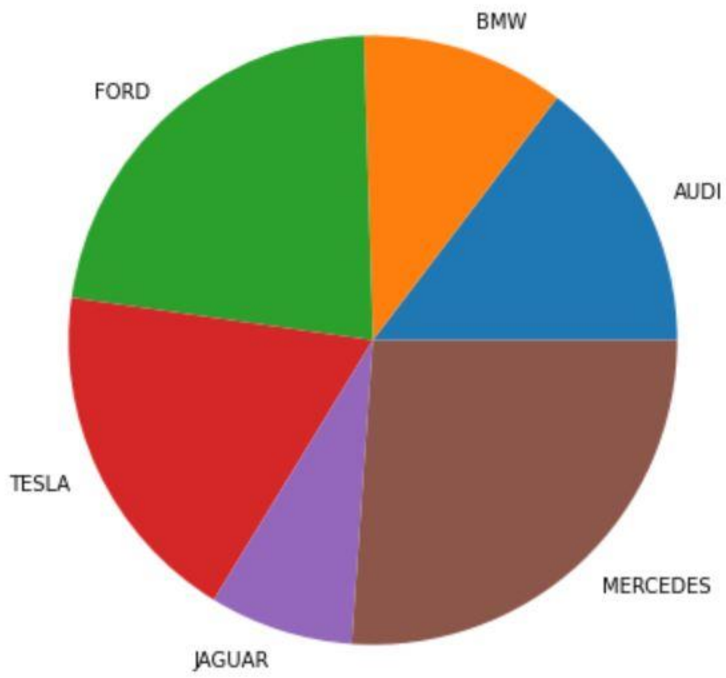
Data Handling and Data Visualization Lab Manual

1. Acquiring and plotting data:

a) Write a program to acquire data from data set and plot a pie chart.

```
# Import libraries
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
data = [23, 17, 35, 29, 12, 41]
# Creating plot
fig = plt.figure(figsize =(10, 7))
plt.pie(data, labels = cars)
# show plot
plt.show()
```

Output:

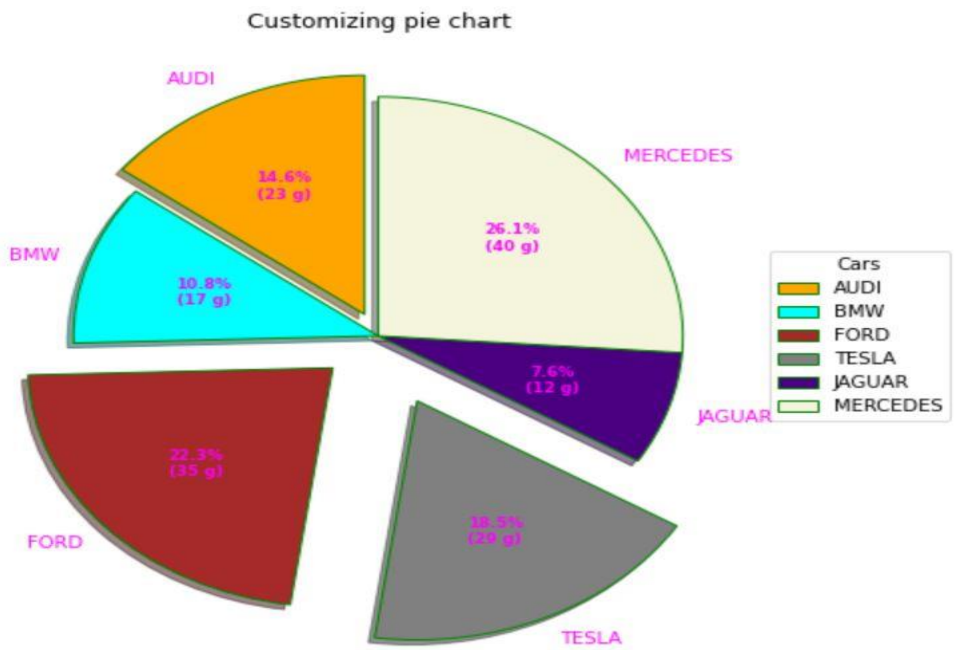


b) Write a program to acquire data from data set and plot a pie chart using explode data.

```
#Import libraries
import numpy as np
import matplotlib.pyplot as plt
# Creating dataset
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
data = [23, 17, 35, 29, 12, 41]
```

```
        # Creating explode data
        explode = (0.1, 0.0, 0.2, 0.3, 0.0, 0.0)
# Creating color parameters
colors = ( "orange", "cyan", "brown", "grey", "indigo", "beige")
# Wedge properties
wp = { 'linewidth' : 1, 'edgecolor' : "green" }
# Creating autocpt arguments
def func(pct, allvalues):
   absolute = int(pct / 100.*np.sum(allvalues))
   return "{:.1f}%\n({:d} g)".format(pct, absolute)
# Creating plot
fig, ax = plt.subplots(figsize =(10, 7))
wedges, texts, autotexts = ax.pie(data,
                                  autopct = lambda pct: func(pct, data),
                                  explode = explode,
                                  labels = cars,
                                  shadow = True,
                                  colors = colors,
                                  startangle = 90,
                                  wedgeprops = wp,
                                  textprops = dict(color ="magenta"))
# Adding legend
ax.legend(wedges, cars,
          title ="Cars",
          loc ="center left",
          bbox_to_anchor =(1, 0, 0.5, 1))
plt.setp(autotexts, size = 8, weight ="bold")
ax.set_title("Customizing pie chart")
# show plot
plt.show()
```

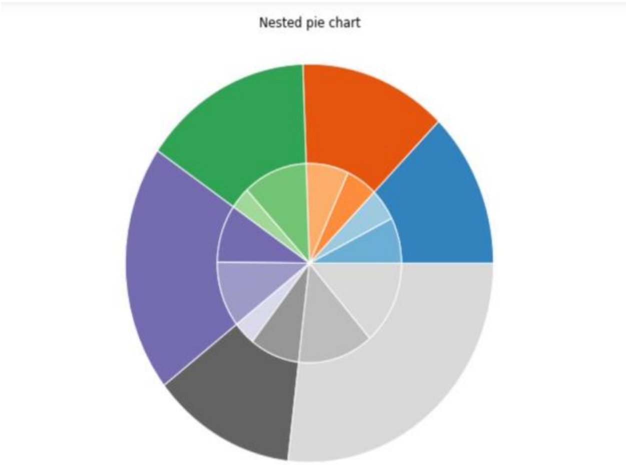
Output:



c) **Write a program to acquire data from data set and plot a pie chart using nested pie chart.**

```
# Import libraries
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
size = 6
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
data = np.array([[23, 16], [17, 23],
                 [35, 11], [29, 33],
                 [12, 27], [41, 42]])
# normalizing data to 2 pi
norm = data / np.sum(data)*2 * np.pi
# obtaining ordinates of bar edges
left = np.cumsum(np.append(0,
                           norm.flatten()[:-1])).reshape(data.shape)
# Creating color scale
cmap = plt.get_cmap("tab20c")
outer_colors = cmap(np.arange(6)*4)
inner_colors = cmap(np.array([1, 2, 5, 6, 9,
                              10, 12, 13, 15,
                              17, 18, 20 ]))
# Creating plot
fig, ax = plt.subplots(figsize =(10, 7),
                       subplot_kw = dict(polar = True))
ax.bar(x = left[:, 0],
    width = norm.sum(axis = 1),
    bottom = 1-size,
    height = size,
    color = outer_colors,
    edgecolor ='w',
    linewidth = 1,
    align ="edge")
ax.bar(x = left.flatten(),
    width = norm.flatten(),
    bottom = 1-2 * size,
    height = size,
    color = inner_colors,
    edgecolor ='w',
    linewidth = 1,
    align ="edge")
ax.set(title ="Nested pie chart")
ax.set_axis_off()
# show plot
plt.show()
```

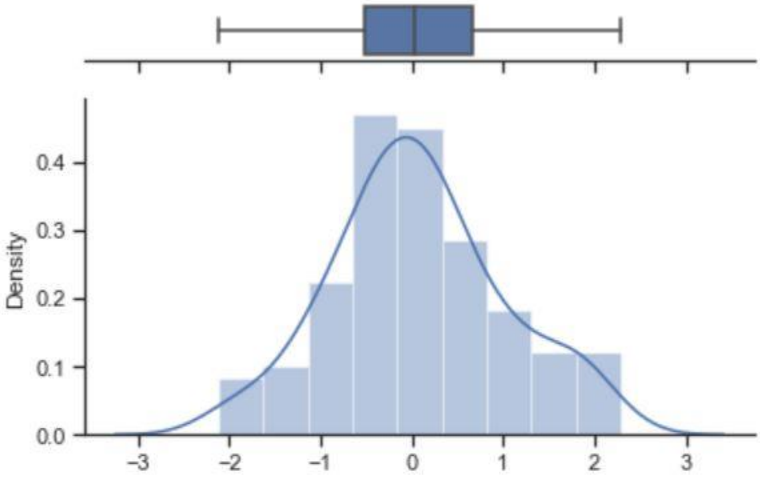
Output:



d) **Write a program to acquire data from data set and plot a pie chart using histogram.**

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="ticks")
x = np.random.randn(100)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,
                                    gridspec_kw={"height_ratios": (.15, .85)})
sns.boxplot(x, ax=ax_box)
sns.distplot(x, ax=ax_hist)
ax_box.set(yticks=[])
sns.despine(ax=ax_hist)
sns.despine(ax=ax_box, left=True)
```

Output:

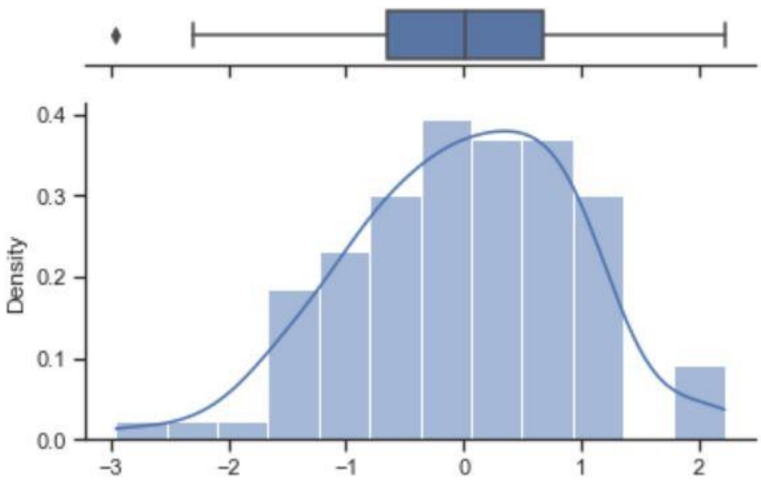


e) **Write a program to acquire data from data set and plot a pie chart using histogram.**

```
np.random.seed(2022)
x = np.random.randn(100)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (.15, .85)})
sns.boxplot(x=x, ax=ax_box)
sns.histplot(x=x, bins=12, kde=True, stat='density', ax=ax_hist)
ax_box.set(yticks=[])
```

```
sns.despine(ax=ax_hist)
sns.despine(ax=ax_box, left=True)
```

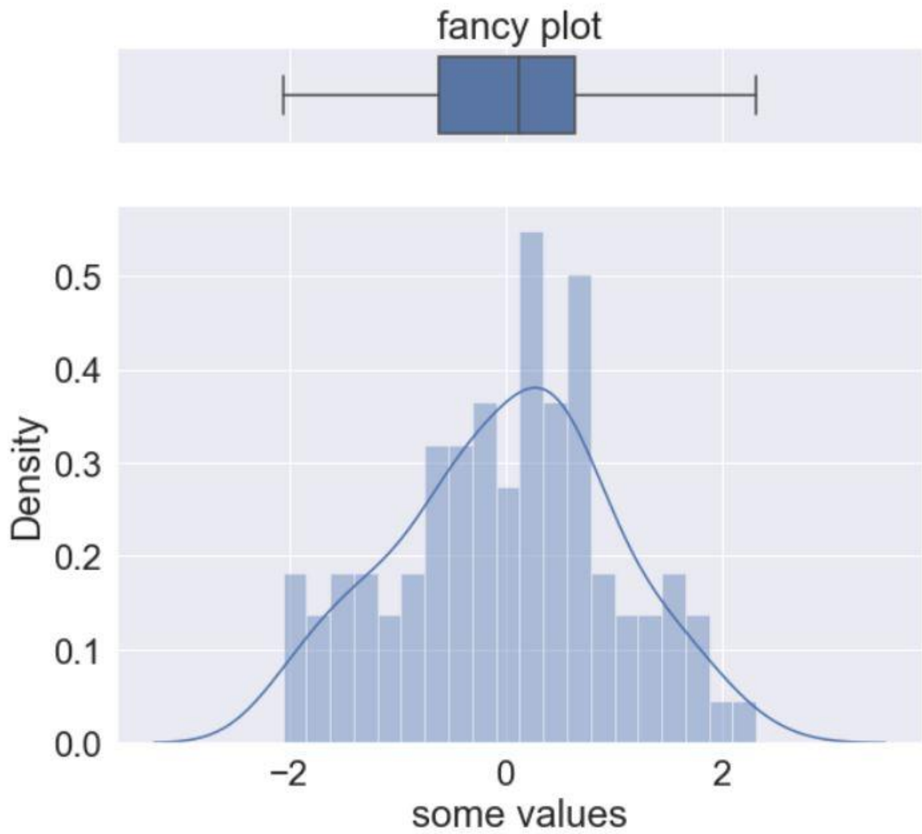
Output:



f) **Write a program to acquire data from data set and plot a pie chart using fancy histogram chart.**

```
import seaborn as sns
def histogram_boxplot(data, xlabel = None, title = None, font_scale=2, figsize=(9,8), bins = None):
    """ Boxplot and histogram combined
    data: 1-d data array
    xlabel: xlabel
    title: title
    font_scale: the scale of the font (default 2)
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    example use: histogram_boxplot(np.random.rand(100), bins = 20, title="Fancy plot")
    """
    sns.set(font_scale=font_scale)
    f2, (ax_box2, ax_hist2) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (.15,
.85)}, figsize=figsize)
    sns.boxplot(data, ax=ax_box2)
    sns.distplot(data, ax=ax_hist2, bins=bins) if bins else sns.distplot(data, ax=ax_hist2)
    if xlabel: ax_hist2.set(xlabel=xlabel)
    if title: ax_box2.set(title=title)
    plt.show()
histogram_boxplot(np.random.randn(100), bins=20, title="fancy plot", xlabel="some values")
```

Output:



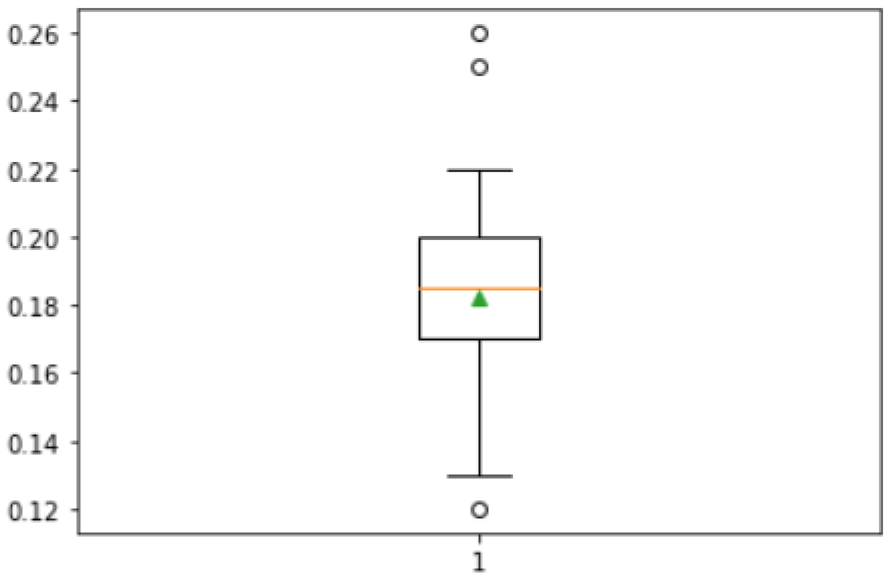
2) Statistical Analysis – such as Multivariate analysis, PCA, LDA, Correlation regression and analysis of variance

a) Compare lda number of components with naive bayes algorithm for classification

```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot

# get the dataset
def get_dataset():
```

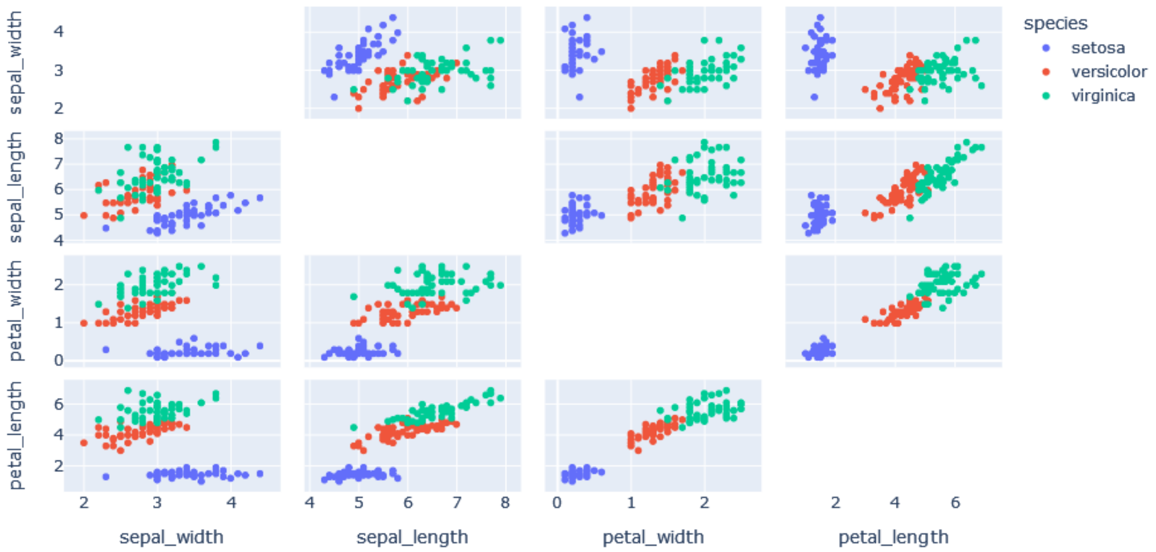
```
 X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=7, n_classes=10)
 return X, y

# get a list of models to evaluate
def get_models():
 models = dict()
 for i in range(1,10):
 steps = [('lda', LinearDiscriminantAnalysis(n_components=i)), ('m', GaussianNB())]
 models[str(i)] = Pipeline(steps=steps)
 return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
 cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
 scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
 return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
 scores = evaluate_model(model, X, y)
 results.append(scores)
 names.append(name)
 print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

```
>1 0.182 (0.032)
```



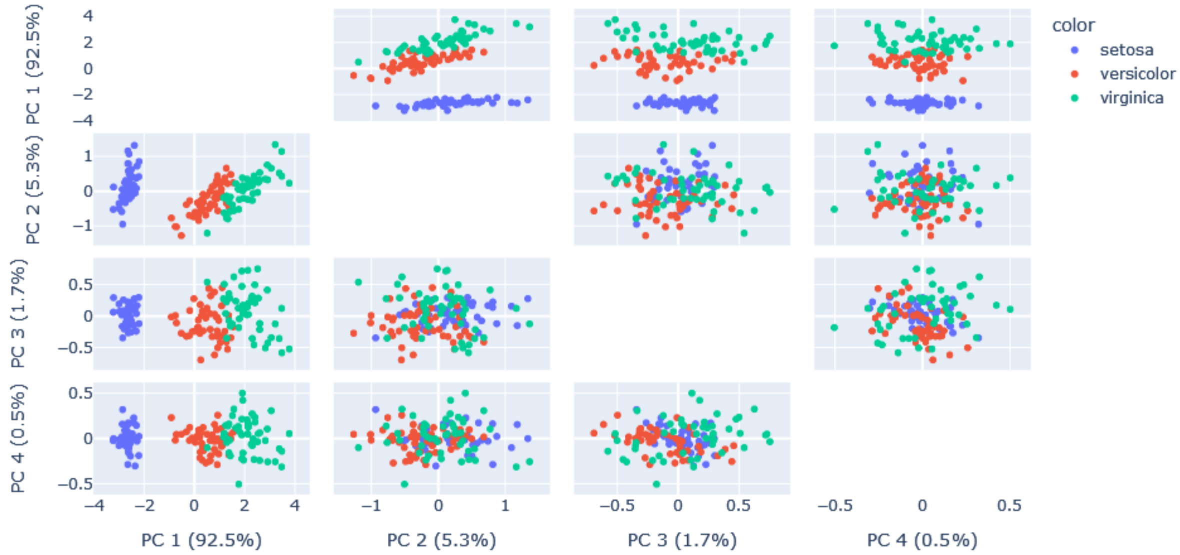
b) High-dimensional PCA Analysis with px.scatter_matrix

```
import plotly.express as px

df = px.data.iris()
features = ["sepal_width", "sepal_length", "petal_width", "petal_length"]

fig = px.scatter_matrix(
    df,
    dimensions=features,
    color="species"
)
fig.update_traces(diagonal_visible=False)
fig.show()
```




```
import plotly.express as px
from sklearn.decomposition import PCA

df = px.data.iris()
features = ["sepal_width", "sepal_length", "petal_width", "petal_length"]

pca = PCA()
components = pca.fit_transform(df[features])
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}

fig = px.scatter_matrix(
    components,
    labels=labels,
    dimensions=range(4),
    color=df["species"]
)
fig.update_traces(diagonal_visible=False)
```

```
fig.show()
```



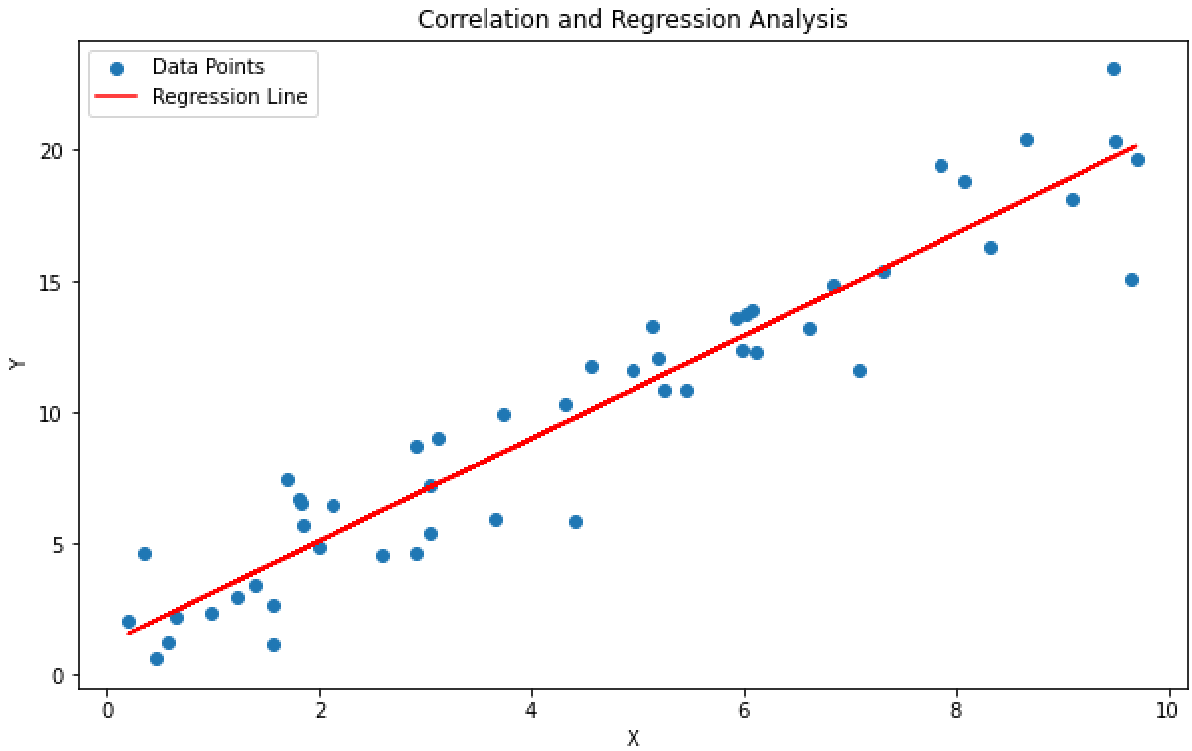
c) Statistical analysis of correlation regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generating synthetic data
np.random.seed(42)
x = np.random.rand(50) * 10
y = 2 * x + 1 + np.random.randn(50) * 2

# Creating a DataFrame
data = pd.DataFrame({'X': x, 'Y': y})

# Correlation analysis
correlation = data.corr()
print("Correlation Matrix:")
print(correlation)

# Linear regression
x_values = data['X'].values.reshape(-1, 1)
y_values = data['Y'].values.reshape(-1, 1)

regression_model = LinearRegression()
regression_model.fit(x_values, y_values)

# Getting regression parameters
slope = regression_model.coef_[0][0]
intercept = regression_model.intercept_[0]

print("\nLinear Regression:")
print(f"Slope (Coefficient): {slope}")
```

```
print(f"Intercept: {intercept}")

# Visualization
plt.figure(figsize=(10, 6))

# Scatter plot
plt.scatter(x, y, label='Data Points')

# Regression line
regression_line = slope * x + intercept
plt.plot(x, regression_line, color='red', label='Regression Line')

# Labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Correlation and Regression Analysis')
plt.legend()

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

Output:

```
Correlation Matrix:
          X         Y
X  1.000000  0.951177
Y  0.951177  1.000000

Linear Regression:
Slope (Coefficient): 1.9553132007706207
Intercept: 1.1933785489377744
```

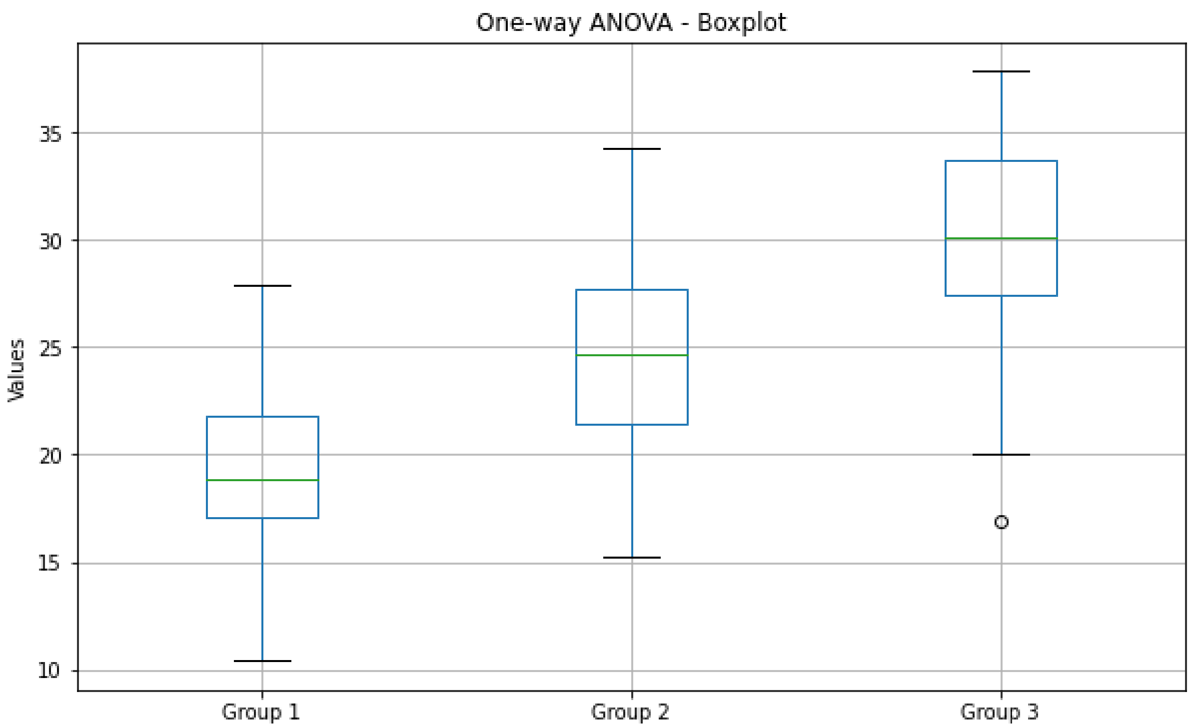


d) Statistical analysis of analysis of variance

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import f_oneway

# Generating synthetic data for three groups
np.random.seed(42)

group1 = np.random.normal(loc=20, scale=5, size=30)
group2 = np.random.normal(loc=25, scale=5, size=30)
group3 = np.random.normal(loc=30, scale=5, size=30)

# Creating a DataFrame
data = pd.DataFrame({
    'Group 1': group1,
    'Group 2': group2,
    'Group 3': group3
})

# One-way ANOVA
statistic, p_value = f_oneway(group1, group2, group3)

# Print ANOVA results
print("One-way ANOVA Results:")
print(f"F-statistic: {statistic}")
print(f"P-value: {p_value}")

# Visualization
plt.figure(figsize=(10, 6))

# Boxplot
data.boxplot()
plt.title('One-way ANOVA - Boxplot')
plt.ylabel('Values')

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

Output:

```
One-way ANOVA Results:
F-statistic: 40.97563597701803
P-value: 2.893768135071631e-13
```



3) Financial analysis using clustering, Histogram and Heatmap

a) Clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_blobs

# Generating synthetic financial data
np.random.seed(42)

# Creating three clusters of financial data
data, labels = make_blobs(n_samples=300, centers=3, random_state=42)

# Creating a DataFrame
financial_data = pd.DataFrame(data, columns=['Feature1', 'Feature2'])

# Standardizing the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(financial_data)

# Applying KMeans clustering
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
financial_data['Cluster'] = kmeans.fit_predict(scaled_data)

# Visualizing the clusters
plt.figure(figsize=(10, 6))
```

```
for cluster in range(num_clusters):
    cluster_data = financial_data[financial_data['Cluster'] == cluster]
    plt.scatter(cluster_data['Feature1'], cluster_data['Feature2'], label=f'Cluster {cluster + 1}')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X',
label='Centroids')
plt.title('Financial Data Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()

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

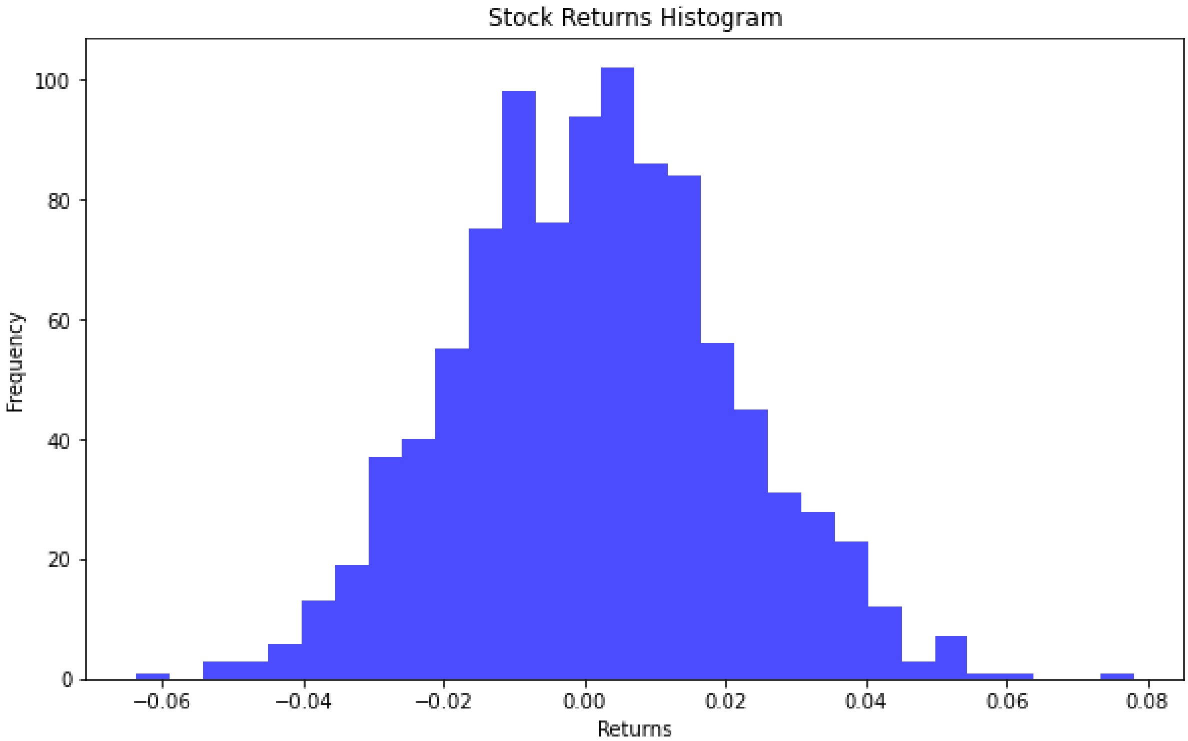
output:

b) Histogram

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

# Generating synthetic financial data (returns of a stock)
np.random.seed(42)
stock_returns = np.random.normal(loc=0.001, scale=0.02, size=1000)

# Creating a DataFrame
financial_data = pd.DataFrame({'Returns': stock_returns})

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.hist(financial_data['Returns'], bins=30, color='blue', alpha=0.7)
plt.title('Stock Returns Histogram')
```

```
plt.xlabel('Returns')
plt.ylabel('Frequency')

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

output:

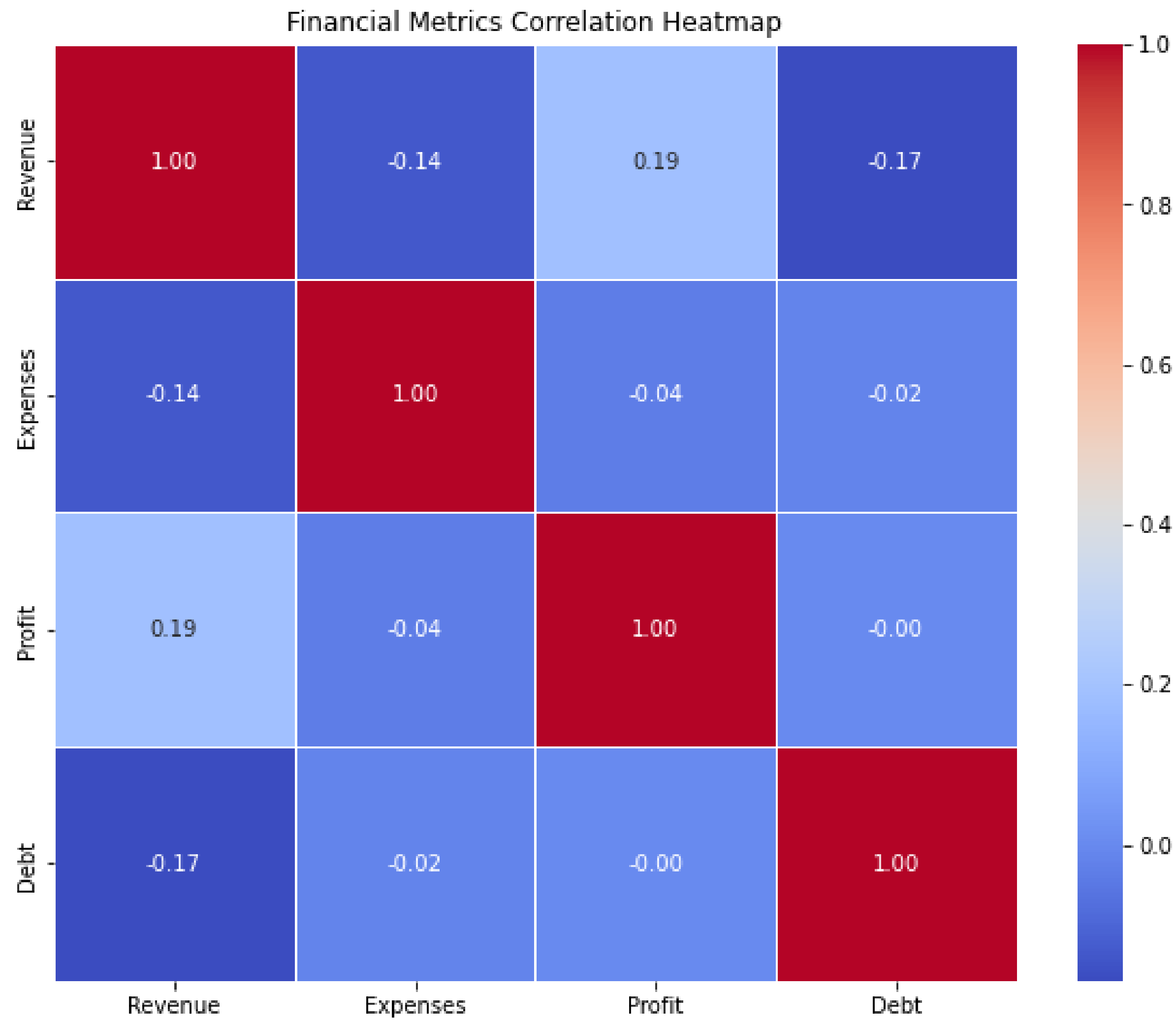


c) Heatrmap

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Generating synthetic financial data (correlated metrics)
np.random.seed(42)

# Creating a DataFrame with synthetic financial metrics
financial_metrics = {
    'Revenue': np.random.normal(loc=100, scale=20, size=100),
    'Expenses': np.random.normal(loc=70, scale=15, size=100),
    'Profit': np.random.normal(loc=30, scale=10, size=100),
    'Debt': np.random.normal(loc=50, scale=15, size=100)
}

financial_data = pd.DataFrame(financial_metrics)

# Calculating correlation matrix
correlation_matrix = financial_data.corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Financial Metrics Correlation Heatmap')

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

output:



4) Time Series Analysis-Stock market

Step 1:

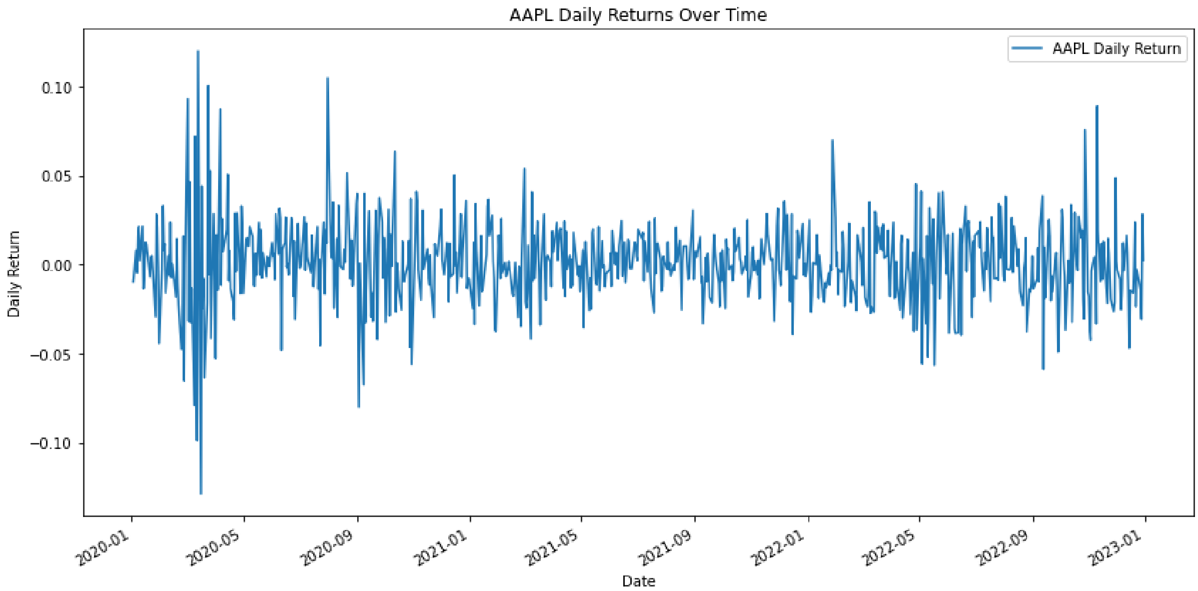
```
!pip install yfinance
```

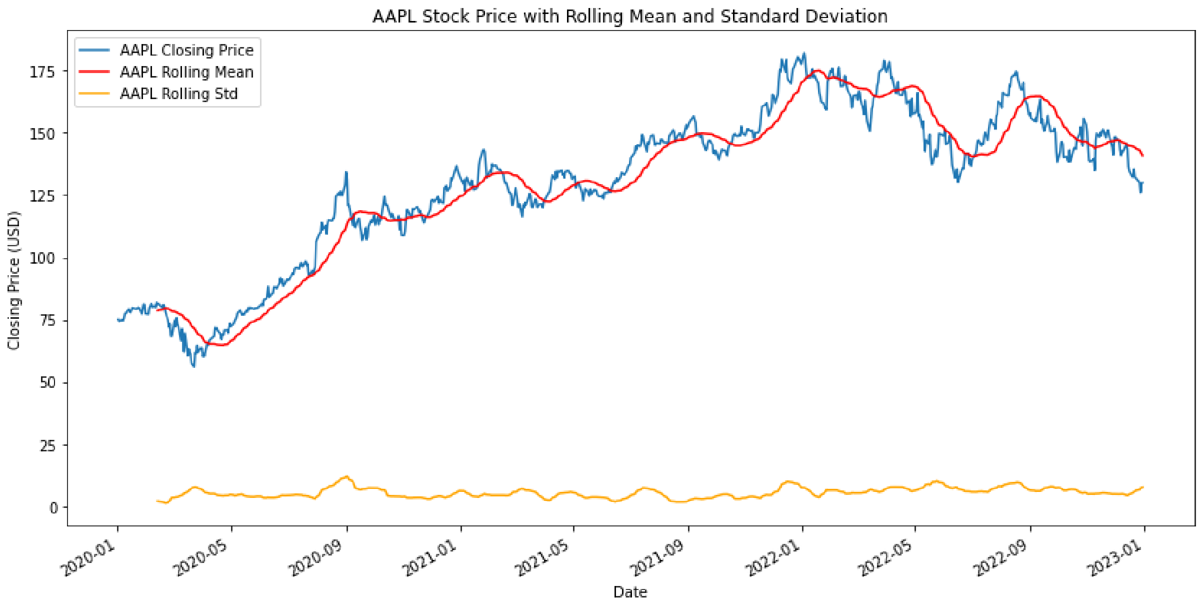
Step 2:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf

# Fetching historical stock data (e.g., Apple Inc.)
ticker_symbol = "AAPL"
start_date = "2020-01-01"
end_date = "2022-12-31"

stock_data = yf.download(ticker_symbol, start=start_date, end=end_date)
```

```
# Displaying the first few rows of the stock data
print(stock_data.head())

# Plotting the closing prices over time
plt.figure(figsize=(14, 7))
stock_data['Close'].plot(label=f'{ticker_symbol} Closing Price')
plt.title(f'{ticker_symbol} Stock Price Over Time')
plt.xlabel('Date')
plt.ylabel('Closing Price (USD)')
plt.legend()
plt.show()

# Calculating and plotting daily returns
stock_data['Daily Return'] = stock_data['Close'].pct_change()
plt.figure(figsize=(14, 7))
stock_data['Daily Return'].plot(label=f'{ticker_symbol} Daily Return')
plt.title(f'{ticker_symbol} Daily Returns Over Time')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.legend()
plt.show()

# Rolling mean and standard deviation for smoothing
window_size = 30
stock_data['Rolling Mean'] = stock_data['Close'].rolling(window=window_size).mean()
stock_data['Rolling Std'] = stock_data['Close'].rolling(window=window_size).std()

# Plotting with rolling mean and standard deviation
plt.figure(figsize=(14, 7))
stock_data['Close'].plot(label=f'{ticker_symbol} Closing Price')
stock_data['Rolling Mean'].plot(label=f'{ticker_symbol} Rolling Mean', color='red')
stock_data['Rolling Std'].plot(label=f'{ticker_symbol} Rolling Std', color='orange')
plt.title(f'{ticker_symbol} Stock Price with Rolling Mean and Standard Deviation')
plt.xlabel('Date')
plt.ylabel('Closing Price (USD)')
plt.legend()
plt.show()
```

Output:





5. Visualization of various massive dataset - Finance - Healthcare-Census-Geospatial.

a) Finance:

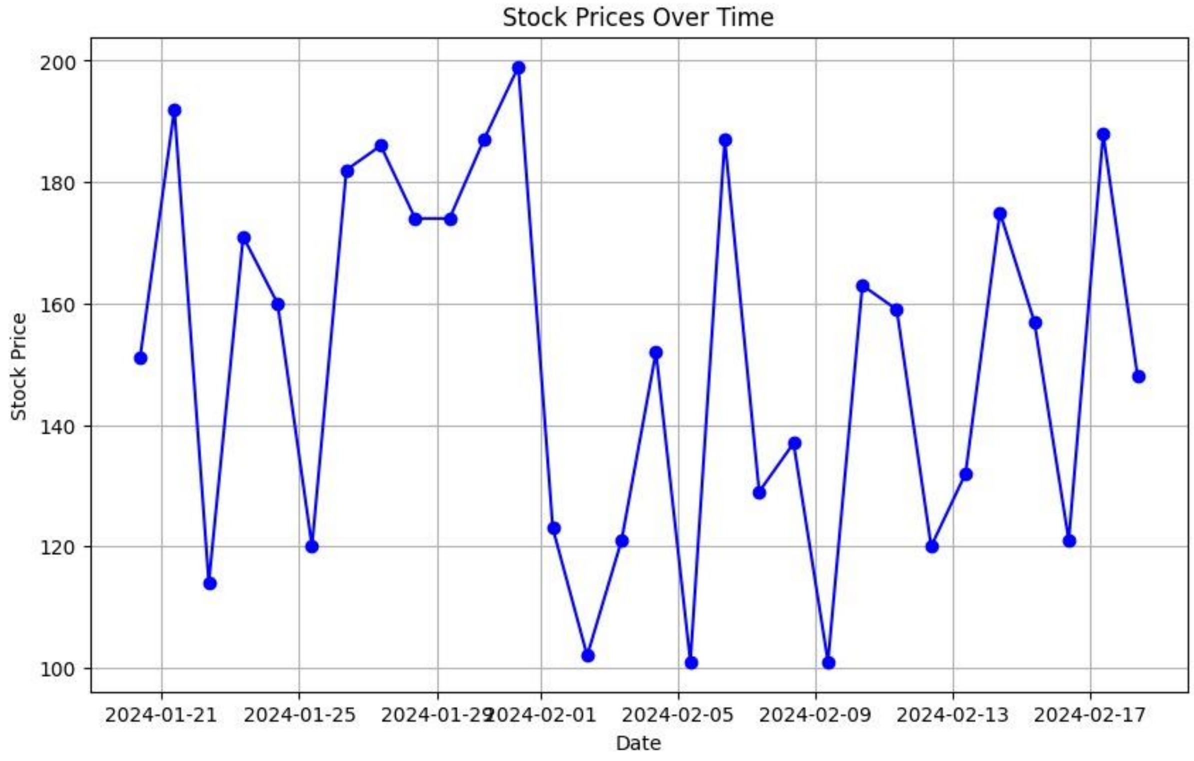
Step 1:-

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Generate synthetic data
np.random.seed(42)
date_today = datetime.now()
days = pd.date_range(date_today, date_today + timedelta(29), freq='D')
data = {'Date': days, 'StockPrice': np.random.randint(100, 200, size=(30,))}
df = pd.DataFrame(data)

# Save the dataset to a CSV file
df.to_csv('finance_dataset.csv',index=False)
```

Step 2:

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

# Read the dataset from the CSV file
df = pd.read_csv('finance_dataset.csv')

# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Plotting the financial data
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['StockPrice'], marker='o', linestyle='-', color='b')

# Adding labels and title
plt.title('Stock Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Stock Price')

# Display the plot
plt.grid(True)
plt.show()
```

Output:-



b) Healthcare:
Dataset: https://drive.google.com/file/d/1YQKfblYW9L4p-JOn2XRibr7VMa3STh86/view?usp=sharing

```
import pandas as pd
```

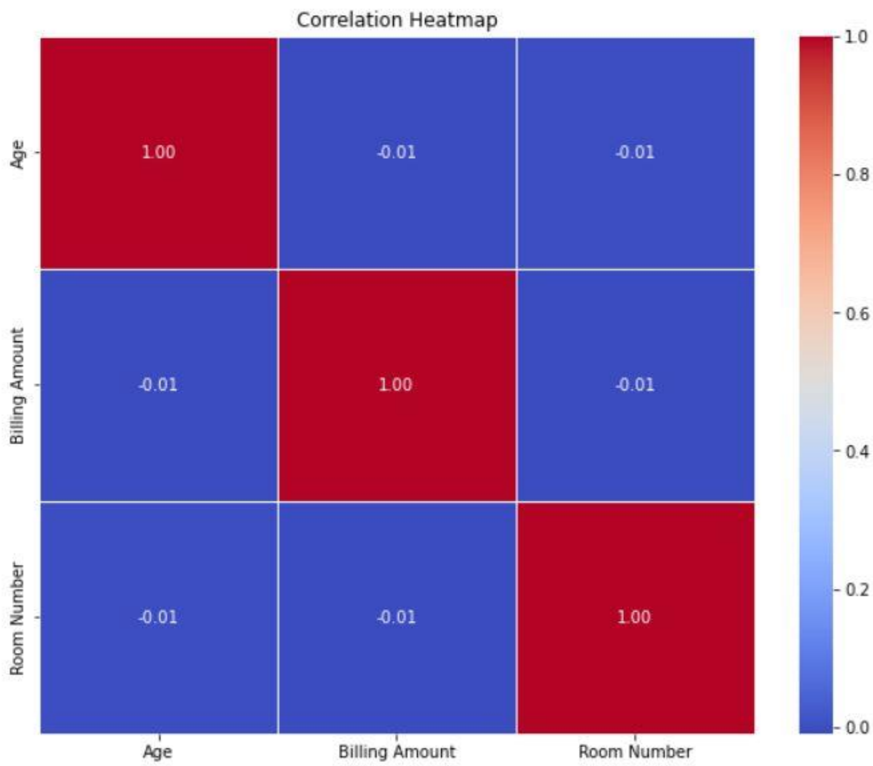
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load your healthcare dataset (replace 'path/to/your/healthcare_dataset.csv' with the actual path)
healthcare_data = pd.read_csv('healthcare_dataset.csv')

# Display basic information about the dataset
print("Dataset Overview:")
print(healthcare_data.info())

# Display the first few rows of the dataset
print("\nFirst Few Rows of the Dataset:")
print(healthcare_data.head())

# Descriptive statistics of numerical columns
print("\nDescriptive Statistics:")
print(healthcare_data.describe())

# Handling missing data
print("\nHandling Missing Data:")
print(healthcare_data.isnull().sum())

# Data visualization

# Example: Bar plot for patient gender distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Gender', data=healthcare_data)
plt.title('Patient Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()

# Example: Box plot for age distribution by gender
plt.figure(figsize=(10, 6))
sns.boxplot(x='Gender', y='Age', data=healthcare_data)
plt.title('Age Distribution by Gender')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.show()

# Example: Heatmap for correlation between numerical variables
plt.figure(figsize=(10, 8))
sns.heatmap(healthcare_data.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

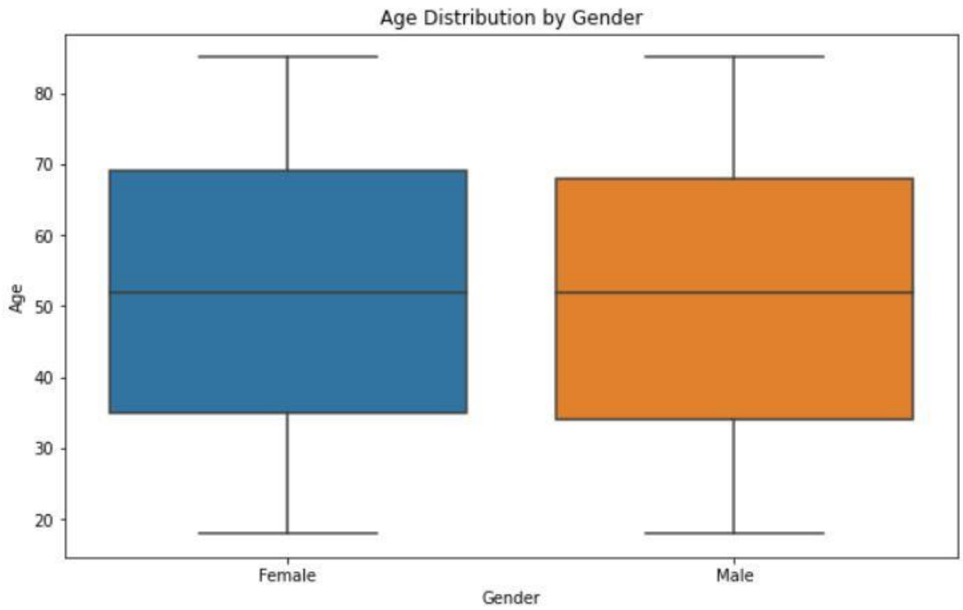
Output:

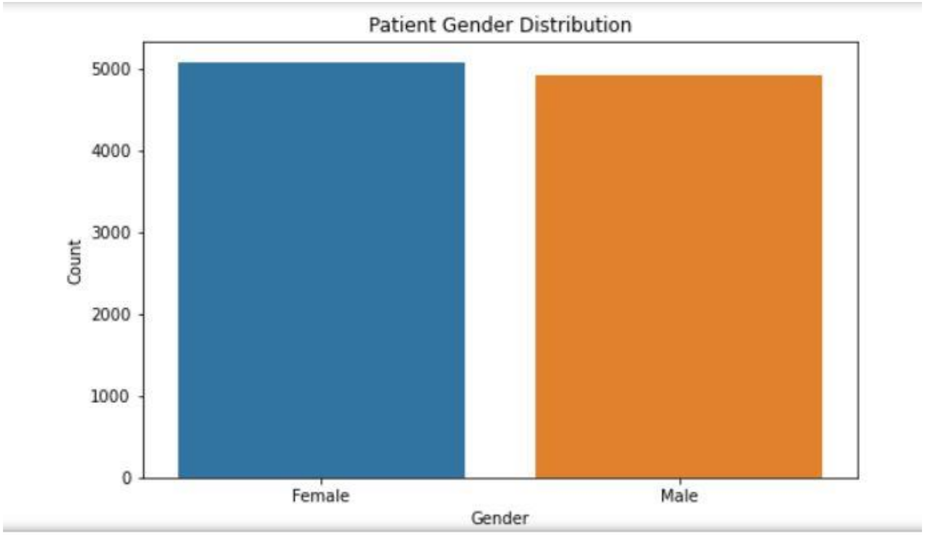
```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Name                10000 non-null  object
 1   Age                 10000 non-null  int64
```

```
 2   Gender              10000 non-null  object 
 3   Blood Type          10000 non-null  object 
 4   Medical Condition   10000 non-null  object 
 5   Date of Admission   10000 non-null  object 
 6   Doctor              10000 non-null  object 
 7   Hospital            10000 non-null  object 
 8   Insurance Provider  10000 non-null  object 
 9   Billing Amount      10000 non-null  float64
 10  Room Number         10000 non-null  int64  
 11  Admission Type      10000 non-null  object 
 12  Discharge Date      10000 non-null  object 
 13  Medication          10000 non-null  object 
 14  Test Results        10000 non-null  object 
dtypes: float64(1), int64(2), object(12)
memory usage: 1.1+ MB
None

First Few Rows of the Dataset:
                  Name  Age  Gender Blood Type Medical Condition  \
0      Tiffany Ramirez   81  Female         O-          Diabetes   
1          Ruben Burns   35    Male         O+            Asthma   
2            Chad Byrd   61    Male         B-           Obesity   
3     Antonio Frederick   49    Male         B-            Asthma   
4  Mrs. Brandy Flowers   51    Male         O-         Arthritis   

  Date of Admission          Doctor                   Hospital  \
0        2022-11-17  Patrick Parker           Wallace-Hamilton   
1        2023-06-01   Diane Jackson  Burke, Griffin and Cooper   
2        2019-01-09      Paul Baker                 Walton LLC   
3        2020-05-02  Brian Chandler                 Garcia Ltd   
4        2021-07-09  Dustin Griffin    Jones, Brown and Murray   

  Insurance Provider  Billing Amount  Room Number Admission Type  \
0           Medicare    37490.983364          146       Elective   
1   UnitedHealthcare    47304.064845          404      Emergency   
2           Medicare    36874.896997          292      Emergency   
3           Medicare    23303.322092          480         Urgent   
4   UnitedHealthcare    18086.344184          477         Urgent   

  Discharge Date   Medication  Test Results  
0     2022-12-01      Aspirin  Inconclusive  
1     2023-06-15      Lipitor        Normal  
2     2019-02-08      Lipitor        Normal  
3     2020-05-03   Penicillin      Abnormal  
4     2021-08-02  Paracetamol        Normal  

Descriptive Statistics:
                Age  Billing Amount   Room Number
count  10000.000000    10000.000000  10000.000000
mean      51.452200    25516.806778    300.082000
std       19.588974    14067.292709    115.806027
min       18.000000     1000.180837    101.000000
25%       35.000000    13506.523967    199.000000
50%       52.000000    25258.112566    299.000000
75%       68.000000    37733.913727    400.000000
max       85.000000    49995.902283    500.000000

Handling Missing Data:
Name                  0
Age                   0
```

```
Gender                0
Blood Type            0
Medical Condition     0
Date of Admission     0
Doctor                0
Hospital              0
Insurance Provider    0
Billing Amount        0
Room Number           0
Admission Type        0
Discharge Date        0
Medication            0
Test Results          0
dtype: int64
```


Correlation Heatmap
Age
Billing Amount
Room Number
1.00
-0.01
-0.01
-0.01
1.00
-0.01
-0.01
-0.01
1.00
Age
Billing Amount
Room Number
1.0
0.8
0.6
0.4
0.2
0.0

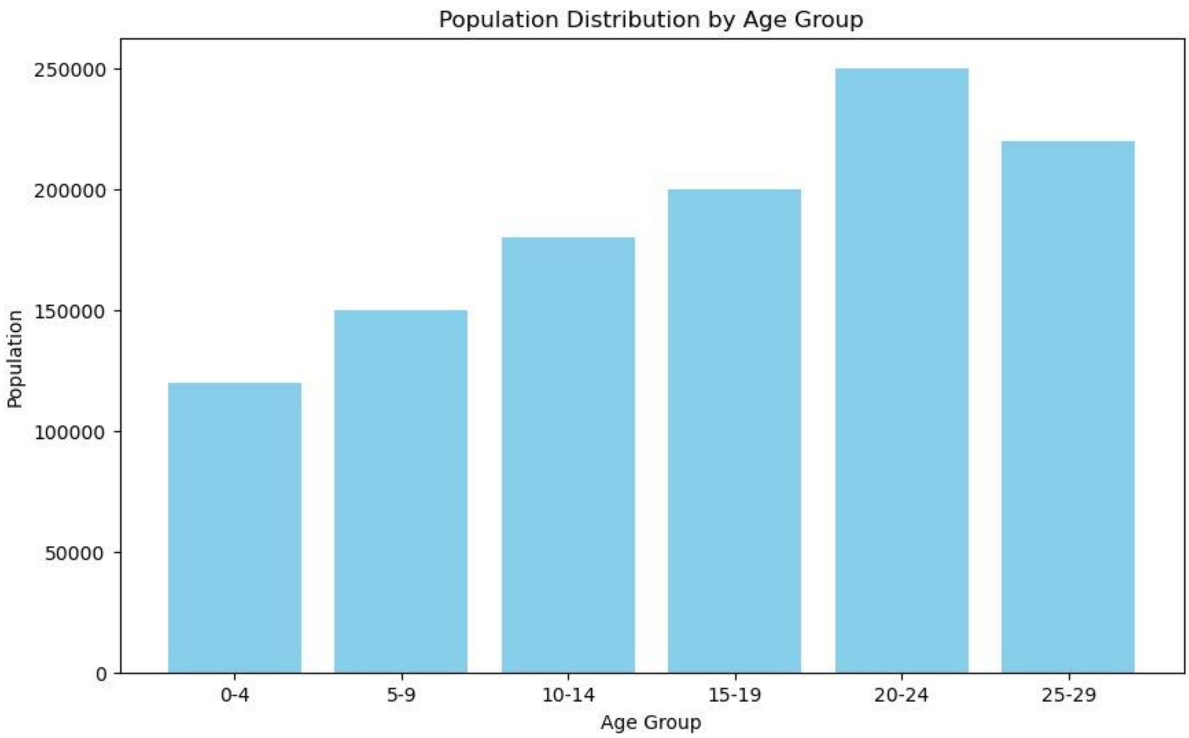

Age Distribution by Gender
80
70
60
50
40
30
20
Age
Female
Male
Gender



c) Census

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

# Assuming you have a pandas DataFrame with columns 'Age' and 'Population'
# Replace this with your actual data loading mechanism
# Example data creation:
data = {
    'Age': ['0-4', '5-9', '10-14', '15-19', '20-24', '25-29'],
    'Population': [120000, 150000, 180000, 200000, 250000, 220000]
}

df = pd.DataFrame(data)

# Plotting the data
plt.figure(figsize=(10, 6))
plt.bar(df['Age'], df['Population'], color='skyblue')
plt.xlabel('Age Group')
plt.ylabel('Population')
plt.title('Population Distribution by Age Group')
plt.show()
```

Output:



d) Geopatial

Step 1:-

```
!pip install geopandas folium
```

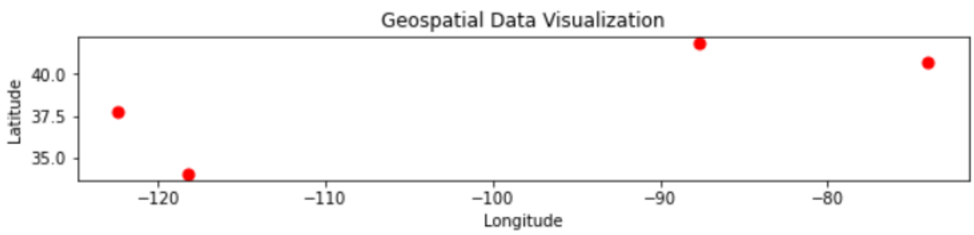
Step 2:-

```
import geopandas as gpd
import folium

# Create a GeoDataFrame with example geospatial data
data = {
    'City': ['City A', 'City B', 'City C', 'City D'],
    'Country': ['Country 1', 'Country 2', 'Country 1', 'Country 3'],
    'Latitude': [34.0522, 40.7128, 41.8781, 37.7749],
    'Longitude': [-118.2437, -74.0060, -87.6298, -122.4194],
}

geometry = gpd.points_from_xy(data['Longitude'], data['Latitude'])
geo_df = gpd.GeoDataFrame(data, geometry=geometry)

# Display the GeoDataFrame
print("GeoDataFrame Information:")
print(geo_df)

# Plot the GeoDataFrame
geo_df.plot(marker='o', color='red', markersize=50, figsize=(10, 6))
plt.title('Geospatial Data Visualization')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()

# Create an interactive map using folium
my_map = folium.Map(location=[geo_df['Latitude'].mean(), geo_df['Longitude'].mean()],
zoom_start=4)
```

```
# Add markers to the map
for index, row in geo_df.iterrows():
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=f"{row['City']}, {row['Country']}",
        icon=folium.Icon(color='blue')
    ).add_to(my_map)

# Save the map as an HTML file (optional)
my_map.save('geospatial_map.html')

# Display the map
my_map
```

Output:-

```
GeoDataFrame Information:
     City    Country  Latitude  Longitude                    geometry
0  City A  Country 1   34.0522  -118.2437  POINT (-118.24370 34.05220)
1  City B  Country 2   40.7128   -74.0060   POINT (-74.00600 40.71280)
2  City C  Country 1   41.8781   -87.6298   POINT (-87.62980 41.87810)
3  City D  Country 3   37.7749  -122.4194  POINT (-122.41940 37.77490)
```



6. Visualization on Streaming dataset (Stock market dataset, weather forecasting).

a) Stockmarket

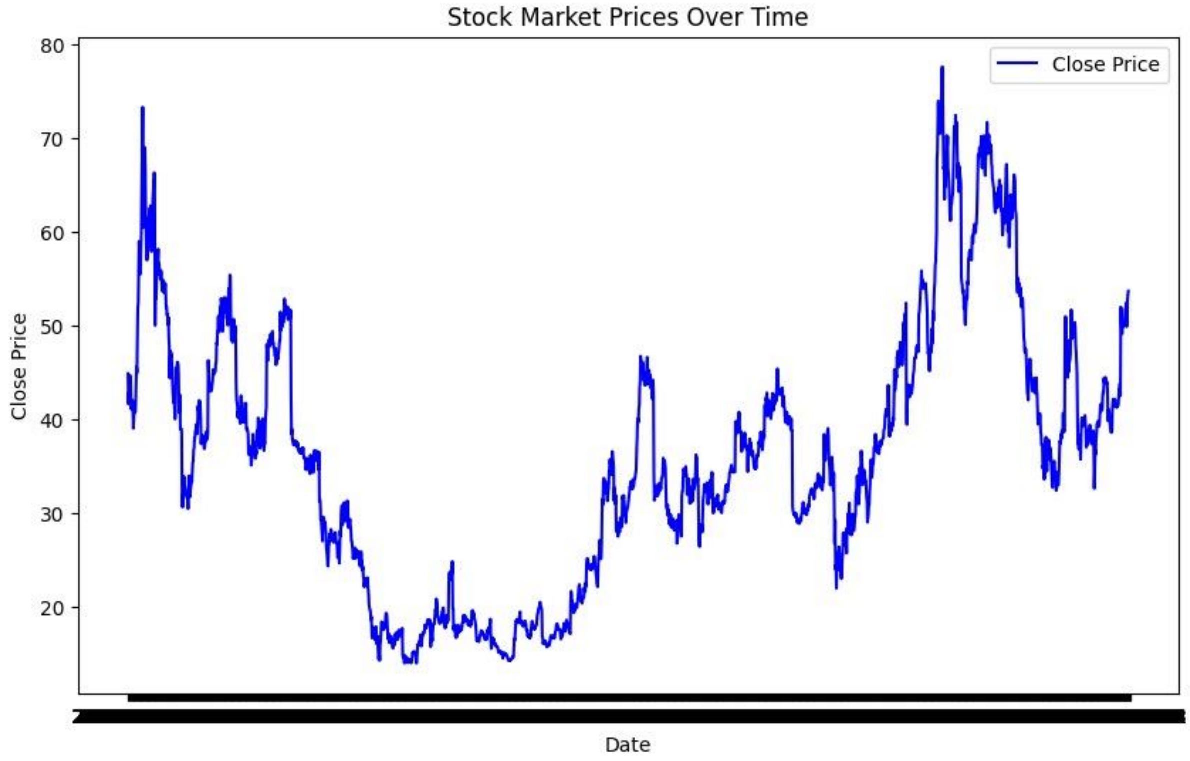
Dataset: https://drive.google.com/file/d/1ODwDMeX97fbfR-gi3VbEC4WgLjUBjAyA/view?usp=sharing

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

# Assuming you have a stock market dataset in a CSV file, load it into a DataFrame
# Replace 'your_dataset.csv' with the actual file path
df = pd.read_csv('Twitter Stock Market Dataset.csv')

# Display the first few rows of the dataset
```

```
print(df.head())

# Plotting the stock prices
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Close'], label='Close Price', color='blue')
plt.title('Stock Market Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

Output:

```
Date       Open       High        Low      Close  Adj Close  \
0  2013-11-07  45.099998  50.090000  44.000000  44.900002  44.900002
1  2013-11-08  45.930000  46.939999  40.685001  41.650002  41.650002
2  2013-11-11  40.500000  43.000000  39.400002  42.900002  42.900002
3  2013-11-12  43.660000  43.779999  41.830002  41.900002  41.900002
4  2013-11-13  41.029999  42.869999  40.759998  42.599998  42.599998

        Volume
0  117701670.0
1   27925307.0
2   16113941.0
3    6316755.0
4    8688325.0
```



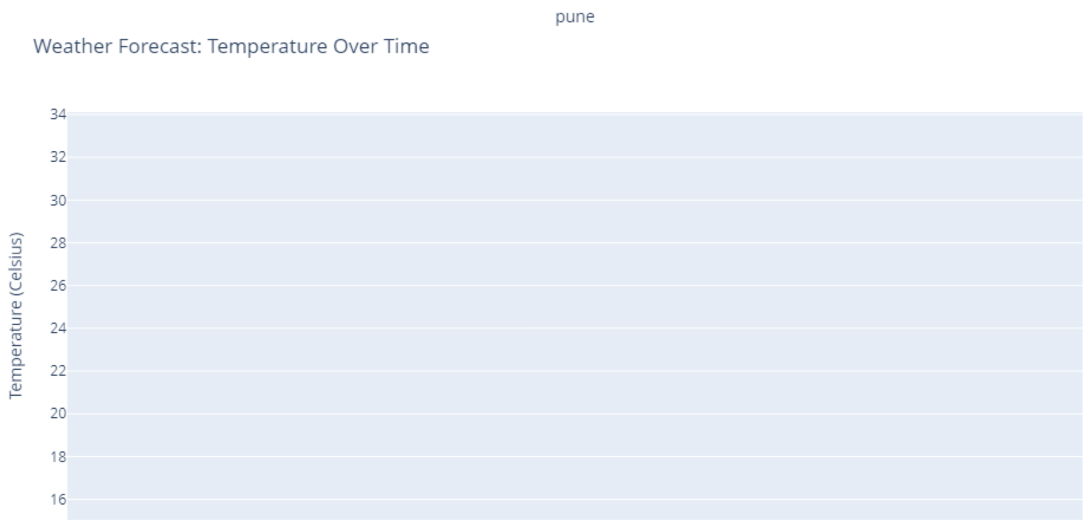
b) Weather forecasting

Dataset:
https://drive.google.com/file/d/1tdmXYJudSINdVLMkQ9bNvhtRGibwneiz/view?usp=sharing

```
!pip install plotly
import pandas as pd
```

```
import plotly.express as px

# Assuming you have a weather forecasting dataset in a CSV file, load it into a DataFrame
# Replace 'your_weather_forecast_data.csv' with the actual file path
df = pd.read_csv('Weather.csv')

# Check and clean column names
df.columns = df.columns.str.strip()

# Plotting interactive temperature over time
fig = px.line(df, x='pune', y='9', title='Weather Forecast: Temperature Over Time', labels={'9':
'Temperature (Celsius)'})
fig.show()
```

output:

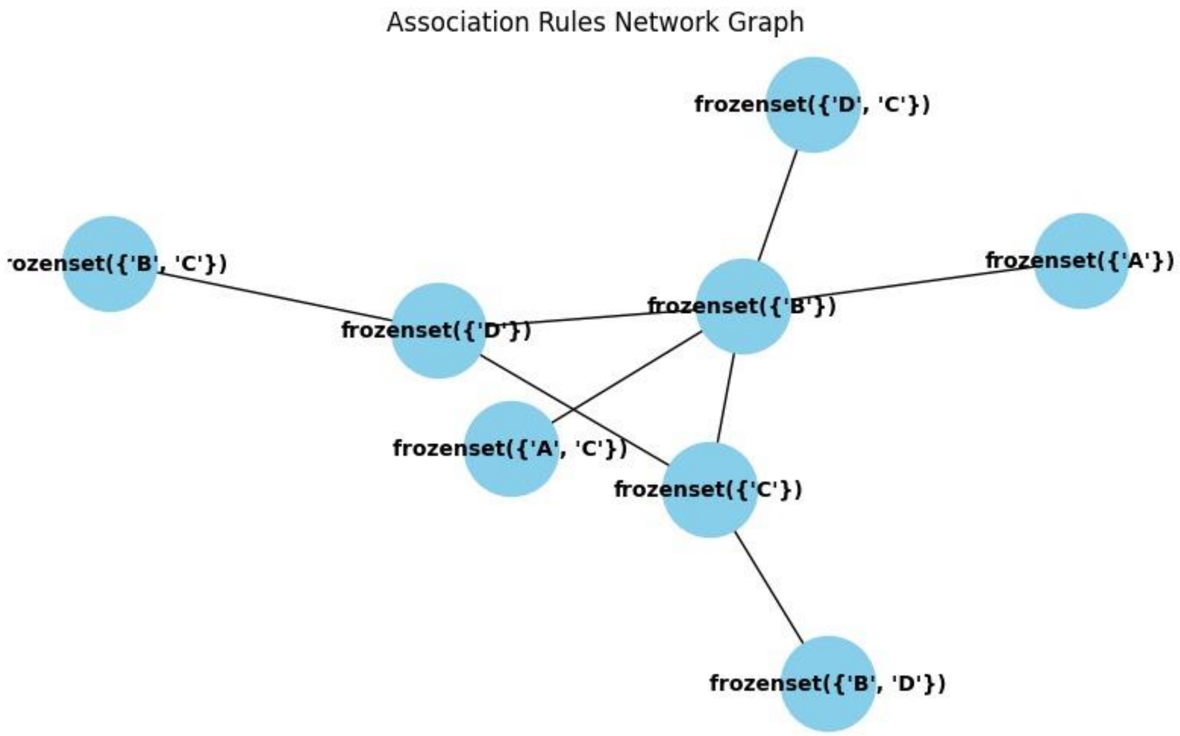


7. Market-Basket Data analysis-visualization.

```
!pip install mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Sample transaction data (replace with your actual data)
data = {'TransactionID': [1, 1, 2, 2, 2, 3, 3, 4, 4, 4],
        'Item': ['A', 'B', 'A', 'B', 'C', 'A', 'B', 'B', 'C', 'D']}
df = pd.DataFrame(data)

# Perform one-hot encoding
basket = (df.groupby(['TransactionID', 'Item'])['Item']
          .count().unstack().reset_index().fillna(0)
          .set_index('TransactionID'))

# Convert counts to binary values (1 or 0)
basket_sets = basket.applymap(lambda x: 1 if x > 0 else 0)
```

```
# Apply Apriori algorithm
frequent_itemsets = apriori(basket_sets, min_support=0.2, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Visualization: Plotting Network Graph of Association Rules
fig, ax = plt.subplots(figsize=(10, 6))
G = nx.from_pandas_edgelist(rules, 'antecedents', 'consequents')
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_size=10, node_size=2000, node_color="skyblue",
font_color="black", font_weight="bold", ax=ax)
plt.title("Association Rules Network Graph")
plt.show()
```

Output:



8. Text visualization using web analytics.

DataSet:
https://drive.google.com/file/d/14P6d7faPAQqVg4ZKGH_RncF367Q8xUgh/view?usp=sharing
Program:

```
!pip install pandas wordcloud matplotlib
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a web analytics dataset
# Replace 'Web_Analytics_Dataset.csv' with the actual file path
df = pd.read_csv('Web Analytic_Dataset.csv')

# Print column names to identify the available columns
print("Available columns:", df.columns)
```

```
# Choose relevant metrics for visualization
metrics_to_visualize = ['Users', 'Pageviews', 'Bounce Rate']

# Plotting web analytics metrics
plt.figure(figsize=(12, 6))
for metric in metrics_to_visualize:
    plt.bar(df['Month of the year'], df[metric], label=metric, alpha=0.7)

plt.title('Web Analytics Metrics Over Time')
plt.xlabel('Month of the year')
plt.ylabel('Metrics Value')
plt.legend()
plt.show()
```

Output:

```
Available columns: Index(['Source / Medium', 'Year', 'Month of the year', 'Users', 'New Users',
       'Sessions', 'Bounce Rate', 'Pageviews', 'Avg. Session Duration',
       'Conversion Rate (%)', 'Transactions', 'Revenue', 'Quantity Sold'],
      dtype='object')
```