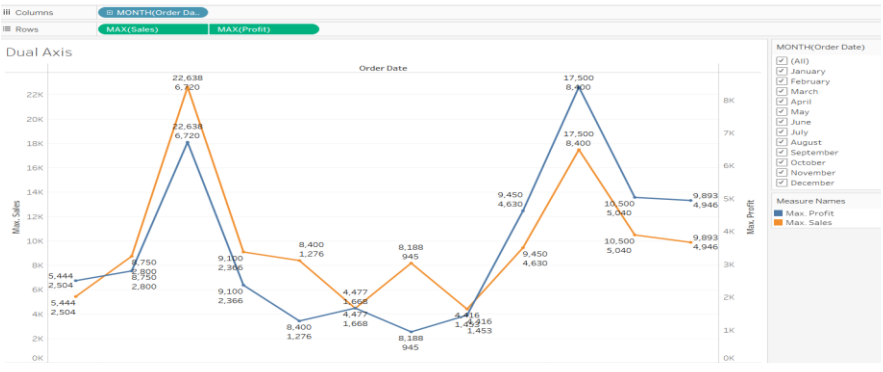
Bar Chart:



Stacked Bar Chart:



Line Chart:



Practical 7

Perform sorting and filtering using tableau, create visualizations and publish it on Tableau Cloud.

To utilize Tableau for sorting and filtering data, creating insightful visualizations, and publishing them on Tableau Cloud for collaboration and sharing.

Step 1: Prepare the Dataset

1. Load your dataset in a structured format (e.g., Excel, CSV, or a database).
2. Ensure the data is clean and properly formatted with no missing values or inconsistent types.
3. Save the dataset to your local system.

Step 2: Open Tableau Desktop

1. Launch Tableau Desktop on your computer.
2. Click on Connect to Data and choose your data source (Excel, CSV, SQL, etc.).
3. Load the dataset by selecting the file or connecting to the database.
4. Drag and drop the relevant tables into the Canvas if needed.

Step 3: Perform Sorting

1. Create a Sheet:
    - Navigate to the Sheets tab in Tableau.
    - Drag a measure (e.g., Sales, Profit) to the Rows shelf and a dimension (e.g., Region, Category) to the Columns shelf.
2. Apply Sorting:
    - Hover over the Axis or Header of the visualization (e.g., Category or Region).
    - Click the Sort button (ascending or descending order) visible on the axis.
    - Alternatively, right-click on the field in the Columns or Rows shelf and select Sort. Choose:
        - Sort Order: Ascending or Descending.
        - Sort By: Field (e.g., Profit) or Manual (drag items to reorder manually).

Step 4: Apply Filtering

1. Drag the desired dimension (e.g., Region, Product) to the Filters shelf.
2. A dialog box will appear. Choose the values you want to include or exclude and click OK.
3. To add an interactive filter:
    - Right-click on the field in the Filters shelf.
    - Select Show Filter to display a filter control on the dashboard or worksheet.
4. Customize the filter display by choosing options like:
    - Single Value (Dropdown).
    - Multi-Value (Checkbox).
    - Range of Dates or Numbers.

Step 5: Create Visualizations

1. Use the Marks Card to change the visualization type (Bar, Line, Pie, etc.).
2. Drag fields to the Columns, Rows, and Marks shelves to build your visualizations. Examples:
    - Bar Chart: Drag a measure (e.g., Sales) to Rows and a dimension (e.g., Category) to Columns.
    - Pie Chart: Select a dimension (e.g., Region) and measure (e.g., Profit) and choose the Pie Chart from the Marks dropdown.
    - Map: Use geographical fields like Country, State, or City to create maps.
3. Add Colors, Labels, and Tooltips from the Marks card to enhance the visualizations.

Step 6: Build a Dashboard

1. Click on the Dashboard tab.
2. Drag your sheets onto the dashboard canvas.
3. Customize the layout by resizing and arranging the sheets.
4. Add interactivity using filter actions:
    - Click Dashboard > Actions > Add Action > Filter.

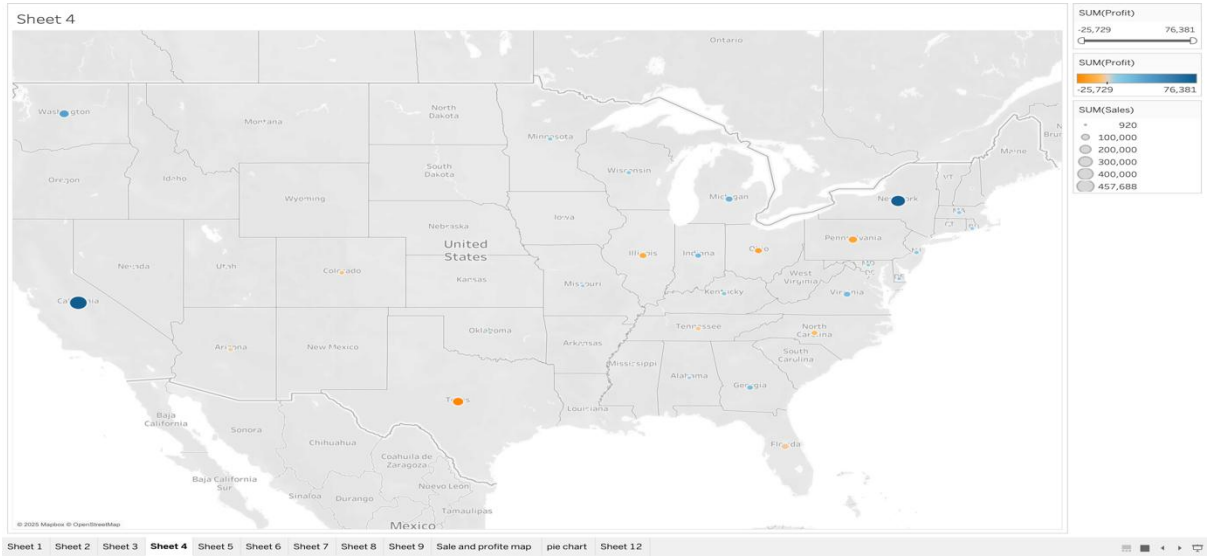
Step 7: Publish to Tableau Cloud

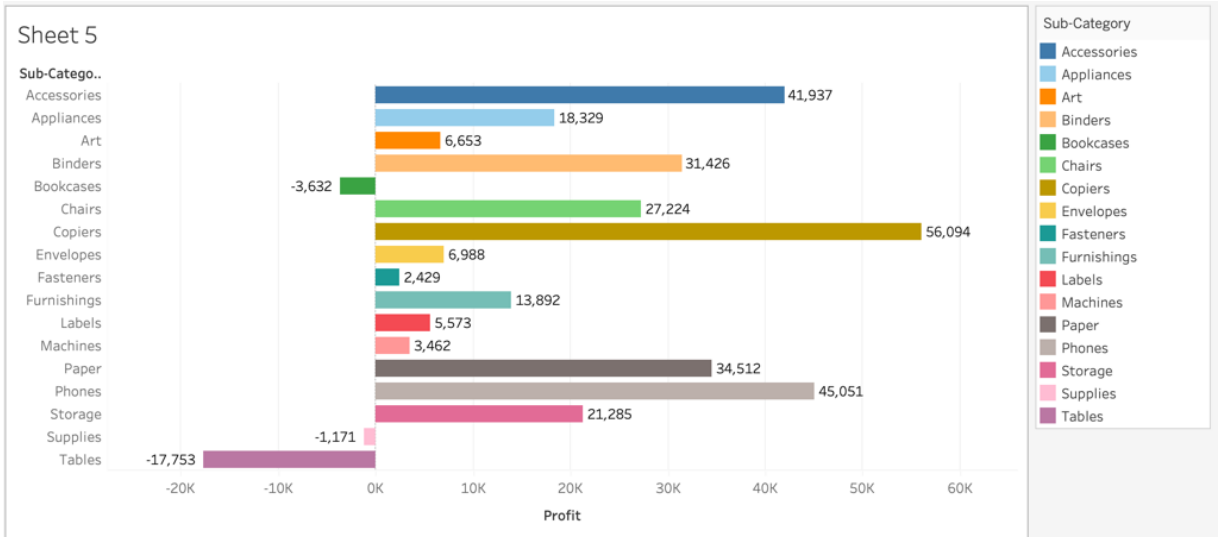
1. Ensure you have a Tableau Cloud account. If not, create one at Tableau Cloud.
2. Go to Server > Sign In in Tableau Desktop and log in with your Tableau Cloud credentials.
3. After signing in:
    - Click File > Publish to Tableau Cloud.
    - Choose the project folder where you want to save the workbook.
    - Provide a descriptive name for your workbook.
4. Adjust Permissions and Data Source Settings as needed.
5. Click Publish to upload the workbook to Tableau Cloud.

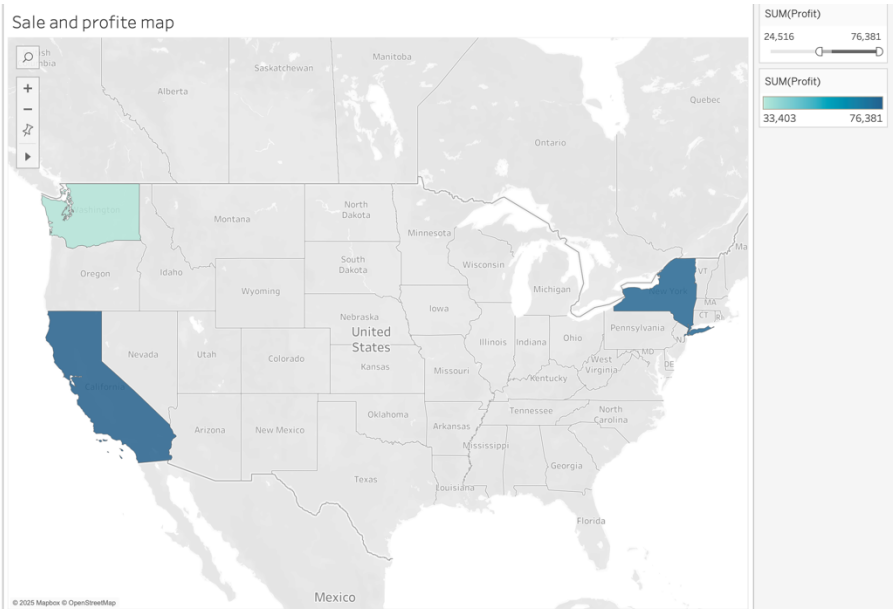
Step 8: Share and View Online

1. Log in to your Tableau Cloud account via a web browser.
2. Navigate to the published workbook.
3. Use the share link or invite others by clicking Share and sending invitations.
4. Users can view and interact with your visualizations online

Output:


Sheet 4
SUM(Profit)
-25,729
76,381
SUM(Profit)
-25,729
76,381
SUM(Sales)
920
100,000
200,000
300,000
400,000
457,688
United States
Mexico
Sheet 1
Sheet 2
Sheet 3
Sheet 4
Sheet 5
Sheet 6
Sheet 7
Sheet 8
Sheet 9
Sale and profite map
pie chart
Sheet 12


Sheet 5
Sub-Catego..
Accessories
Appliances
Art
Binders
Bookcases
Chairs
Copiers
Envelopes
Fasteners
Furnishings
Labels
Machines
Paper
Phones
Storage
Supplies
Tables
41,937
18,329
6,653
31,426
-3,632
27,224
56,094
6,988
2,429
13,892
5,573
3,462
34,512
45,051
21,285
-1,171
-17,753
-20K
-10K
0K
10K
20K
30K
40K
50K
60K
Profit
Sub-Category
Accessories
Appliances
Art
Binders
Bookcases
Chairs
Copiers
Envelopes
Fasteners
Furnishings
Labels
Machines
Paper
Phones
Storage
Supplies
Tables


Sale and profite map
SUM(Profit)
24,516
76,381
SUM(Profit)
33,403
76,381
United States
Mexico

Practical 8

Perform data visualization using Power BI.

Theory:

Power BI provides a user-friendly interface with a drag-and-drop functionality, making it easy to create compelling visualizations and reports. Adjustments can be made in real-time, and data can be refreshed to reflect the latest changes

Code:

1. Import Data:

a) Open Power BI Desktop.

b) Click on "Get Data" and select the data source you want to connect to (e.g., Excel, CSV, SQL Server, etc.).

c) Load the data into Power BI.

2. Create Visualizations:

a) In the "Visualizations" pane on the right, select the type of visualization you want to create (e.g., Bar Chart, Line Chart, Pie Chart).

b) Drag and drop the required fields from your data into the appropriate areas (e.g., Axis, Values, Legend) within the visualization.

c) Customize the visualization by adding titles, labels, and adjusting formatting in the "Format" and "Visualizations" panes.

d) Repeat the process to create multiple visualizations on the report canvas.

3. Apply Filters and Slicers:

a) Use the "Filters" pane to filter data based on specific criteria.

b) Drag the "Slicer" visualization onto the report canvas to allow users to interactively filter data.

4. Create Relationships:

a) If your data comes from multiple tables, establish relationships between them by going to the "Model" view.

b) Drag and drop fields between tables to create relationships.

5. Add Calculated Columns and Measures:

a) In the "Data" view, click on "New Column" to add calculated columns based on existing data.

b) Use the "New Measure" option to create calculated measures for aggregations or custom calculations.

6. Create a Dashboard:

a) Go to the "View" tab and click on "Report" to view your report.

b) Drag visualizations onto the "Dashboard" view to create a dashboard.

c) Arrange and resize visualizations on the dashboard canvas.

7. Publish to Power BI Service:

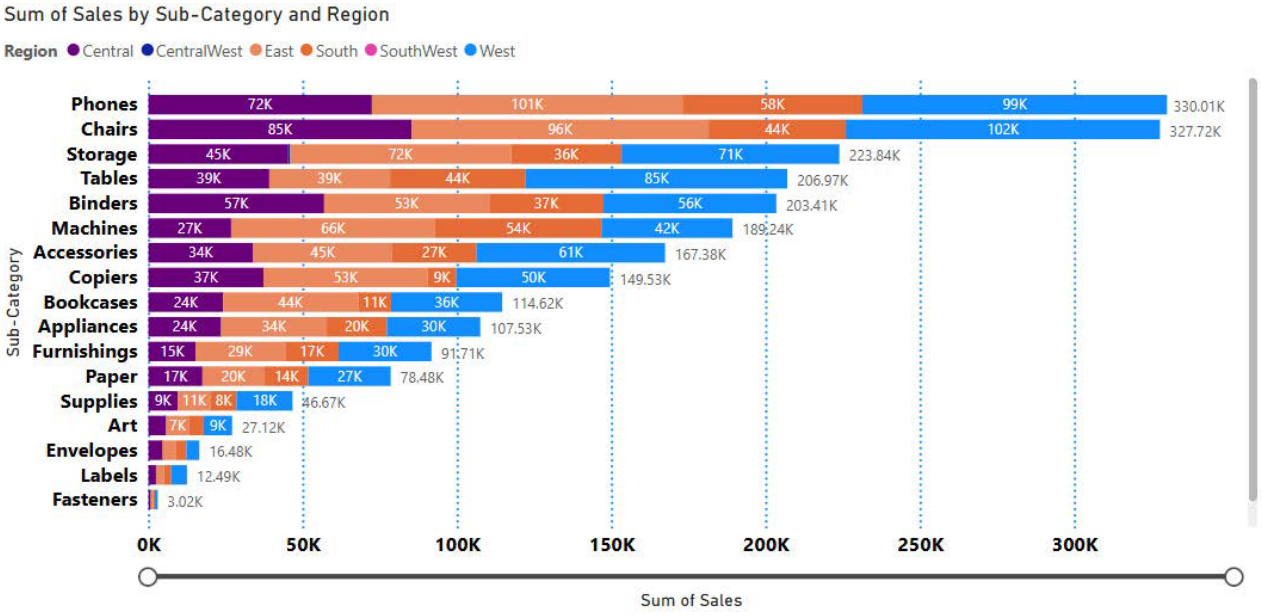
a) Save your Power BI Desktop file.

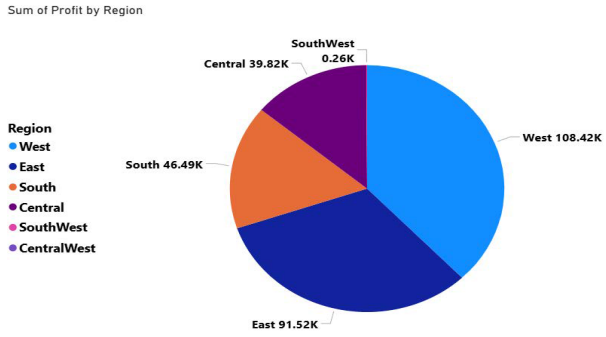
b) Click on "Publish" in the Power BI Desktop to upload your report to the Power BI Service.

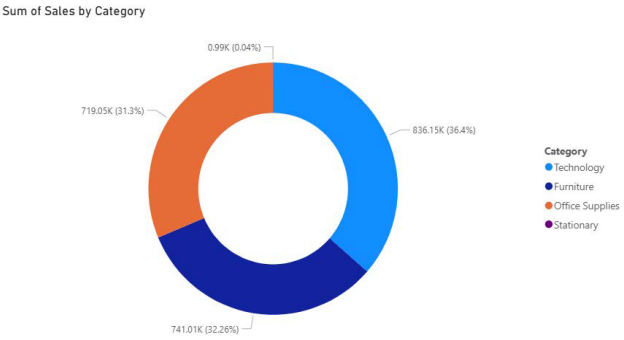
c) Log in to your Power BI Service account.

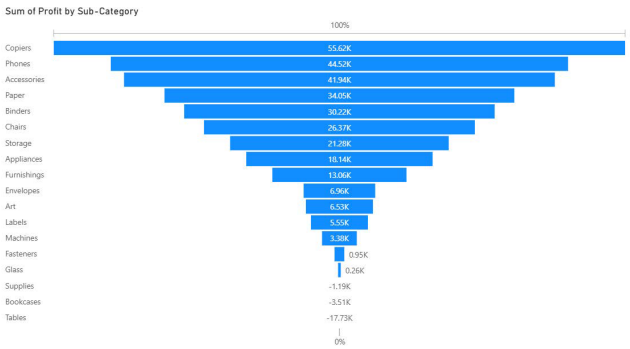
d) Once published, your report will be available on the Power BI Service.

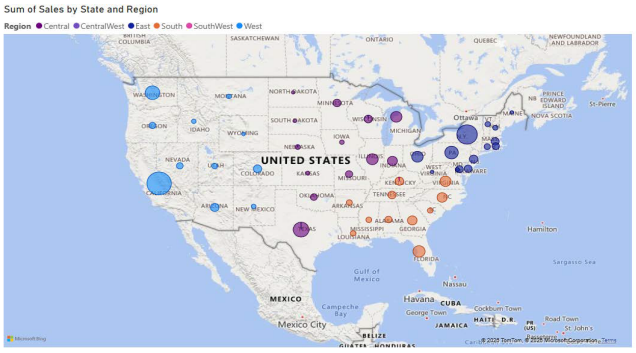
Output :

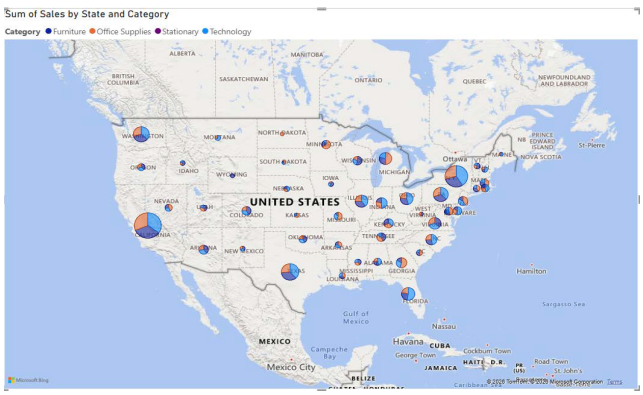

Sum of Sales by Sub-Category and Region
Region Central CentralWest East South SouthWest West
Phones 72K 101K 58K 99K 330.01K
Chairs 85K 96K 44K 102K 327.72K
Storage 45K 72K 36K 71K 223.84K
Tables 39K 39K 44K 85K 206.97K
Binders 57K 53K 37K 56K 203.41K
Machines 27K 66K 54K 42K 189.24K
Accessories 34K 45K 27K 61K 167.38K
Copiers 37K 53K 9K 50K 149.53K
Bookcases 24K 44K 11K 36K 114.62K
Appliances 24K 34K 20K 30K 107.53K
Furnishings 15K 29K 17K 30K 91.71K
Paper 17K 20K 14K 27K 78.48K
Supplies 9K 11K 8K 18K 46.67K
Art 7K 9K 27.12K
Envelopes 16.48K
Labels 12.49K
Fasteners 3.02K
Sub-Category
0K 50K 100K 150K 200K 250K 300K
Sum of Sales


Sum of Profit by Region
SouthWest 0.26K
Central 39.82K
West 108.42K
South 46.49K
East 91.52K
Region
West
East
South
Central
SouthWest
CentralWest


Sum of Sales by Category
0.99K (0.04%)
719.05K (31.3%)
836.15K (36.4%)
741.01K (32.26%)
Category
Technology
Furniture
Office Supplies
Stationary


Sum of Profit by Sub-Category
100%
Copiers 55.62K
Phones 44.52K
Accessories 41.94K
Paper 34.05K
Binders 30.22K
Chairs 26.37K
Storage 21.28K
Appliances 18.14K
Furnishings 13.06K
Envelopes 6.96K
Art 6.53K
Labels 5.55K
Machines 3.38K
Fasteners 0.95K
Glass 0.26K
Supplies -1.19K
Bookcases -3.51K
Tables -17.73K
0%


Sum of Sales by State and Region
Region Central CentralWest East South SouthWest West
UNITED STATES
MEXICO
Mexico City
Havana
CUBA
Gulf of Mexico


Sum of Sales by State and Category
Category Furniture Office Supplies Stationary Technology
UNITED STATES
MEXICO
Mexico City
Havana
CUBA
Gulf of Mexico

State

- Alabama
- Arizona
- Arkansas
- California
- Colorado
- Connecticut
- Delaware
- District of Columbia
- Florida
- Georgia
- Idaho
- Illinois
- Indiana
- Iowa
- Kansas
- Kentucky
- Louisiana
- Maine
- Maryland
- Massachusetts
- Michigan
- Minnesota
- Mississippi
- Missouri

Accessories
41,936.64
Sum of Profit

Binders
30,221.76
Sum of Profit

Copiers
55,617.82
Sum of Profit

Paper
34,053.57
Sum of Profit

Phones
44,515.73
Sum of Profit

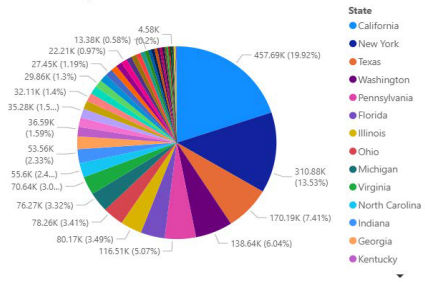
Sub-Category	Sum of Sales	Sum of Profit	Sum of Quantity
Accessories	1,67,380.32	41,936.64	2976
Appliances	1,07,532.16	18,138.01	1729
Art	27,118.79	6,527.79	3000
Binders	2,03,412.73	30,221.76	5974
Bookcases	1,14,618.04	-3,514.47	866
Chairs	3,27,717.16	26,370.58	2353
Copiers	1,49,528.03	55,617.82	234
Envelopes	16,476.40	6,964.18	906
Fasteners	3,024.28	949.52	914
Furnishings	91,705.16	13,059.14	3563
Glass	993.90	261.50	5
Labels	12,486.31	5,546.25	1400
Machines	1,89,238.63	3,384.76	440
Paper	78,479.21	34,053.57	5178
Phones	3,30,007.05	44,515.73	3289
Storage	2,23,843.61	21,278.83	3158
Supplies	46,673.54	-1,189.10	647
Tables	2,06,965.53	-17,725.48	1241
Total	**22,97,200.86**	**2,86,397.02**	**37873**

2.30M
Sum of Sales

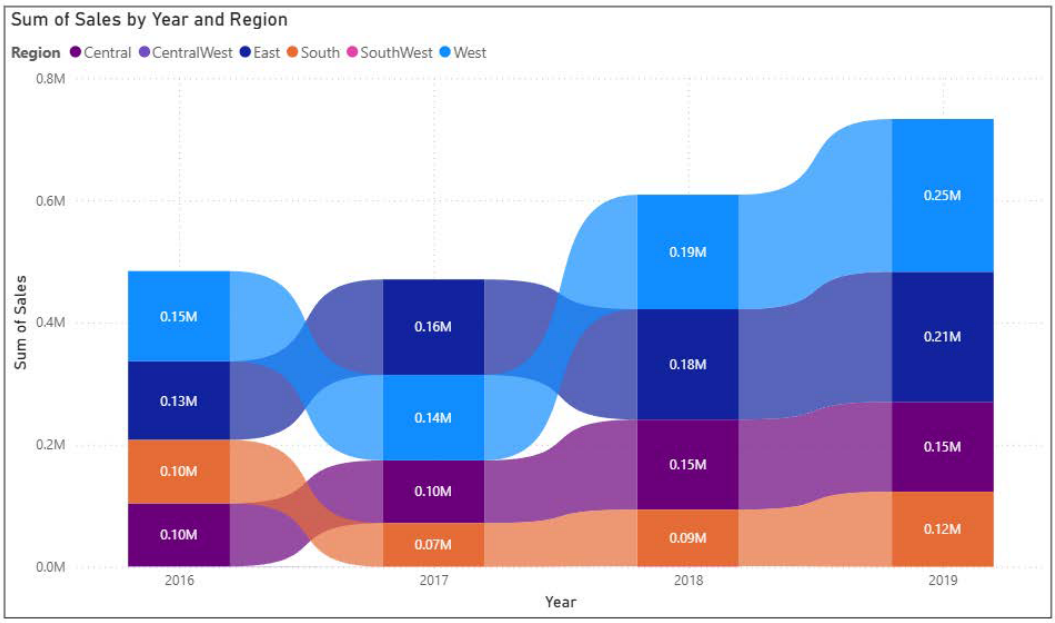
286.40K
Sum of Profit

38K
Sum of Quantity

Sum of Sales by State



Category	Central	CentralWest	East	South	SouthWest	West	Total
Furniture	**1,63,797.16**		**2,08,291.20**	**1,16,304.78**		**2,52,612.74**	**7,41,005.90**
Bookcases	24,157.18		43,819.33	10,637.40		36,004.12	**1,14,618.04**
Chairs	85,230.65		96,260.68	44,444.51		1,01,781.33	**3,27,717.16**
Furnishings	15,254.37		29,071.38	17,306.68		30,072.73	**91,705.16**
Tables	39,154.97		39,139.81	43,916.19		84,754.56	**2,06,965.53**
Office Supplies	**1,66,269.72**	**756.69**	**2,05,516.05**	**1,25,651.31**		**2,20,853.25**	**7,19,047.03**
Appliances	23,513.22	68.81	34,188.47	19,525.33		30,236.34	**1,07,532.16**
Art	5,745.88	19.46	7,485.76	4,655.62		9,212.07	**27,118.79**
Binders	56,920.74	2.54	53,498.00	37,030.34		55,961.11	**2,03,412.73**
Envelopes	4,636.87		4,375.87	3,345.56		4,118.10	**16,476.40**
Fasteners	778.03		819.72	503.32		923.22	**3,024.28**
Labels	2,451.47		2,602.93	2,353.18		5,078.73	**12,486.31**
Paper	17,491.90		20,172.60	14,150.98		26,663.72	**78,479.21**
Storage	45,264.23	665.88	71,612.58	35,768.06		70,532.85	**2,23,843.61**
Supplies	9,467.37		10,760.12	8,318.93		18,127.12	**46,673.54**
Stationary					**993.90**		**993.90**
Technology	**1,70,416.31**		**2,64,973.98**	**1,48,771.91**		**2,51,991.83**	**8,36,154.03**
Total	**5,00,483.20**	**756.69**	**6,78,781.24**	**3,90,728.00**	**993.90**	**7,25,457.82**	**22,97,200.86**



Practical 9

Create reports using Power BI.

Theory:
A Power BI report is a multi-perspective view into a semantic model, with visuals that represent findings and insights from that semantic model. A report can have a single visual or many pages full of visuals. Depending on your job role, you might be someone who designs reports, or you might be a business user who consumes reports.

Step 1: Import Data
a) Open Power BI Desktop.
b) Click on "Get Data" and choose the data source you want to connect to (e.g., Excel, CSV, SQL Server).
c) Select the data you want to import and click "Load."

Step 2: Create Visualizations
a) In the "Visualizations" pane on the right, choose the type of visualization you want to create (e.g., Bar Chart, Line Chart, Table).
b) Drag and drop fields from your dataset into the appropriate areas of the visualization (e.g., Axis, Values, Legend).
c) Customize the visualization using the formatting options and the "Visualizations" pane.
d) Repeat the process to create multiple visualizations on the report canvas.

Step 3: Apply Filters and Slicers
a) Use the "Filters" pane to add filters to your visualizations.
b) Drag the "Slicer" visualization onto the canvas to allow users to interactively filter data.

Step 4: Create Relationships
a) Go to the "Model" view by clicking on the "Model" icon on the left sidebar.
b) Establish relationships between tables by dragging and dropping fields between them**.**

Step 5: Add Calculated Columns and Measures
a) In the "Data" view, click on "New Column" to add calculated columns based on existing data.
b) Use the "New Measure" option to create calculated measures for aggregations or custom calculations.

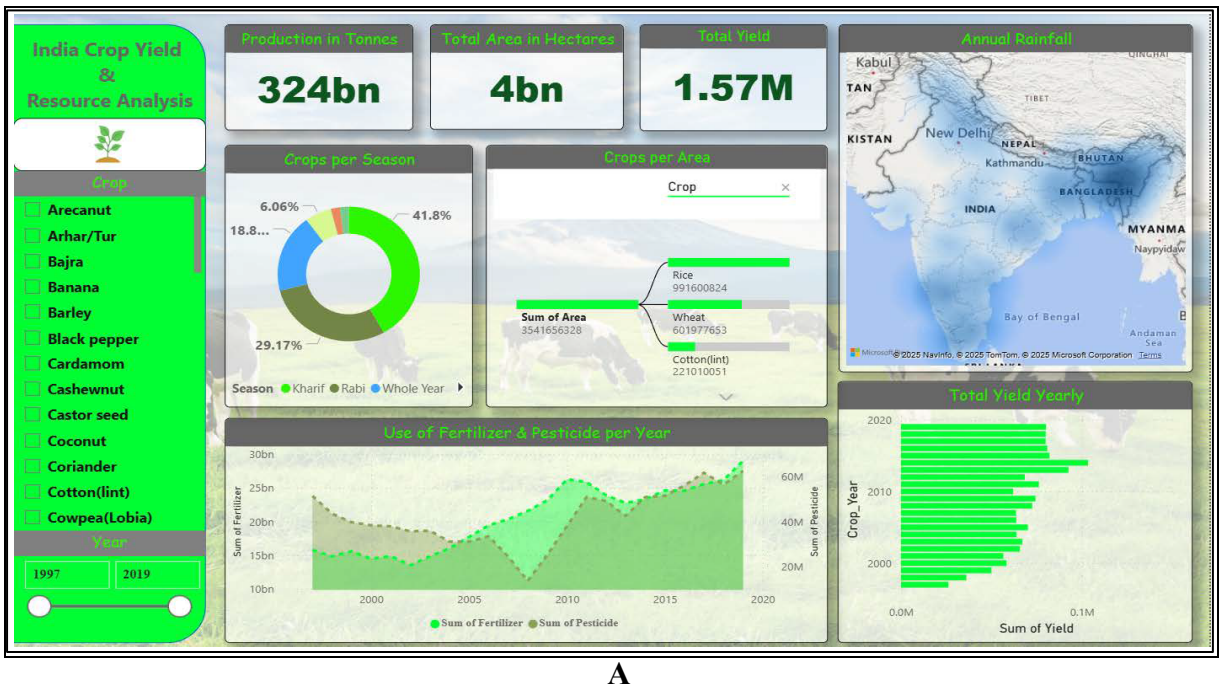
Step 6: Create a Dashboard
a) Go back to the "Report" view.
b) Drag visualizations onto the "Dashboard" view at the bottom to create a dashboard.
c) Arrange and resize visualizations on the dashboard canvas.

Step 7: Save and Publish
a) Save your Power BI Desktop file.
b) Click on "Publish" in Power BI Desktop.
c) Log in to your Power BI Service account.
d) Select the workspace where you want to publish the report.

Step 8: Share and Collaborate
a) Share your report by creating a sharing link or inviting collaborators.
b) Access your reports on the Power BI Service through a web browser or mobile app.
c) Collaborate with team members by allowing them to edit or view the report

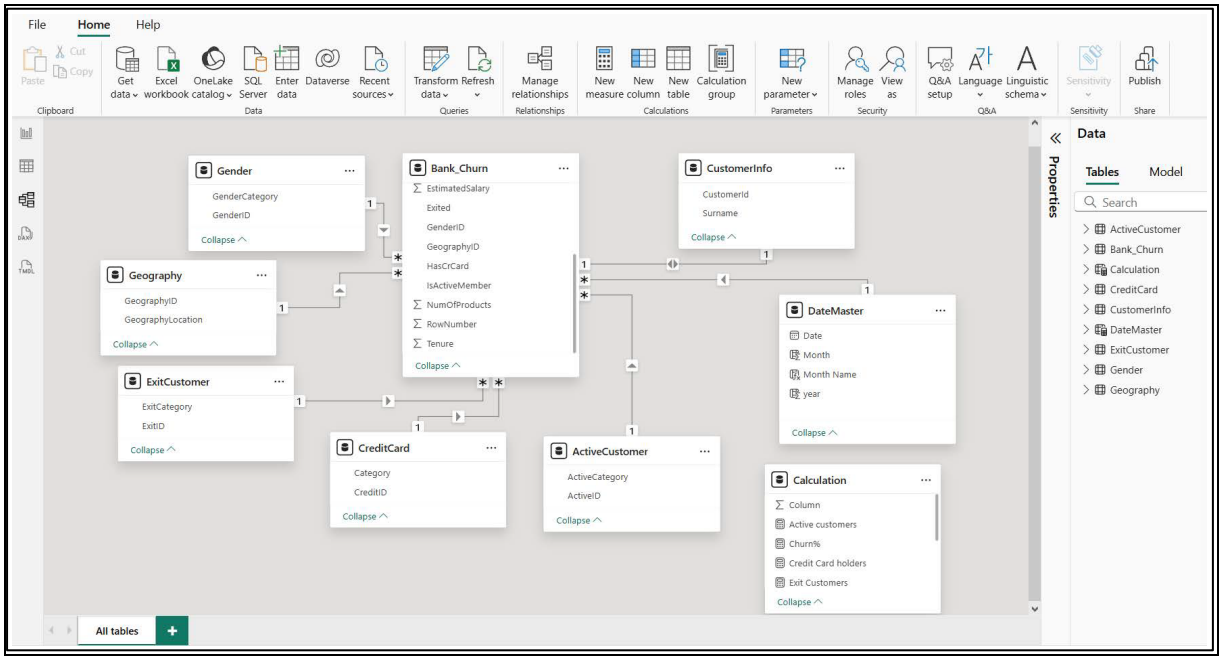
Output:

Report - India Crop Yield & Resource Analysis



A

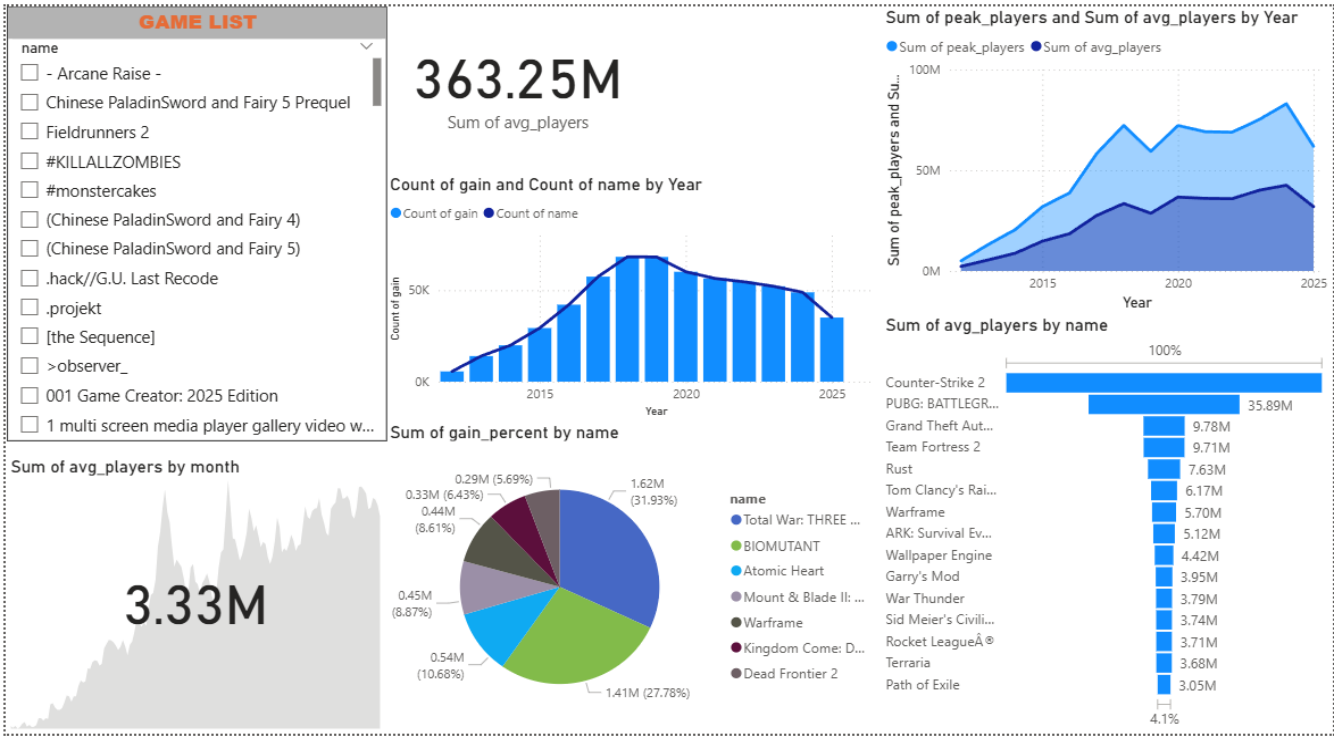
Model - alignment of tables



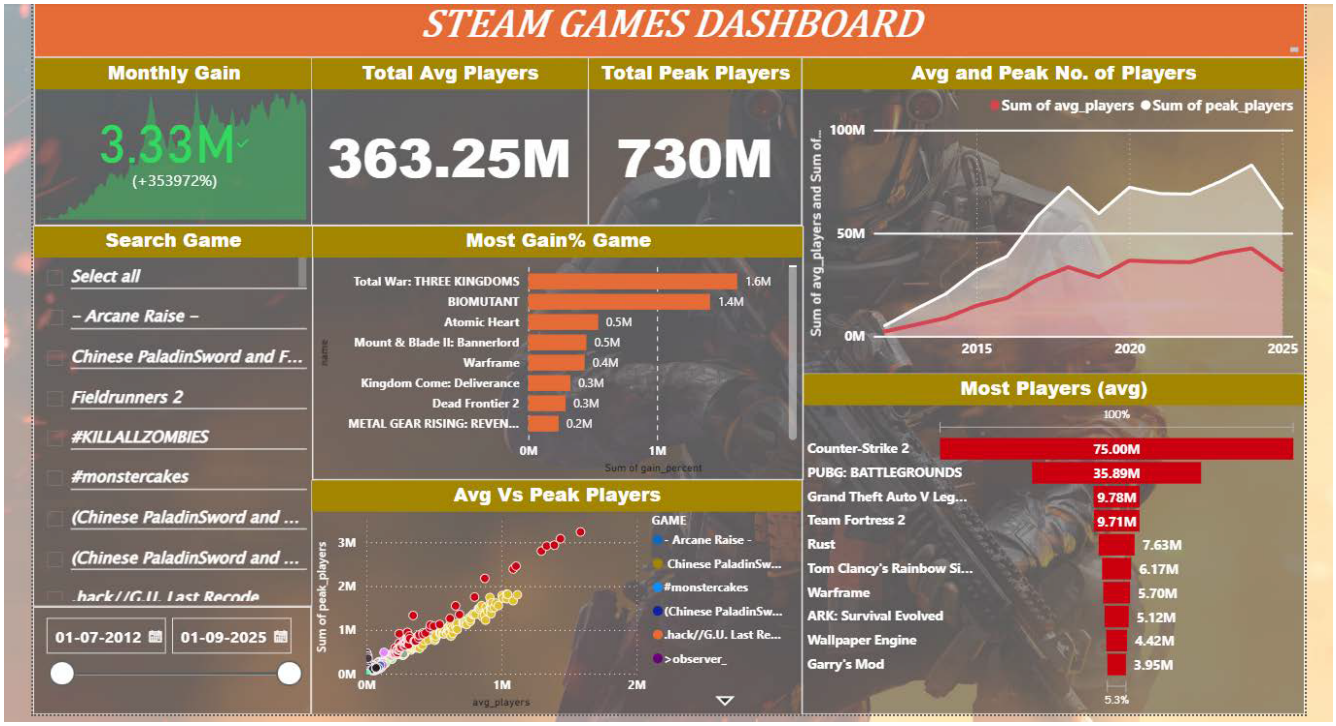
B

Output:

Raw data type



Final Output:



Practical 10

Create a data story in Tableau or power BI.

Theory : To create a compelling data story that provides actionable insights into a business problem or phenomenon, using data visualization to simplify complex information and facilitate data-driven decision-making.

Objective:
The main objectives of this data story are:

1. Identify and Understand Trends: Analyze the dataset to uncover key patterns or trends.
2. Provide Insightful Visualizations: Represent data visually to enhance understanding and retention.
3. Support Decision-Making: Equip stakeholders with the information needed to make informed decisions.
4. Engage the Audience: Use storytelling techniques to make the data narrative engaging and memorable.

STEPS:

1. Load the Dataset
• Open Tableau Desktop.
• In the Connect pane on the left side, select the required data source such as Excel, CSV, or SQL.
• Browse and select the dataset (for example, COVID-19 dataset).
• Click on Open and then Load to import the data into Tableau.
• Drag the data source into the canvas area.
• Preview the rows and columns to ensure important fields such as Date, Country, Cases, Deaths, and Vaccinations are available.
52

2. Prepare the Data
• Verify the data types of all fields.
• Set Date fields as Date.
• Set numeric fields such as Cases and Deaths as Number (Decimal).
• Assign geographic roles to fields such as Country or State:
o Right-click the field
o Select Geographic Role → Country/Region
• If multiple datasets are used, join them using common keys such as Country and Date.
• Create calculated fields if required, such as:
o Case Fatality Rate = (Deaths / Cases) × 100

3. Build Visualizations:
Time-Series Chart (Cases and Deaths Over Time)
- **Open a new worksheet.**
- **Drag Date to the Columns shelf.**
- **Drag Cases to the Rows shelf.**
- **Click the drop-down arrow on Date and select Exact Date or Month.**
- **Drag Deaths to the Rows shelf.**
- **Right-click on the axis and select Dual Axis.**
- **Synchronize the axes by right-clicking the secondary axis and selecting Synchronize Axis.**
- **Format the lines using different colors for cases and deaths.**

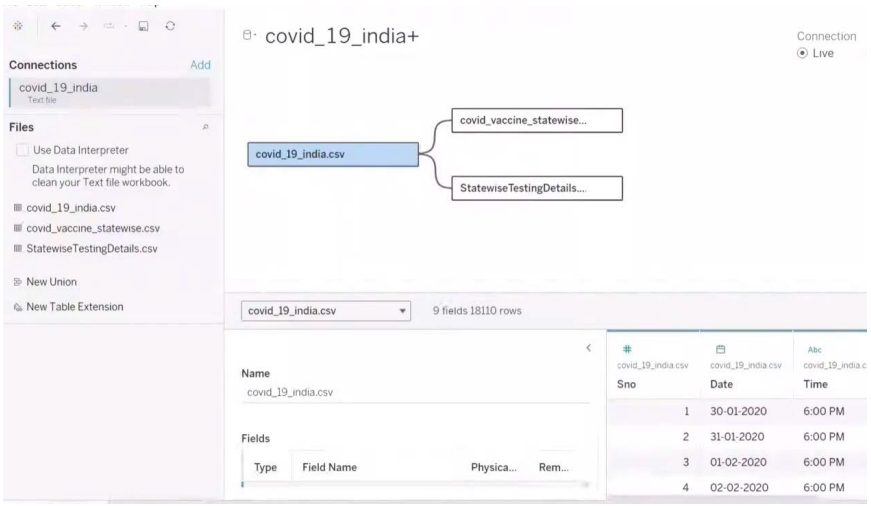
4. Create Additional Visuals
- **Create bar charts to compare cases or deaths by country or region.**
- **Create maps using geographic fields to show the spread of cases.**
- **Add KPI cards to highlight key metrics such as total cases, total deaths, or vaccination rate.**
- **Add titles, labels, legends, and tooltips to improve readability.**

5. Combine Visualizations into a Dashboard
- **Click on the Dashboard tab at the bottom.**
- **Create a new dashboard.**
- **Drag all the created worksheets (charts, maps, KPIs) onto the dashboard canvas.**
- **Use horizontal and vertical containers to organize the layout properly.**
- **Add meaningful titles and headings for each section of the dashboard.**

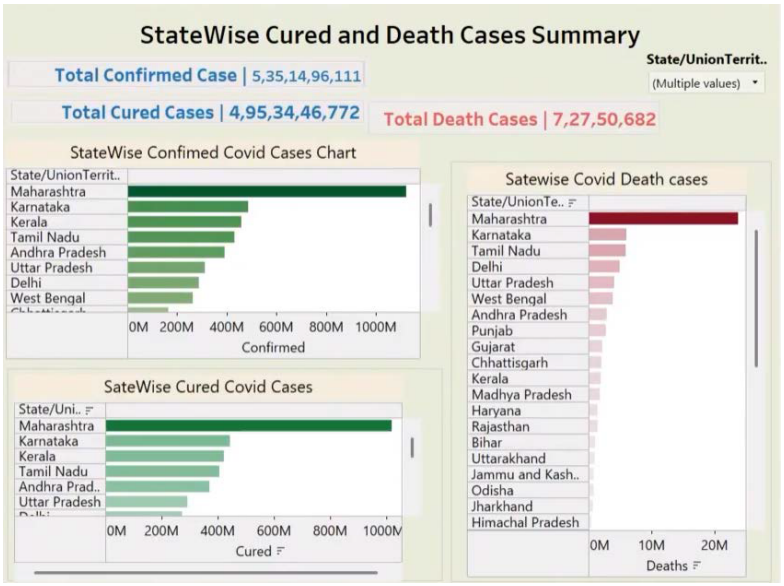
6. Add Interactivity
- **Drag filters such as Country or Date Range onto the dashboard.**
- **Enable interactive filtering by clicking Use as Filter on charts.**
- **Allow users to explore data dynamically by selecting different regions or time periods.**

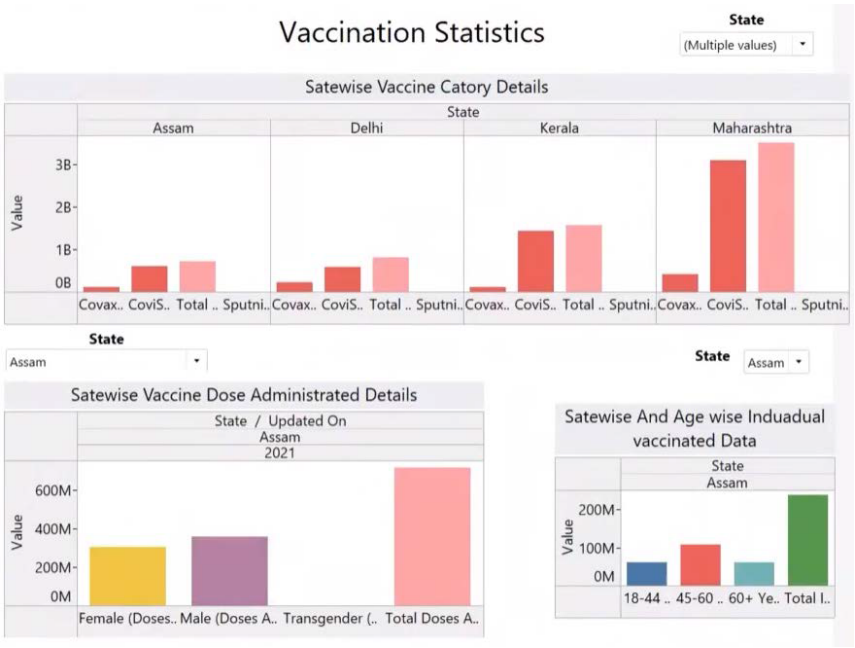
7. Publish and Share the Data Story
- **Save the Tableau workbook.**
- **Go to File → Publish Workbook.**
- **Select Tableau Public or Tableau Server as the destination.**
- **Publish the workbook.**
- **Share the dashboard link with stakeholders.**
- **Export the data story as PDF or PowerPoint if required.**

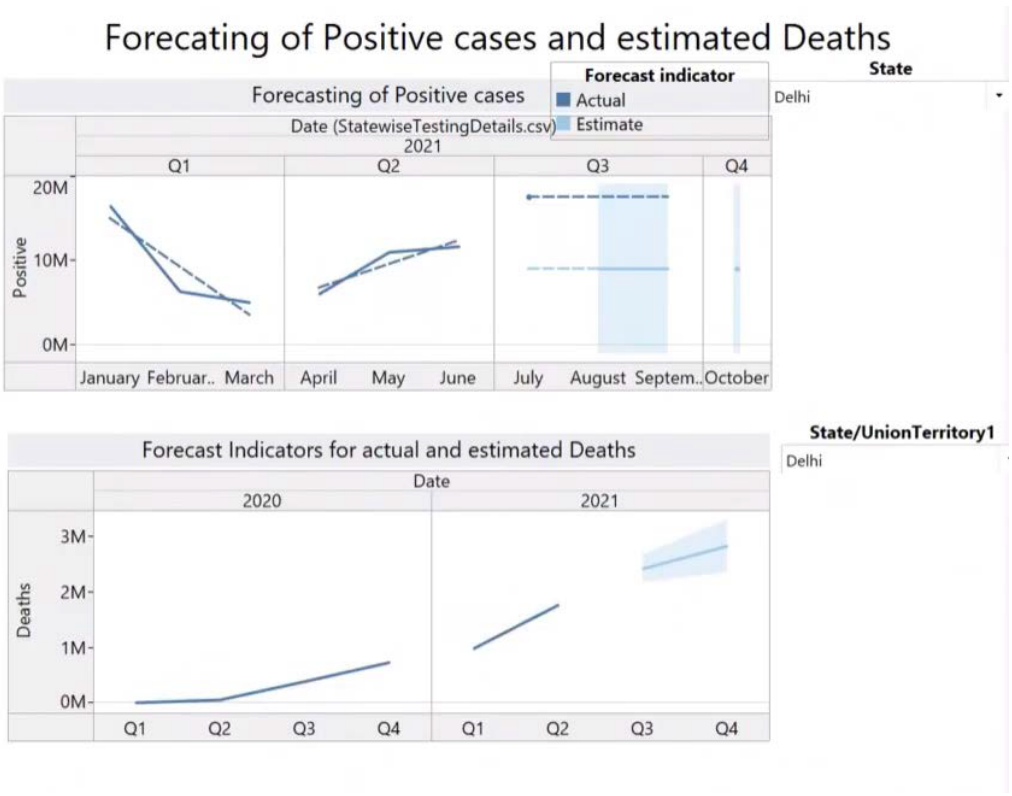
OUTPUT:

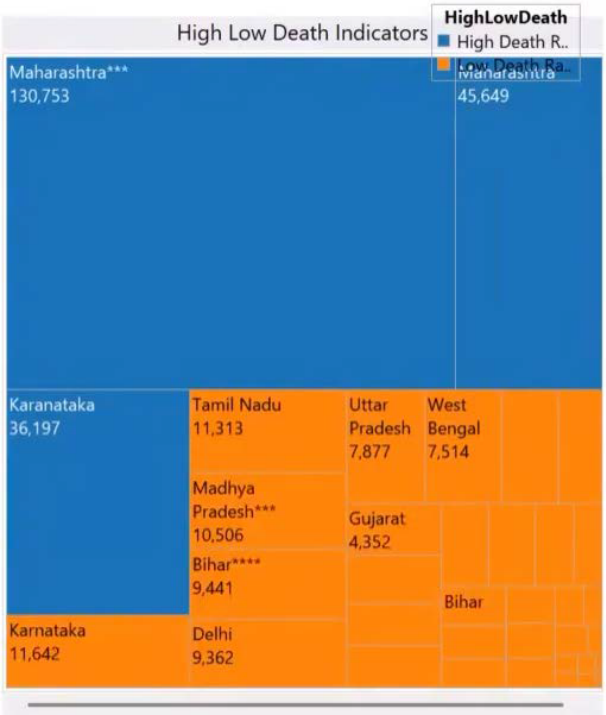


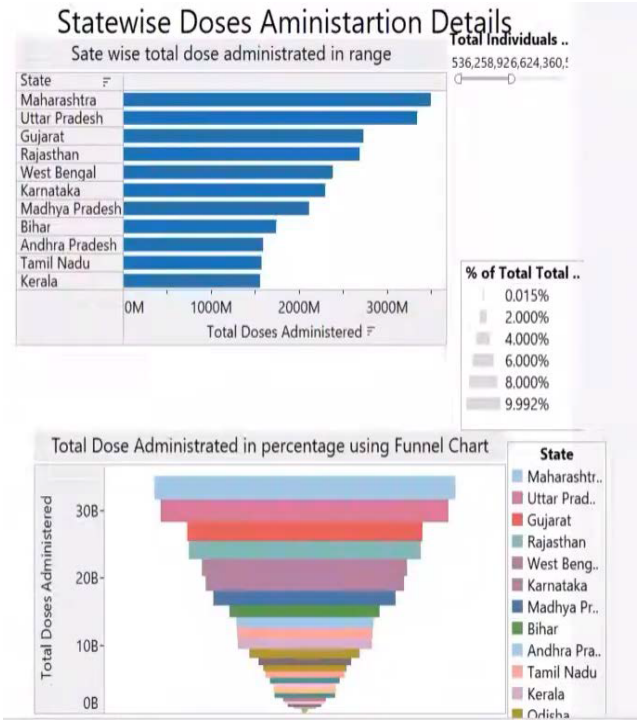
Dashboard s:



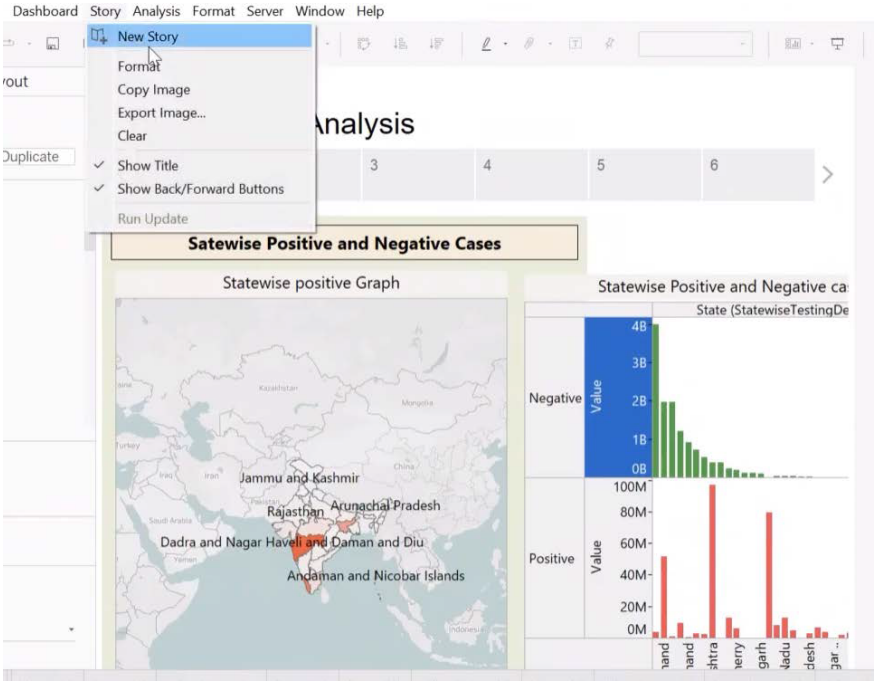



Forecating of Positive cases and estimated Deaths
Forecasting of Positive cases
Forecast indicator
Actual
Estimate
State
Delhi
Date (StatewiseTestingDetails.csv)
2021
Q1
Q2
Q3
Q4
20M
10M
0M
Positive
January
Februar..
March
April
May
June
July
August
Septem..
October
Forecast Indicators for actual and estimated Deaths
State/UnionTerritory1
Delhi
Date
2020
2021
3M
2M
1M
0M
Deaths


High Low Death Indicators
HighLowDeath
High Death R..
Low Death Ra..
Maharashtra***
130,753
45,649
Karanataka
36,197
Tamil Nadu
11,313
Uttar Pradesh
7,877
West Bengal
7,514
Madhya Pradesh***
10,506
Gujarat
4,352
Bihar****
9,441
Bihar
Karnataka
11,642
Delhi
9,362


Statewise Doses Aministartion Details
Sate wise total dose administrated in range
Total Individuals ..
536,258,926,624,360,
State
Maharashtra
Uttar Pradesh
Gujarat
Rajasthan
West Bengal
Karnataka
Madhya Pradesh
Bihar
Andhra Pradesh
Tamil Nadu
Kerala
0M
1000M
2000M
3000M
Total Doses Administered
% of Total Total ..
0.015%
2.000%
4.000%
6.000%
8.000%
9.992%
Total Dose Administrated in percentage using Funnel Chart
Total Doses Administered
30B
20B
10B
0B
State
Maharashtr..
Uttar Prad..
Gujarat
Rajasthan
West Beng..
Karnataka
Madhya Pr..
Bihar
Andhra Pra..
Tamil Nadu
Kerala
Odisha

Create New Story :



Dashboard are in Story Mode: