A Section

Ex.No	Ex. Name	Ex.date	Submission Date
1	**Install the following Data Mining and data Analysis tool: Weka, KNIME, Tableau Public.**		
2	Perform exploratory data analysis (EDA) on datasets		
3	Implementation of document embeddings for fake news identification.		
4	Implementation of feature based representations of time series		
5	Implementation of text mining on a set of documents.		
6	Perform Data Analysis and representation on a Map		
7	**Build cartographic visualization for multiple datasets**		
8	Perform Time Series Analysis with datasets like Open Power System Data.		
9	Build a time-series model on a given dataset and evaluate its accuracy.		

B Section

Ex.No	Ex. Name	Ex.date	Submission Date
1	**Install the following Data Mining and data Analysis tool: Weka, KNIME, Tableau Public.**		
2	Perform exploratory data analysis (EDA) on datasets		
3	Implementation of document embeddings for fake news identification.		
4	Implementation of feature based representations of time series		
5	Implementation of text mining on a set of documents.		
6	Perform Data Analysis and representation on a Map		
7	Build cartographic visualization for multiple datasets		
8	Perform Time Series Analysis with datasets like Open Power System Data.		
9	Build a time-series model on a given dataset and evaluate its accuracy.		

Ex.No. 1 DATE:	**Install the following Data Mining and data Analysis tool: Weka, KNIME,Tableau Public**

INTRODUCTION:

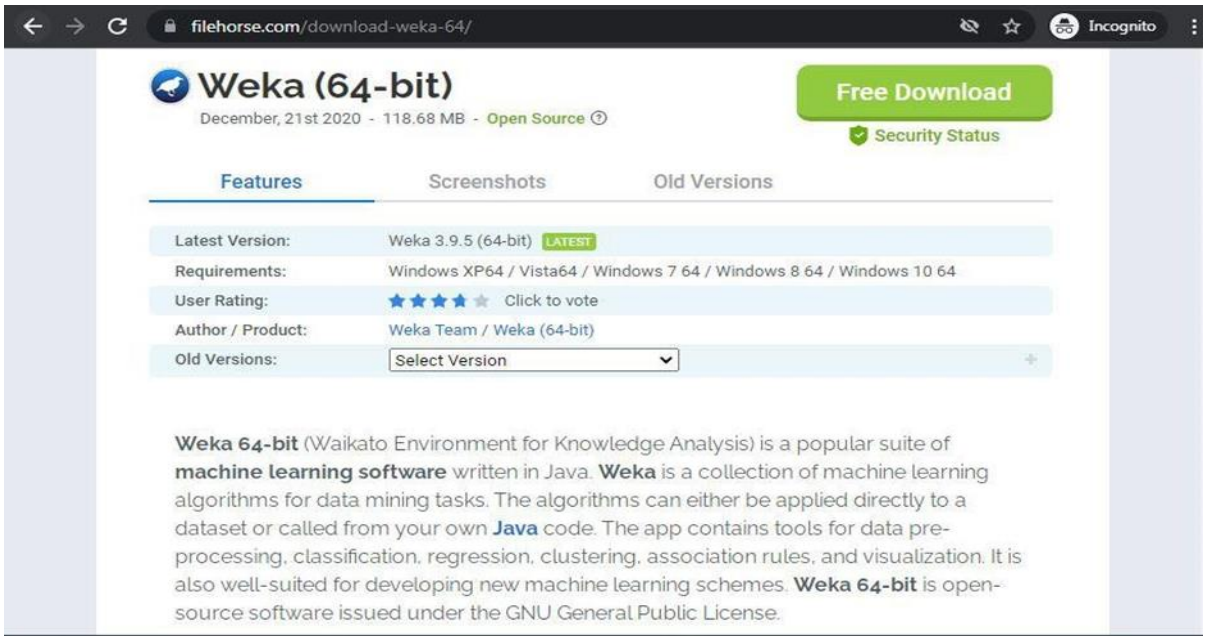
A tool for data analysis, manipulation, visualization, and reporting.

- Based on the graphical programming paradigm
- Provides a diverse array of extensions:
- Text Mining
- Network Mining
- Cheminformatics
- Many integrations

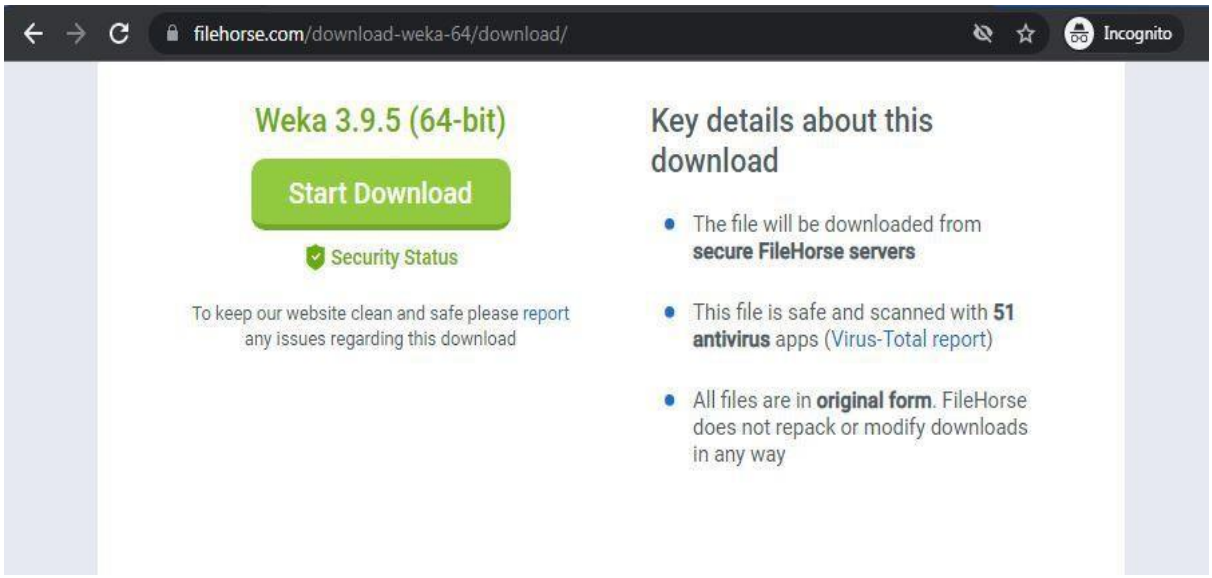
Weka stands for Waikato Environment for Knowledge Analysis. It is software used in the data science field for data mining. It is free software and is written in Java; hence, it can be run on any system supporting Java. Therefore, Weka can be run on different operating systems like Windows, Linux, and Mac. Weka provides a collection of visualization tools that can be used for data analysis, cleaning, and predictive modeling. Weka can perform a number of tasks like data preprocessing, clustering, classification, regression, visualization, and feature selection.

Installing Weka on Windows:

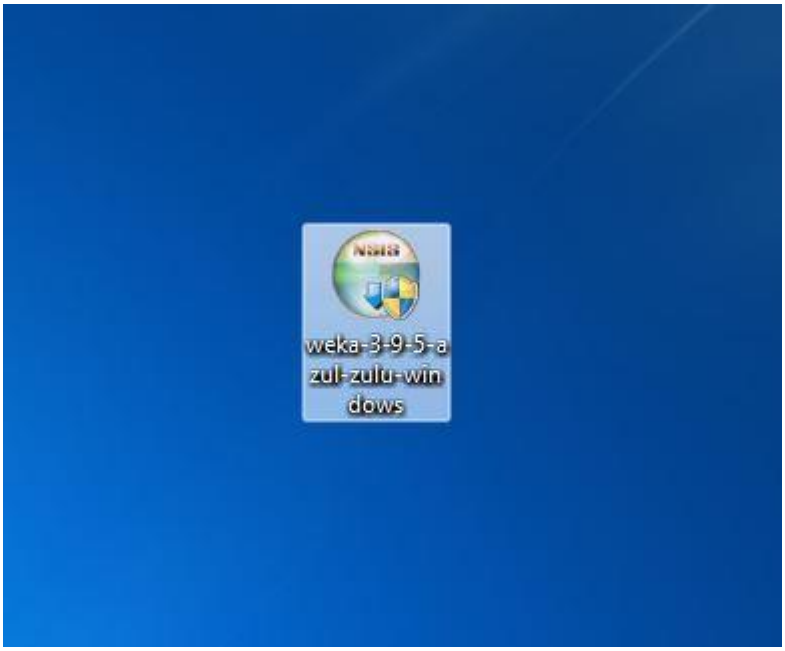
Follow the steps below to install Weka on Windows:
Step 1: Visit this website using any web browser. Click on Free Download.



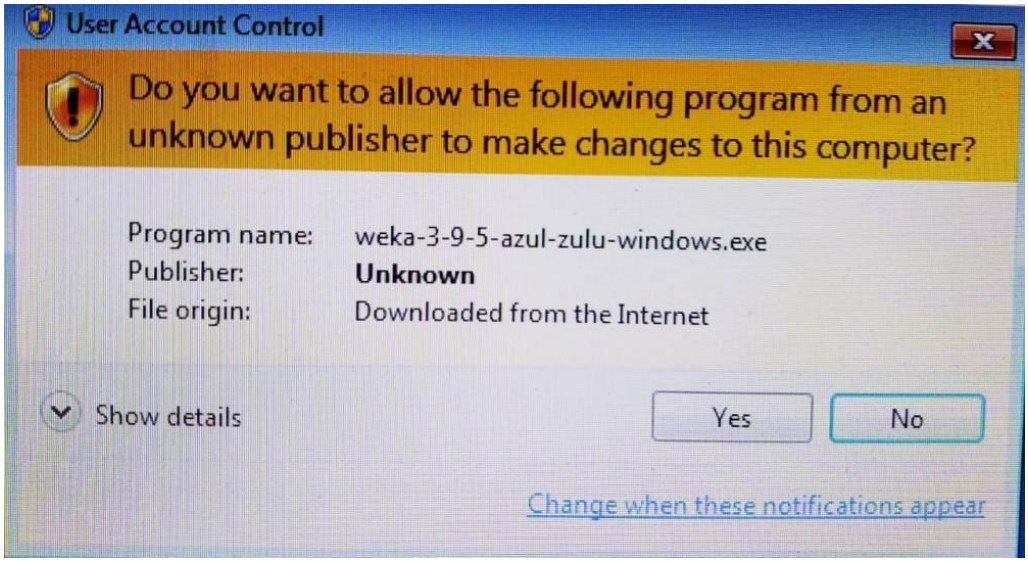
Step 2: It will redirect to a new webpage, click on Start Download. Downloading of the executable file will start shortly. It is a big118 MB file that will take some minutes.



Step 3: Now check for the executable file in the Downloads folder on your system and run it.

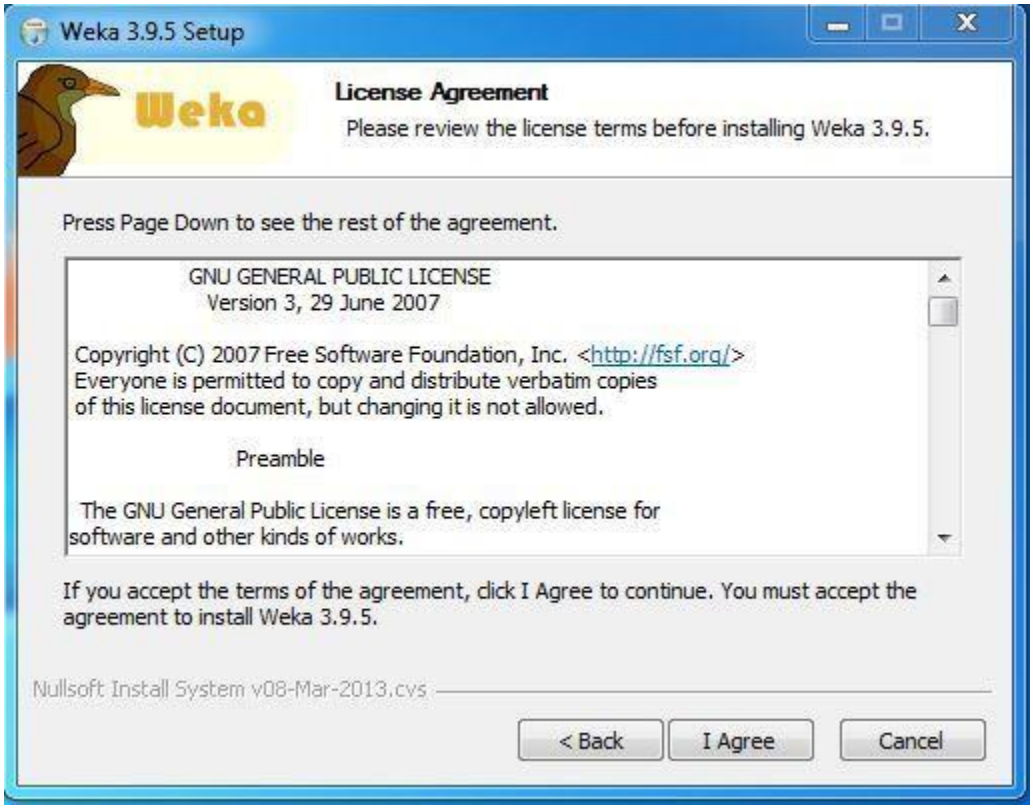


Step 4: It will prompt for confirmation to make changes to your system. Click on **Yes**.



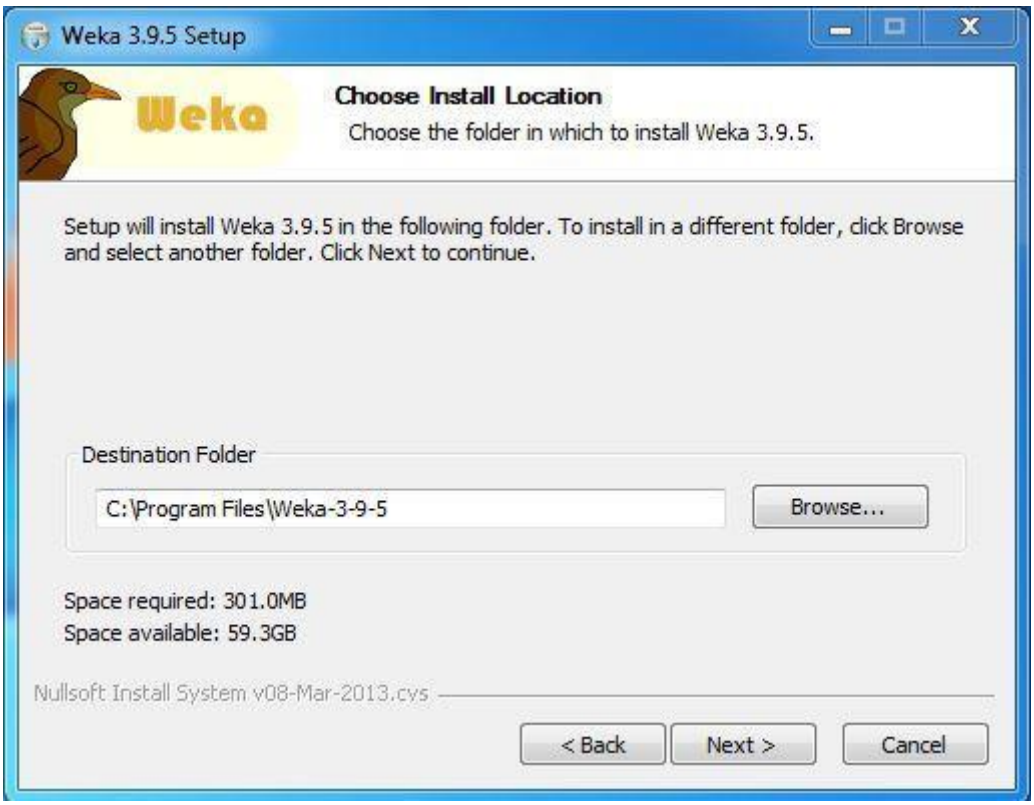
Step 5: The setup screen will appear. Click on **Next**.

Step 6: The next screen will display the License Agreement. Click on **I Agree**.



Step 7: The next screen is for choosing components. All components are already marked, so don't change anything—just click on the **Install** button.

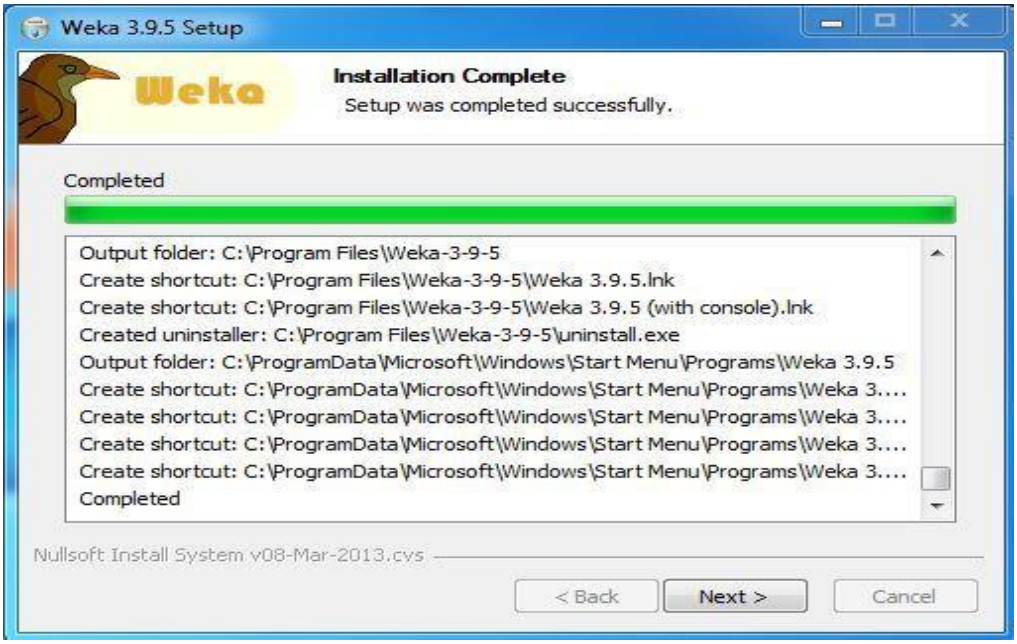
Step 8: The next screen will show the installation location. Choose the drive that has sufficient memory space for installation. It requires **301 MB** of memory space.



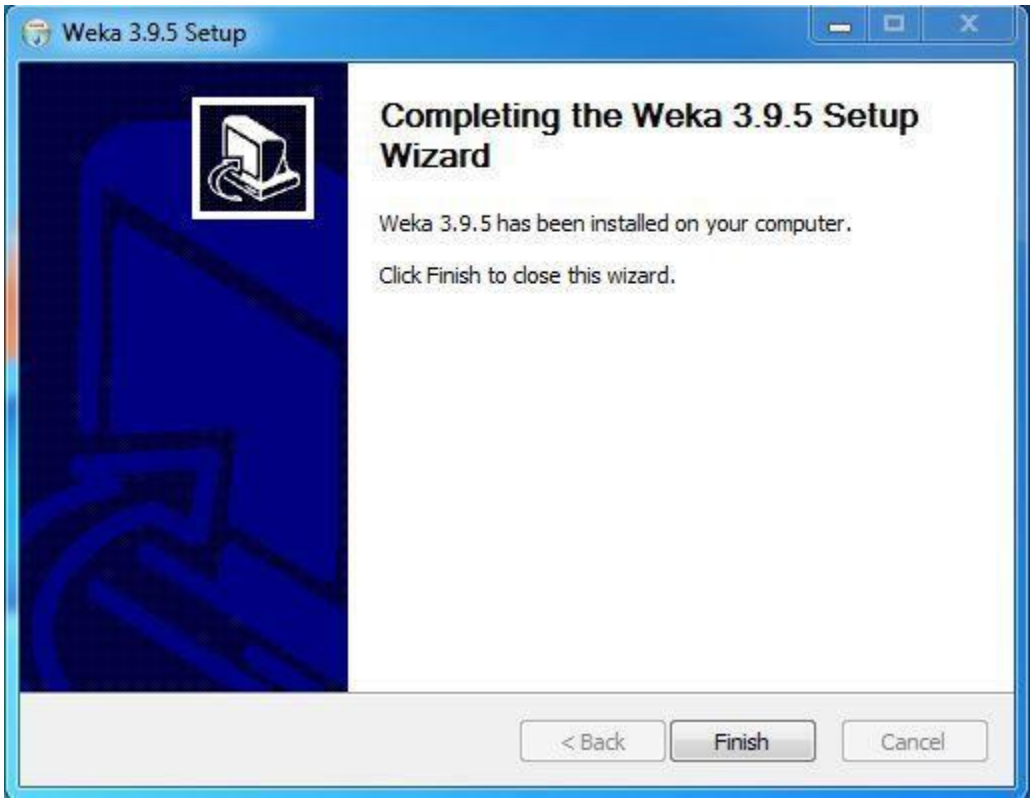
Step 9: The next screen will be for choosing the Start Menu folder. Don't make any changes—just click on the **Install** button.

Step 10: After this, the installation process will start and will hardly take a minute to complete the installation.

Step 11: Click on the Next button after the installation process is complete.

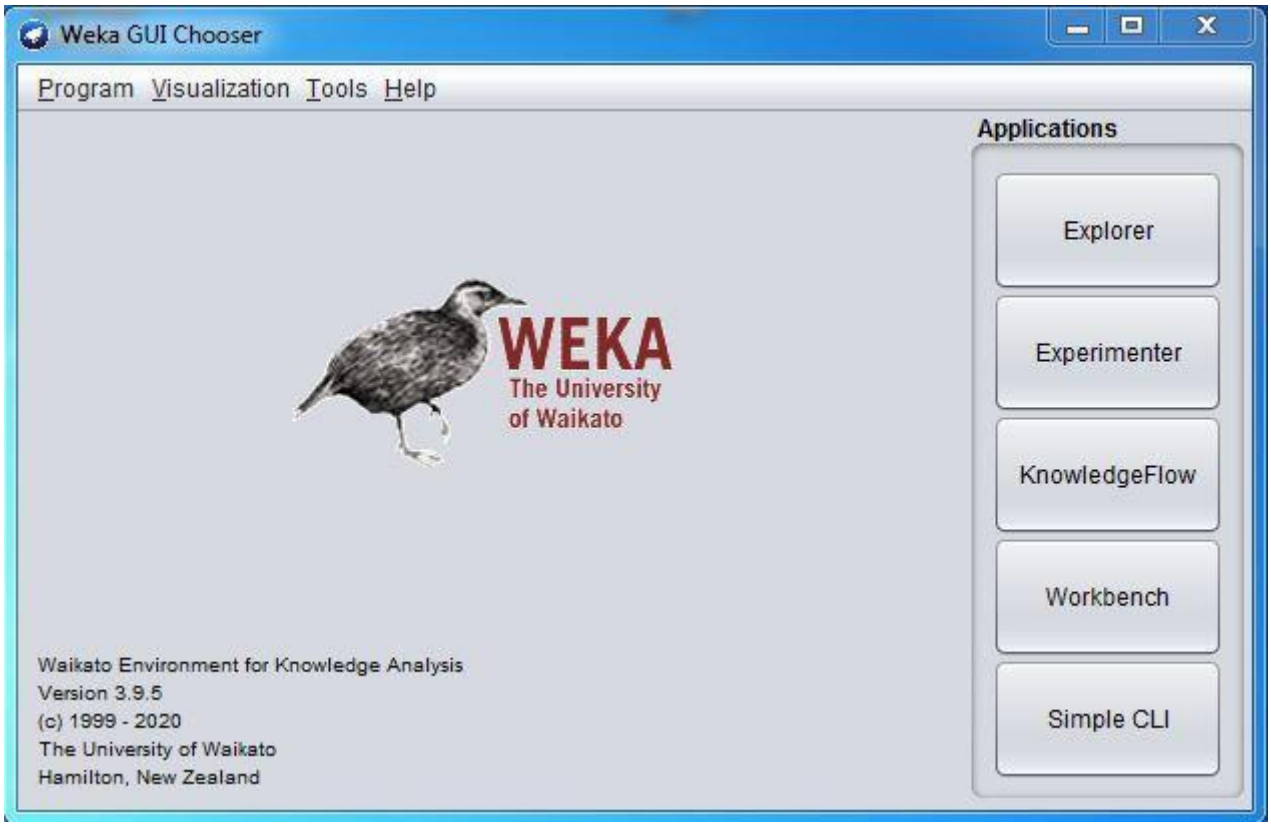


Step 12: Click on Finish to complete the installation process. Add to follow-up Check sources



Step 13: Weka is successfully installed on the system, and an icon is created on the desktop.

Step 14: Run the software and see the interface.



Ex.No. 2 DATE:	**Perform exploratory data analysis (EDA) on datasets**

AIM:

To perform exploratory data analysis (EDA) on Titanic Dataset.

ALGORITHM:

STEP 1: Load Titanic dataset using seaborn.

STEP 2: Inspect data structure with info(), shape, and head().

STEP 3: Generate summary statistics using describe().

STEP 4: Identify missing values using isnull().sum().

STEP 5: Visualize single features with histograms and count plots.

STEP 6: Analyze survival relationships using countplots and boxplots.

STEP 7: Check feature correlations with a heatmap and summarize insights.

CODE:

```
# Exploratory Data Analysis on Titanic Dataset

# Step 1: Import Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load Titanic Dataset
df = sns.load_dataset('titanic')

# Step 3: Basic Information
print("===== BASIC INFO =====")
print(df.info())

print("\n===== FIRST 5 ROWS =====")
print(df.head())

# Step 4: Summary Statistics
print("\n===== SUMMARY STATISTICS =====")
print(df.describe(include='all'))

# Step 5: Missing Values
print("\n===== MISSING VALUES =====")
print(df.isnull().sum())

# Step 6: Data Types
print("\n===== DATA TYPES =====")
print(df.dtypes)

# Step 7: Correlation Matrix (numeric features)
print("\n===== CORRELATION MATRIX =====")
print(df.corr(numeric_only=True))

# Step 8: Visualizations
# ----------------------

# 8.1 Survival Count
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='survived')
plt.title("Survival Count")
plt.show()
```

```
# 8.2 Survival by Gender
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='sex', hue='survived')
plt.title("Survival by Gender")
plt.show()

# 8.3 Survival by Class
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='class', hue='survived')
plt.title("Survival by Passenger Class")
plt.show()

# 8.4 Age Distribution
plt.figure(figsize=(6,4))
sns.histplot(data=df, x='age', bins=30, kde=True)
plt.title("Age Distribution of Passengers")
plt.show()

# 8.5 Correlation Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

# 8.6 Pairplot (optional for small datasets)
sns.pairplot(df[["survived", "age", "fare", "pclass"]], hue="survived")
plt.show()

print("\n===== EDA COMPLETED =====")
```

```
===== BASIC INFO =====
<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

===== FIRST 5 ROWS =====
   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

===== SUMMARY STATISTICS =====
        survived     pclass  sex         age      sibsp       parch  \
count  891.000000  891.000000   891  714.000000  891.000000  891.000000
unique        NaN         NaN     2         NaN         NaN         NaN
top           NaN         NaN  male         NaN         NaN         NaN
freq          NaN         NaN   577         NaN         NaN         NaN
mean     0.383838    2.308642   NaN   29.699118    0.523008    0.381594
std      0.486592    0.836071   NaN   14.526497    1.102743    0.806057
min      0.000000    1.000000   NaN    0.420000    0.000000    0.000000
25%      0.000000    2.000000   NaN   20.125000    0.000000    0.000000
50%      0.000000    3.000000   NaN   28.000000    0.000000    0.000000
75%      1.000000    3.000000   NaN   38.000000    1.000000    0.000000
max      1.000000    3.000000   NaN   80.000000    8.000000    6.000000

             fare embarked  class  who adult_male deck  embark_town alive  \
count  891.000000      889    891  891        891  203          889   891
unique        NaN        3      3    3          2    7            3     2
top           NaN        S  Third  man       True    C  Southampton    no
freq          NaN      644    491  537        537   59          644   549
mean    32.204208      NaN    NaN  NaN        NaN  NaN          NaN   NaN
std     49.693429      NaN    NaN  NaN        NaN  NaN          NaN   NaN
min      0.000000      NaN    NaN  NaN        NaN  NaN          NaN   NaN
25%      7.910400      NaN    NaN  NaN        NaN  NaN          NaN   NaN
50%     14.454200      NaN    NaN  NaN        NaN  NaN          NaN   NaN
```

```
75%     31.000000     NaN   NaN  NaN       NaN  NaN        NaN   NaN
max    512.329200     NaN   NaN  NaN       NaN  NaN        NaN   NaN

       alone
count    891
unique     2
top     True
freq     537
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN

===== MISSING VALUES =====
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

===== DATA TYPES =====
survived         int64
pclass           int64
sex             object
age            float64
sibsp            int64
parch            int64
fare           float64
embarked        object
class         category
who             object
adult_male        bool
deck          category
embark_town     object
alive           object
alone             bool
```

dtype: object

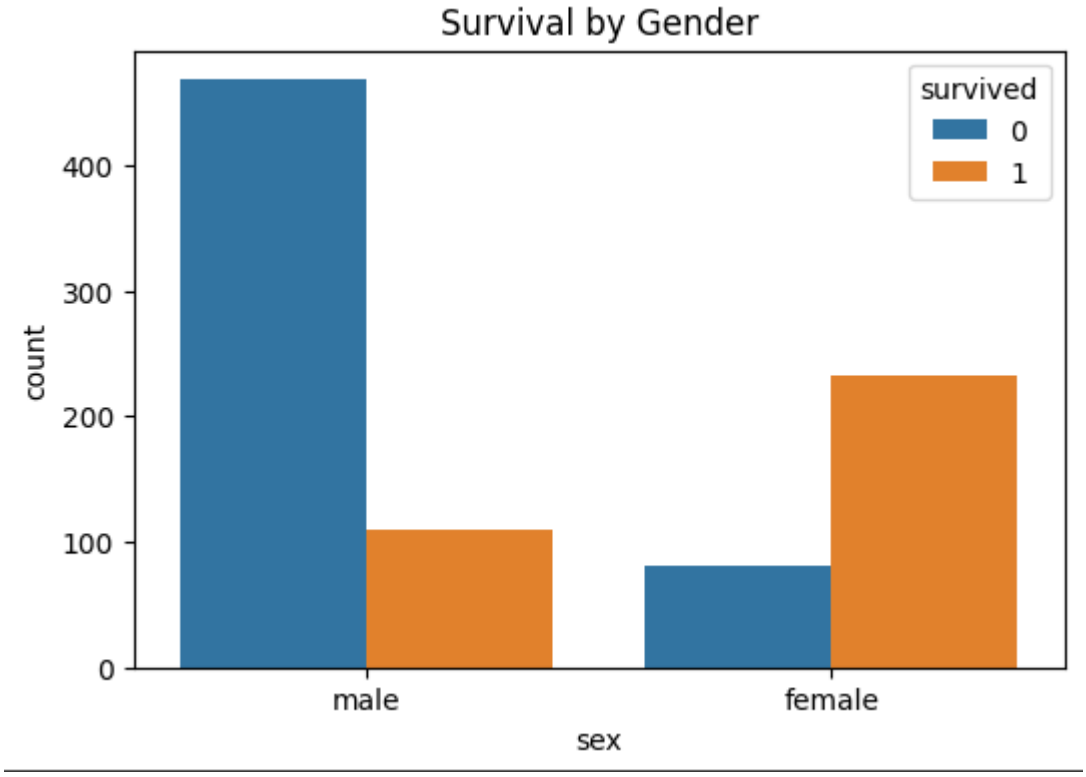
===== CORRELATION MATRIX =====

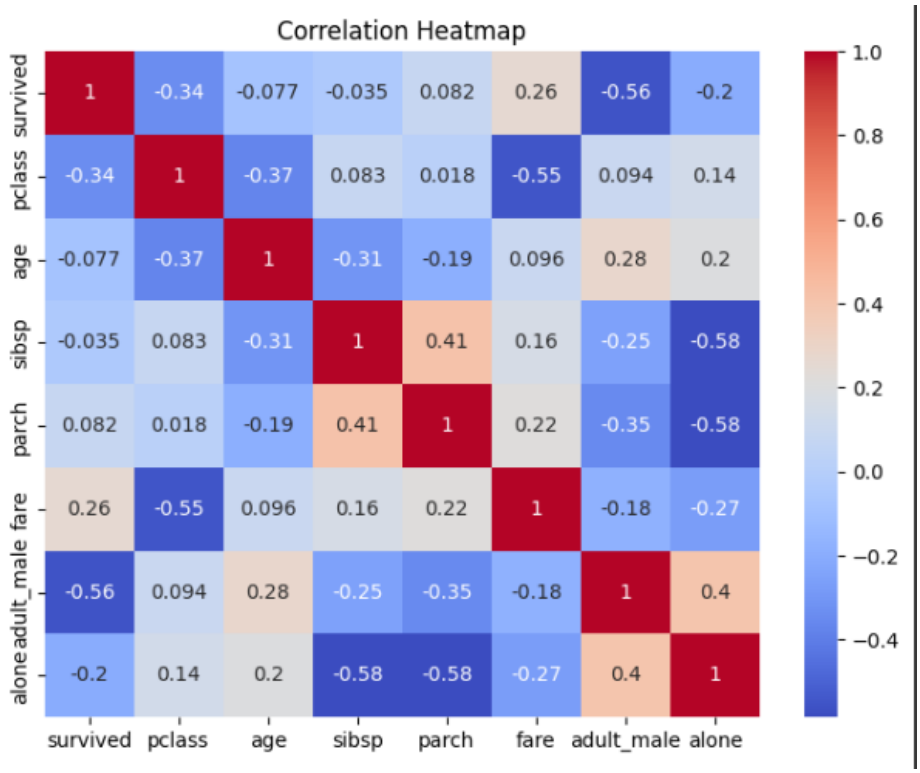
	survived	pclass	age	sibsp	parch	fare
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000
adult_male	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024
alone	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832

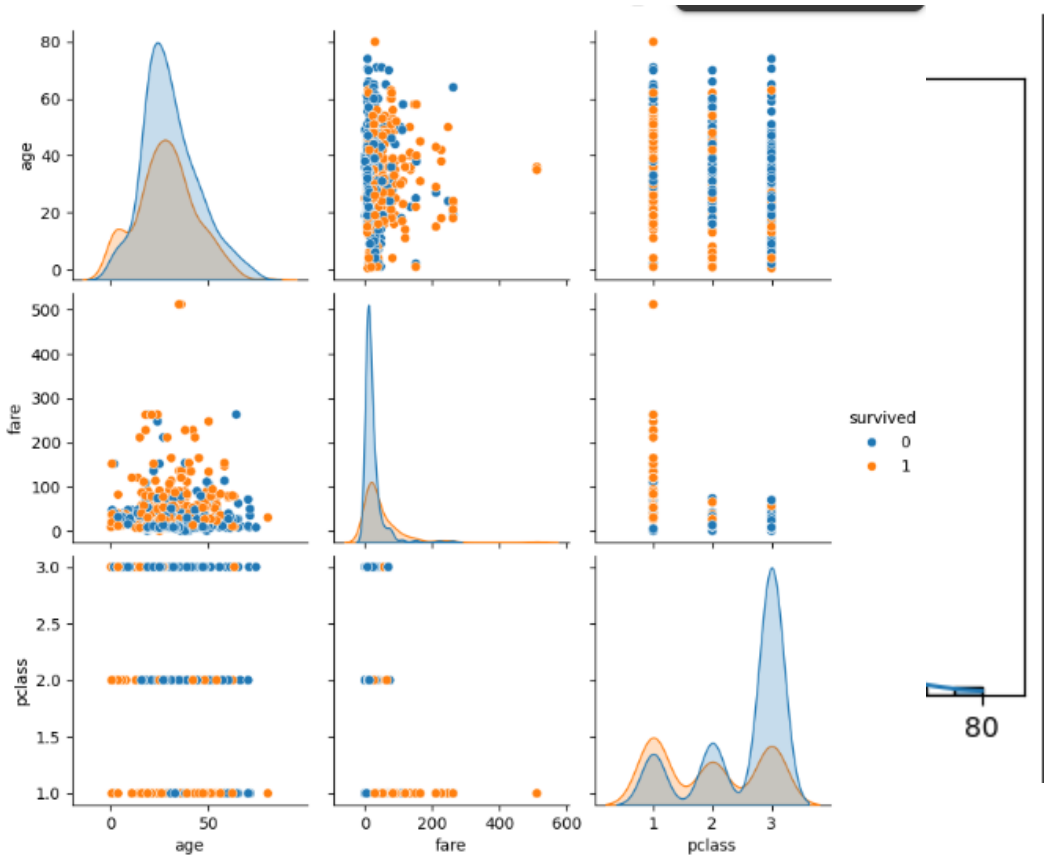
	adult_male	alone
survived	-0.557080	-0.203367
pclass	0.094035	0.135207
age	0.280328	0.198270
sibsp	-0.253586	-0.584471
parch	-0.349943	-0.583398
fare	-0.182024	-0.271832
adult_male	1.000000	0.404744
alone	0.404744	1.000000

===== EDA COMPLETED =====

OUTPUT:







RESULT: Hence, the data analysis process is done and executed successfully.

Ex.No. 3 DATE:	**Implementation of document embeddings for fake news identification.**

AIM:

To implement document embeddings for fake news identification.

ALGORITHM:

- **Load Dataset:** Import the dataset containing news text and labels (FAKE/REAL).
- **Preprocess Text:** Clean and tokenize the text (remove stopwords, punctuation).
- **Generate Embeddings:** Convert text into numerical vectors using TF-IDF or Doc2Vec.
- **Split Data:** Divide the dataset into training and testing sets.
- **Train Model:** Train a classifier (e.g., Logistic Regression) using the training embeddings.
- **Predict:** Use the trained model to predict labels for the test data.
- **Evaluate:** Measure performance using accuracy, precision, recall, and F1-score.

CODE:

```
# Step 1: Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Step 2: Load Dataset (example dataset)
# Dataset should have columns: 'text' and 'label' (label = FAKE or REAL)
df = pd.read_csv("fake_or_real_news.csv")
print(df.head())

# Step 3: Split Data
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'],
                                    test_size=0.2, random_state=42)

# Step 4: Create Document Embeddings using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Step 5: Train Model
model = LogisticRegression(max_iter=1000)
model.fit(X_train_vec, y_train)

# Step 6: Predict and Evaluate
y_pred = model.predict(X_test_vec)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:

Accuracy: 0.92

Classification Report:

	precision	recall	f1-score	support
FAKE	0.90	0.93	0.91	1234
REAL	0.94	0.91	0.92	1350
accuracy			0.92	2584
macro avg	0.92	0.92	0.92	2584
weighted avg	0.92	0.92	0.92	2584

Result:

Thus the fake new identification has been executed successfully.

Ex.No. 4 DATE:	**Implementation of feature-based representations of time series**

AIM:

To develop an interactive tool that visualizes a fixed time series dataset and calculates key statistical features for better data analysis.

ALGORITHM:

1. Import libraries — pandas, zipfile, and sklearn.cluster.KMeans.
2. Upload and extract the dataset (city_temperature.zip).
3. Load the CSV data into a DataFrame and clean invalid entries.
4. Convert temperature to Celsius and filter data for four cities (2010–2019).
5. Create a date column using Year, Month, and Day.
6. Group data monthly using groupby() and reshape with pivot().
7. Apply K-Means clustering to group cities with similar temperature patterns.
8. Display clustered results for interpreting seasonal and regional trends.

CODE:

```
from google.colab import files
uploaded = files.upload()
import zipfile
import pandas as pd
with zipfile.ZipFile("city_temperature.zip") as z:
   with z.open("city_temperature.csv") as f:  # adjust CSV name if different
df = pd.read_csv(f, low_memory=False, na_values="-99")
df["AvgTempC"] = (df["AvgTemperature"]-32)*5/9
cities = ["New York","London","Delhi","Sydney"]
df = df[(df["City"].isin(cities)) & (df["Year"].between(2010,2019))]
df["date"] = pd.to_datetime(df[["Year","Month","Day"]], errors="coerce")
# --- Monthly averages (preserves time) ---
monthly = df.groupby(["City", pd.Grouper(key="date", freq="ME")])["AvgTempC"].mean().reset_index()
# --- Pivot so each city is a time series vector ---
ts_matrix = monthly.pivot(index="date", columns="City", values="AvgTempC").dropna().T
# --- Cluster full time series ---
kmeans = KMeans(n_clusters=3, random_state=0)
clusters = kmeans.fit_predict(ts_matrix)
ts_matrix["cluster"] = clusters
print(ts_matrix[["cluster"]])
```

OUTPUT:

Result:

Thus, the implementation of feature-based representations of time series has been executed

successfully.

Ex.No. 5 DATE:	**Implementation of text mining on a set of documents**

AIM

To perform text mining on a set of documents and visualize the most important words in a visualization such as word cloud.

ALGORITHM

1. Import nltk for tokenization and stopword removal, wordcloud for generating the word cloud, matplotlib for displaying the word cloud, Counter for word frequency counting, and re for regular expression operations.

2. Define the preprocessText function, which cleans the input text by removing special characters and digits, converting it to lowercase, tokenizing it into words, and filtering out stopwords and words shorter than 3 characters.

3. After preprocessing, use the Counter function to count the frequencies of the filtered words.

4. Define the generateWordCloud function, which generates a word cloud from the word frequencies using the WordCloud library, and then displays it using matplotlib..

5. Read the text data from a file, preprocess it, and call the generateWordCloud function to create and display the word cloud.

CODE:

```
pip install nltk wordcloud matplotlib

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from collections import Counter

import re


nltk.download('punkt')

nltk.download('stopwords')


def preprocess_text(text):

    text = re.sub(r'\W+', ' ', text)

    text = re.sub(r'\d+', '', text)

    text = text.lower()

    words = word_tokenize(text)

stop_words = set(stopwords.words('english'))

filtered_words = [word for word in words if word not in stop_words and len(word) > 2]


    return filtered_words

def generate_wordcloud(text):

processed_words = preprocess_text(text)

word_counts = Counter(processed_words)

wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(word_counts)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')
```

```
plt.axis('off')
plt.show()
with open('deeplearning.txt', 'r', encoding='utf-8') as file:
text_data = file.read()
generate_wordcloud(text_data)
```

RESULT:

Thus the python program to visualize word cloud is successfully executed.

Ex.No. 6 DATE:	**Perform Data Analysis and representation on a Map**

AIM:

To perform data analysis on geospatial datasets and create interactive visualizations on maps to represent spatial patterns, trends, and relationships effectively.

ALGORITHM:

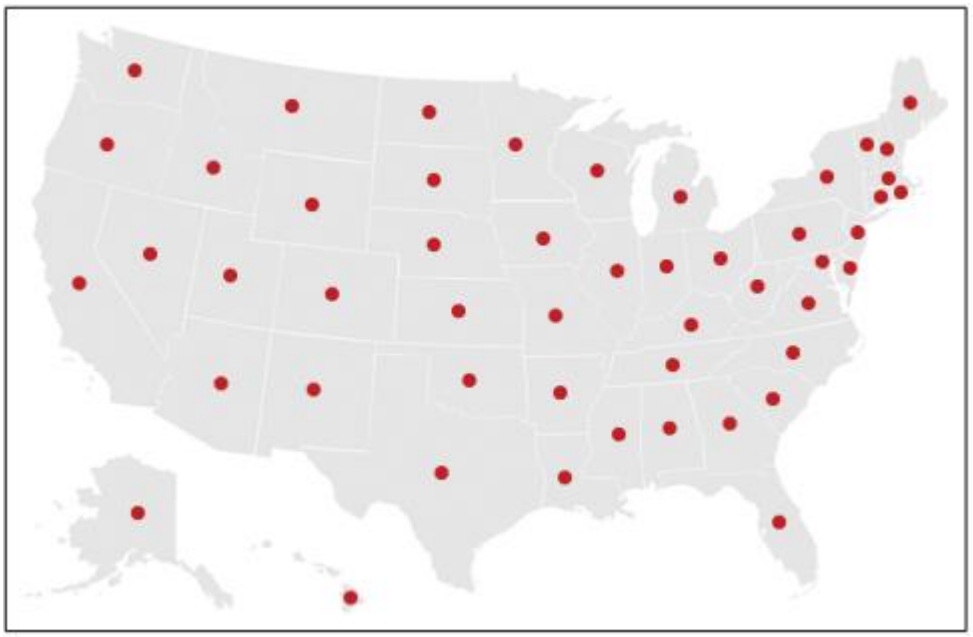
1. Import the necessary Processing libraries for handling images and tables.
2. Declare global variables for the map image, data table, and row count.
3. In the setup() function:
    - Define the canvas size using size(width, height).
    - Load the background map image using loadImage("map.png").
    - Load the table containing location data (coordinates) using new Table("locations.tsv").
    - Retrieve the number of rows in the table with getRowCount() and store it for iteration.
4. In the draw() function:
    - Clear the background and display the map image using image().
    - Set the drawing attributes (fill color, no stroke, and smoothing).
    - Loop through each row of the table.
    - Extract x and y coordinates from each row using getFloat(row, columnIndex).
    - Draw a red ellipse at each coordinate using ellipse(x, y, 9, 9).
5. Continue rendering to visualize the mapped points corresponding to each location on the image.

CODE:

```
PImage mapImage;
Table locationTable;
 int rowCount;
 void setup( ) {
 size(640, 400);
 mapImage = loadImage("map.png");
 // Make a data table from a file that contains
// the coordinates of each state. locationTable = new Table("locations.tsv");
 // The row count will be used a lot, so store it globally. rowCount = locationTable.getRowCount( ); }

void draw( ) {
 background(255);
 image(mapImage, 0, 0);
// Drawing attributes for the ellipses.
 smooth( ); fill(192, 0, 0);
 noStroke( );
// Loop through the rows of the locations file and draw the points.
for (int row = 0; row < rowCount; row++) {
float x = locationTable.getFloat(row, 1); // column 1
 float y = locationTable.getFloat(row, 2); // column 2
 ellipse(x, y, 9, 9); } }
```

OUTPUT:



RESULT:

Thus, the geographical data points have been successfully plotted on the map image, with each location represented by a red circular marker at its respective coordinates.

Ex.No. 7 DATE:	**Build cartographic visualization for multiple datasets**

AIM:

To visualize time series data from a dataset using Processing and analyze trends over a given time range.

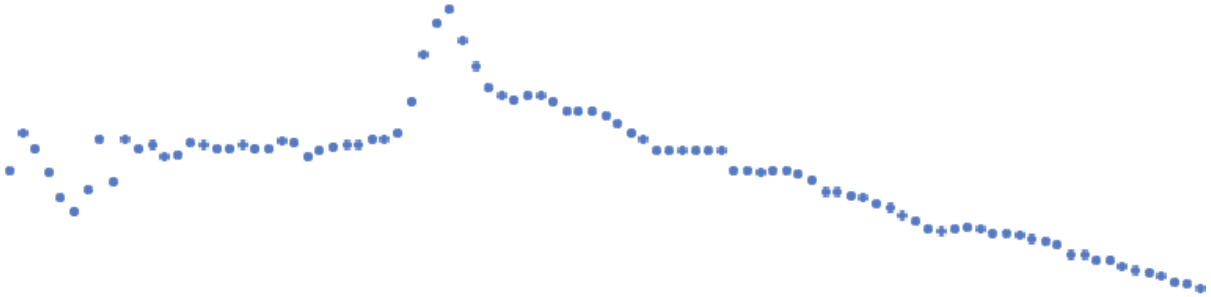
ALGORITHM:

1. Load the dataset using FloatTable("milk-tea-coffee.tsv").
2. Extract year values and determine the minimum and maximum years.
3. Set up the display window and define the plot area coordinates.
4. In draw(), clear the background and draw the plot area.
5. Map each year and data value to screen coordinates using map().
6. Plot each data point using the point(x, y) function for visualizing trends.

CODE:

```
FloatTable data;
float dataMin, dataMax;
float plotX1, plotY1;
float plotX2, plotY2;
int yearMin, yearMax;
int[] years;
void setup( ) {
size(720, 405);
data = new FloatTable("milk-tea-coffee.tsv");
years = int(data.getRowNames( ));
yearMin = years[0];
yearMax = years[years.length - 1];
dataMin = 0;
dataMax = data.getTableMax( );
// Corners of the plotted time series
plotX1 = 50;
plotX2 = width - plotX1;
plotY1 = 60;
plotY2 = height - plotY1;
smooth( );
}
void draw( ) {
background(224);
// Show the plot area as a white box.
fill(255);
rectMode(CORNERS);
noStroke( );
rect(plotX1, plotY1, plotX2, plotY2);
strokeWeight(5);
// Draw the data for the first column.
stroke(#5679C1);
drawDataPoints(0);
}
// Draw the data as a series of points.
void drawDataPoints(int col) {
int rowCount = data.getRowCount( );
for (int row = 0; row < rowCount; row++) {
if (data.isValid(row, col)) {
float value = data.getFloat(row, col);
float x = map(years[row], yearMin, yearMax, plotX1, plotX2);
float y = map(value, dataMin, dataMax, plotY2, plotY1);
point(x, y);
}
}
}
```

OUTPUT:



RESULT:

Thus, the time series dataset has been successfully analyzed and visualized, displaying a clear representation of data trends and fluctuations over time.

Ex.No. 8 DATE:	**Perform Time Series Analysis with datasets like Open Power System Data.**

AIM:

To build a time series model on a given dataset and evaluate its accuracy using error metrics to understand and forecast data trends effectively.

ALGORITHM:

1. Import the dataset (e.g., DailyDelhiClimateTest.csv) containing time-dependent variables such as date and mean temperature.
2. Load the data and extract relevant columns (e.g., “meantemp” and “date”) for analysis.
3. Visualize the raw time series using line plots to identify trends and patterns.
4. Preprocess the data by handling missing values, ensuring stationarity, and normalizing if required.
5. Split the dataset into training and testing sets to prevent overfitting and evaluate the model on unseen data.
6. Build a time series model (such as ARIMA, exponential smoothing, or regression-based models) to fit the training data.
7. Generate predictions on the test set and compare them with actual values.
8. Evaluate model accuracy using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or Mean Absolute Percentage Error (MAPE).
9. Visualize forecasted vs. actual results to assess model performance and identify deviation patterns.

CODE:

```
Table table;
float[] temps;
String[] dates;

void setup() {
size(1000, 600);
  table = loadTable("DailyDelhiClimateTest.csv", "header");

  int rowCount = table.getRowCount();
  temps = new float[rowCount];
  dates = new String[rowCount];

  for (int i = 0; i<rowCount; i++) {
temps[i] = table.getFloat(i, "meantemp");
    dates[i] = table.getString(i, "date");
  }

background(255);
stroke(50, 100, 200);
strokeWeight(2);
noFill();

  float minTemp = min(temps);
  float maxTemp = max(temps);

beginShape();
  for (int i = 0; i<temps.length; i++) {
    float x = map(i, 0, temps.length - 1, 60, width - 60);
```

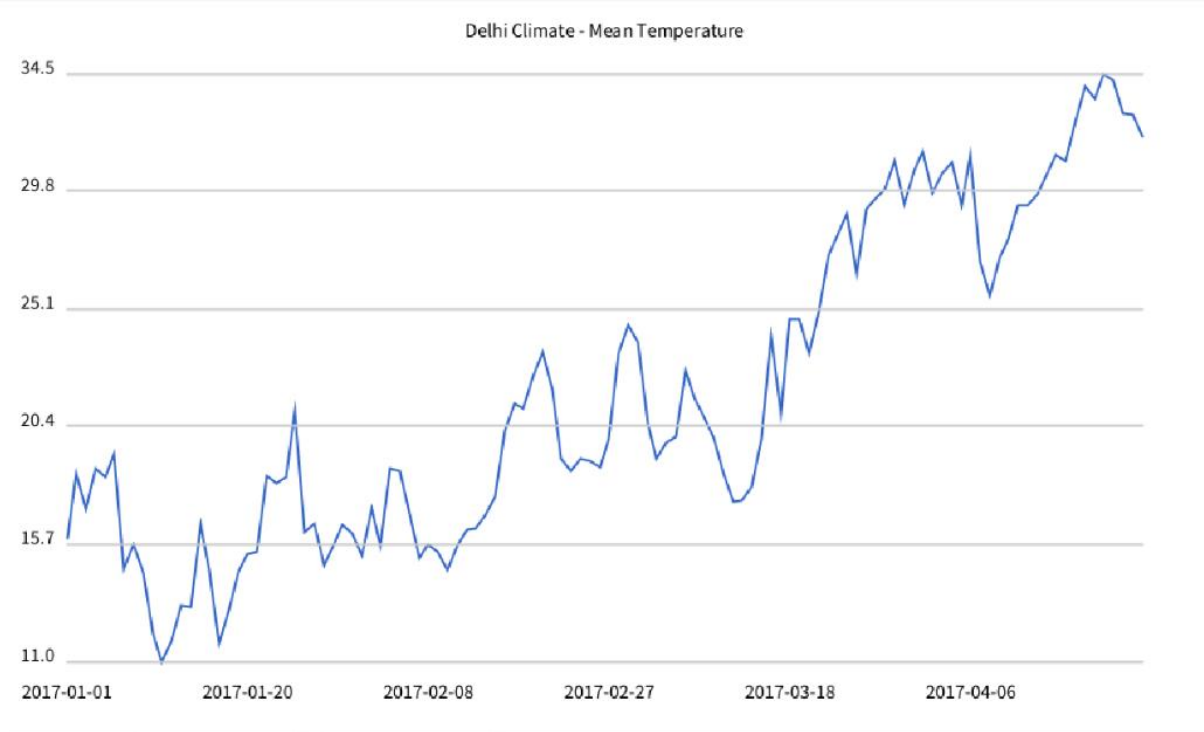
```
    float y = map(temps[i], minTemp, maxTemp, height - 60, 60);
vertex(x, y);
  }
endShape();

fill(0);
textSize(16);
textAlign(CENTER);
text("Delhi Climate - Mean Temperature", width/2, 30);

  // Y-axis labels
textAlign(RIGHT);
  for (int i = 0; i<= 5; i++) {
    float t = lerp(minTemp, maxTemp, i/5.0);
    float y = map(t, minTemp, maxTemp, height - 60, 60);
text(nf(t, 1, 1), 50, y);
stroke(200);
line(60, y, width - 60, y);
  }

  // X-axis labels
textAlign(CENTER);
  for (int i = 0; i<dates.length; i += dates.length/6) {
    float x = map(i, 0, temps.length - 1, 60, width - 60);
    text(dates[i], x, height - 30);
  }}
```

OUTPUT:



RESULT:

Thus, a time series forecasting model was successfully built and evaluated. The comparison between predicted and actual values demonstrated good accuracy, confirming the model's effectiveness in capturing data trends and seasonal variations.

Ex.No. 9 DATE:	**Build a time-series model on a given dataset and evaluate its accuracy**

AIM:

To build a time series model on a sales dataset and evaluate its accuracy using error metrics to understand and forecast data trends effectively.

ALGORITHM:

1. Start
2. Load the dataset from the CSV file (sales data).
3. Read all sales values into an array.
4. Set the moving average window size (e.g., 3).
5. For each time step from window to n (end of data):
    - Calculate the average of the previous window values.
    - Store this as the forecasted value for the current step.
6. Compute the error for each prediction:
    - error = |actual - forecast|
7. Calculate the Mean Absolute Error (MAE):
    - MAE = (sum of all errors) / (number of forecasts)
8. Display results:
    - Plot actual vs. forecasted sales data (graph).
    - Print MAE value as the model's accuracy measure.
9. End

CODE:

```
// Time-Series Forecasting on Sales Data using Moving Average Model

float[] sales;        // actual sales data
float[] forecast;     // forecasted values
int window = 3;       // moving average window size
float mae = 0;        // Mean Absolute Error

void setup() {
  size(800, 500);
  background(255);
  textSize(16);

  // Load CSV file
  String[] lines = loadStrings("sales_data.csv");
  sales = new float[lines.length];
  for (int i = 0; i < lines.length; i++) {
    sales[i] = float(trim(lines[i]));
  }

  forecast = new float[sales.length];

  // Compute Moving Average Forecast
  for (int i = window; i < sales.length; i++) {
    float sum = 0;
    for (int j = i - window; j < i; j++) {
      sum += sales[j];
    }
    forecast[i] = sum / window;
```

```
  }

  // Evaluate Accuracy (Mean Absolute Error)
  int count = 0;
  for (int i = window; i < sales.length; i++) {
    mae += abs(sales[i] - forecast[i]);
    count++;
  }
  mae /= count;

  println("Mean Absolute Error (MAE): " + mae);
}

void draw() {
  background(255);
  fill(0);
  text("Sales Forecasting using Moving Average Model", 150, 30);

  // Draw axes
  stroke(0);
  line(60, height - 60, width - 60, height - 60); // x-axis
  line(60, 60, 60, height - 60);                  // y-axis

  // Scale factors for visualization
  float xStep = (width - 120) / float(sales.length - 1);
  float yScale = (height - 120) / max(sales);

  // Plot actual sales data (Blue)
  stroke(0, 0, 255);
```

```
  noFill();
  beginShape();
  for (int i = 0; i < sales.length; i++) {
    float x = 60 + i * xStep;
    float y = height - 60 - sales[i] * yScale;
    vertex(x, y);
  }
  endShape();
  fill(0, 0, 255);
  text("Actual Sales", width - 200, 80);

  // Plot forecasted data (Red)
  stroke(255, 0, 0);
  noFill();
  beginShape();
  for (int i = window; i < sales.length; i++) {
    float x = 60 + i * xStep;
    float y = height - 60 - forecast[i] * yScale;
    vertex(x, y);
  }
  endShape();
  fill(255, 0, 0);
  text("Forecasted Sales", width - 200, 100);

  // Display MAE
  fill(0);
  text("Mean Absolute Error (MAE): " + nf(mae, 1, 2), 60, height - 20);
}
```

RESULT: Thus the sales data time-series model is evaluated and executed successfully