EX NO 2. DATA ANALYSIS ON EMAIL DATA

Program:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the dataset
df = pd.read_csv('D:\ARCHANA\dxv\LAB\DXV\Emaildataset.csv')
# Display basic information about the dataset
print(df.info())
# Display the first few rows of the dataset
print(df.head())
# Descriptive statistics
print(df.describe())
# Check for missing values
print(df.isnull().sum())
# Visualize the distribution of numerical variables
sns.pairplot(df)
plt.show()
# Visualize the distribution of categorical variables
sns.countplot(x='label', data=df)
plt.show()
# Correlation matrix for numerical variables
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
# Word cloud for text data (if you have a column with text data)
from wordcloud import WordCloud
text_data = ' '.join(df['text_column'])
wordcloud = WordCloud(width=800, height=400, random_state=21,
max_font_size=110).generate(text_data)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

EX NO.6. PERFORM DATA ANALYSIS AND REPRESENTATION ON A MAP

```
import pandas as pd
import folium

# Sample data (City, Latitude, Longitude, Population)
data = {
  'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix'],
  'Latitude': [40.7128, 34.0522, 41.8781, 29.7604, 33.4484],
  'Longitude': [-74.0060, -118.2437, -87.6298, -95.3698, -112.0740],
  'Population': [8419600, 3980400, 2716000, 2328000, 1690000]
}

# Create a pandas DataFrame
df = pd.DataFrame(data)

# Create a base map centered around the US
map = folium.Map(location=[37.0902, -95.7129], zoom_start=4)

# Add city markers to the map
for i, row in df.iterrows():
  folium.CircleMarker(
    location=[row['Latitude'], row['Longitude']],
    radius=row['Population'] / 1000000,  # Marker size based on population
    popup=f"{row['City']} (Population: {row['Population']})",
    color='blue',
    fill=True,
    fill_color='blue'
  ).add_to(map)

# Save the map to an HTML file
map.save("city_population_map.html")

# Display map in Jupyter notebook or inline (optional for notebooks)
map
```

EX.NO : 7

CARTOGRAPHIC VISUALIZATION

AIM:

To build cartographic visualization for multiple datasets involving various countries of the world; states and districts in India etc.

DEFINITION:

Creating cartographic visualizations for multiple datasets involving various countries, states, or districts often involves combining data with geographica boundaries. Here's a Python script that utilizes `geopandas` and `folium` to create visualizations for both world countries and states in India, along with fictional data for illustration

PROGRAM

```
import folium

# Create a folium map centered around a specific location
m = folium.Map(location=[20.5937, 78.9629], zoom_start=5)

# Add a marker for a few world countries
folium.Marker([37.7749, -122.4194],
popup='USA').add_to(m)
folium.Marker([35.8617, 104.1954], popup='China').add_to(m)
folium.Marker([20.5937, 78.9629], popup='India').add_to(m)

# Add a marker for a few Indian states
folium.Marker([19.7515, 75.7139], popup='Maharashtra').add_to(m)
folium.Marker([27.0238, 74.2179], popup='Rajasthan').add_to(m)

# Save the map
m.save("world_and_india_visualization_simple.html")
```

AIM:

To Perform EDA on Wine Quality Data Set.

DEFINITION:
Exploratory Data Analysis (EDA) is a crucial step in understanding the characteristics of a dataset. Let's perform EDA on a wine quality dataset. For this example, I'll use the Wine Quality dataset available in the UCI Machine Learning Repository.

PROGRAM:

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

# Load the Wine Quality dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality- white.csv"
wine_data = pd.read_csv(url, sep=';')

# Display the first few rows of the
dataset print(wine_data.head())

# Summary statistics
print(wine_data.describe())

# Distribution of Wine Quality
sns.countplot(x='quality',
data=wine_data) plt.title('Distribution
of Wine Quality') plt.show()

# Correlation heatmap
correlation_matrix =
wine_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.!
plt.title('Correlation Heatmap')
plt.show()
```



```
# Pairplot for selected features
selected_features = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'quality']
sns.pairplot(wine_data[selected_features], hue='quality',

markers='o')

plt.title('Pairplot of Selected Features')

plt.show()
```

```
df <- iris

column_names <- c("sepal_length",
"sepal_width", "petal_length", "petal_width",
"class")

# Display the first few rows of the dataset
cat("Sample Dataset:\n")
print(head(df))
# Load the built-in Iris dataset
df <- iris

# Display basic information about the dataset
cat("Dataset Information:\n")
str(df)

# Display summary statistics
cat("\nSummary Statistics:\n")
summary(df)

# Display the first few rows of the dataset
cat("\nFirst Few Rows of the Dataset:\n")
head(df)

# Display unique classes in the 'Species'
column
cat("\nUnique Classes:\n")
unique(df$Species)
# Create a sample DataFrame
data <- data.frame(
  Name = c('Alice', 'Bob', 'Charlie'),
  Age = c(25, 30, 22),
  City = c('New York', 'San Francisco', 'Los
Angeles')
)

# Display the original DataFrame
cat("Original DataFrame:\n")
print(data)

# Filter specific variables (columns)
selected_columns <- c('Name', 'City')
filtered_df <- data[selected_columns]

# Display the filtered DataFrame
cat("\nFiltered DataFrame:\n")
print(filtered_df)

# Sample data
data <- data.frame(
  Name = c('Alice', 'Bob', 'Charlie', 'David'),
  Age = c(20, 22, 21, 23),
  Grade = c(85, 92, 78, 95)
)

# Display the original DataFrame
cat("Original DataFrame:\n")
print(data)
# Sample data with missing values and
duplicates
data <- data.frame(
  Name = c('Alice', 'Bob', 'Charlie', 'David',
'Alice'),
  Age = c(20, NA, 21, 23, 22),
  Grade = c(85, 92, 78, 95, 92)
)

# Display the original DataFrame
cat("Original DataFrame:\n")
print(data)
```

EX NO 5. PERFORM TIME SERIES ANALYSIS AND APPLY THE VARIOUS VISUALISATION TECHNIQUES

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from pandas.plotting import autocorrelation_plot

# Generate a time series data for demonstration (or you can load your own dataset)
def generate_time_series_data():
   np.random.seed(0)
   date_range = pd.date_range(start='2020-01-01', periods=1000, freq='D')
   data = np.cumsum(np.random.randn(1000)) + 10 * np.sin(np.linspace(0, 50, 1000))
   return pd.DataFrame(data, index=date_range, columns=['Value'])

# Load or generate your time series data
data = generate_time_series_data()
# 1. Basic Line Plot of Time Series
def plot_time_series(data):
   plt.figure(figsize=(10, 6))
   plt.plot(data.index, data['Value'], label='Value', color='blue')
   plt.title('Time Series Plot')
   plt.xlabel('Date')
   plt.ylabel('Value')
   plt.legend()
   plt.show()

# 2. Rolling Mean and Rolling Standard Deviation Plot
def plot_rolling_statistics(data, window=30):
  rolling_mean = data['Value'].rolling(win dow=window).mean()
   rolling_std = data['Value'].rolling(window=window).std()
   plt.figure(figsize=(10, 6))
   plt.plot(data['Value'], label='Original', color='blue')
   plt.plot(rolling_mean, label='Rolling Mean', color='red')
   plt.plot(rolling_std, label='Rolling Std Dev', color='green')
   plt.title(f'Rolling Mean and Standard Deviation (window={window})')
   plt.xlabel('Date')
   plt.ylabel('Value')
   plt.legend()
   plt.show()

# 3. Seasonal Decomposition of Time Series
def seasonal_decomposition(data, model='additive', freq=365):
   result = seasonal_decompose(data['Value'], model=model, period=freq)
   result.plot()
   plt.show()

# 4. Autocorrelation Plot
def plot_autocorrelation(data):
   plt.figure(figsize=(10, 6))
   autocorrelation_plot(data['Value'])
   plt.title('Autocorrelation Plot')
   plt.show()
```

```
5. Resampling and Aggregation (e.g., Monthly mean)
def plot_resampled_data(data, freq='M'):
   monthly_mean = data.resample(freq).mean()
   plt.figure(figsize=(10, 6))
   plt.plot(monthly_mean.index, monthly_mean['Value'], label=f'{freq} Mean',
color='orange')
   plt.title(f'Time Series Resampled ({freq})')
   plt.xlabel('Date')
   plt.ylabel('Value')
   plt.legend()
   plt.show()

# 6. Heatmap of Time Series Data (Monthly Averages)
def plot_heatmap(data):
   data['Year'] = data.index.year
   data['Month'] = data.index.month
   monthly_data = data.pivot_table(values='Value', index='Year',
columns='Month', aggfunc='mean')

   plt.figure(figsize=(12, 8))
   sns.heatmap(monthly_data, cmap='coolwarm', annot=True, fmt=".1f")
   plt.title('Heatmap of Monthly Averages')
   plt.xlabel('Month')
   plt.ylabel('Year')
   plt.show()

# Applying various time series visualizations
plot_time_series(data)
plot_rolling_statistics(data, window=30)
seasonal_decomposition(data)
plot_autocorrelation(data)
plot_resampled_data(data, freq='M')
plot_heatmap(data)
```