

**D. Y. PATIL COLLEGE OF ENGINEERING, PUNE****Department of  
Artificial Intelligence and Data Science****LAB MANUAL  
BE  
Computer Laboratory-I  
Semester-I**

**Prepared By:**  
**Mrs. Rasika Wattamwar**  
**Mrs. Vaishnavi Jagtap**  
**Mrs. Aparna Lavangade**



**Computer Laboratory -I**

Course Code	Course Name	Teaching Scheme (Hrs./ Week)	Credits
417525	Computer Laboratory-I: Machine Learning, Data Modeling and Visualization	4	2

**Course Objectives:**

- Apply regression, classification and clustering algorithms for creation of ML models
- Introduce and integrate models in the form of advanced ensembles
- Conceptualized representation of Data objects
- Create associations between different data objects, and the rules
- Organized data description, data semantics, and consistency constraints of data

**Course Outcomes:**

After completion of the course, learners should be able to-

CO1: Implement regression, classification and clustering models

CO2: Integrate multiple machine learning algorithms in the form of ensemble learning

CO3: Apply reinforcement learning and its algorithms for real world applications

CO4: Analyze the characteristics, requirements of data and select an appropriate data model

CO5: Apply data analysis and visualization techniques in the field of exploratory data science

CO6: Evaluate time series data.

**Operating System recommended:** 64-bit Open source Linux or its derivative

Assignment No.	Title of the Experiment	Page No
1	<p><b>Feature Transformation (Any one)</b></p> <p><b>A.</b> To use PCA Algorithm for dimensionality reduction. You have a dataset that includes measurements for different variables on wine (alcohol, ash, magnesium, and so on). Apply PCA algorithm &amp; transform this data so that most variations in the measurements of the variables are captured by a small number of principal components so that it is easier to distinguish between red and white wine by inspecting these principal components. Dataset Link: <a href="https://media.geeksforgeeks.org/wp-content/uploads/Wine.csv">https://media.geeksforgeeks.org/wp-content/uploads/Wine.csv</a></p> <p><b>B.</b> Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link: <a href="https://www.kaggle.com/datasets/uciml/iris">https://www.kaggle.com/datasets/uciml/iris</a></p>	9
2	<p><b>Regression Analysis:(Any one)</b></p> <p><b>A.</b> Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and ridge, Lasso regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset <a href="https://www.kaggle.com/datasets/yasserh/uber-fares-dataset">https://www.kaggle.com/datasets/yasserh/uber-fares-dataset</a></p> <p><b>B.</b> Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets Dataset link: <a href="https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database">https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database</a></p>	20
3	<p><b>Classification Analysis (Any one)</b></p> <p><b>A.</b> Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).</p> <p><b>B.</b> Implement K-Nearest Neighbours" algorithm on Social network ad dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset. Dataset link: <a href="https://www.kaggle.com/datasets/rakeshrau/social-network-ads">https://www.kaggle.com/datasets/rakeshrau/social-network-ads</a></p>	38
4	<b>Clustering Analysis (Any one)</b>	52

	A.Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method. Dataset Link: <a href="https://www.kaggle.com/datasets/uciml/iris">https://www.kaggle.com/datasets/uciml/iris</a>  B. Implement K-Mediod Algorithm on a credit card dataset. Determine the number of clusters using the Silhouette Method. Dataset Link:	
5	<b>Ensemble Learning (Any one)</b>  A. Implement Random Forest Classifier model to predict the safety of the car. Dataset link: <a href="https://www.kaggle.com/datasets/elikplim/car-evaluation-data-set">https://www.kaggle.com/datasets/elikplim/car-evaluation-data-set</a>  B. Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient Tree Boosting (GBM), XGBoost classification on Iris dataset and compare the performance of three models using different evaluation measures. Dataset Link: <a href="https://www.kaggle.com/datasets/uciml/iris">https://www.kaggle.com/datasets/uciml/iris</a>	70
6	<b>Reinforcement Learning (Any one)</b>  A. Implement Reinforcement Learning using an example of a maze environment that the agent needs to explore.  B. Solve the Taxi problem using reinforcement learning where the agent acts as a taxi driver to pick up a passenger at one location and then drop the passenger off at their destination.  C. Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks a. Setting up the environment b. Defining the Tic-Tac-Toe game c. Building the reinforcement learning model d. Training the model e. Testing the model	79
7	<b>Data Loading, Storage and File Formats</b>  <b>Problem Statement:</b> Analyzing Sales Data from Multiple File Formats  <b>Dataset:</b> Sales data in multiple file formats (e.g., CSV, Excel, JSON)  <b>Description:</b> The goal is to load and analyze sales data from different file formats, including CSV, Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset. <b>Tasks to Perform:</b> Obtain sales data files in various formats, such as CSV, Excel, and JSON. 1. Load the sales data from each file format into the appropriate data structures or dataframes. 2. Explore the structure and content of the loaded data, identifying any inconsistencies, missing values, or data quality issues.	92

	<p>3. Perform data cleaning operations, such as handling missing values, removing duplicates, or correcting inconsistencies.</p> <p>4. Convert the data into a unified format, such as a common dataframe or data structure, to enable seamless analysis.</p> <p>5. Perform data transformation tasks, such as merging multiple datasets, splitting columns, or deriving new variables.</p> <p>6. Analyze the sales data by performing descriptive statistics, aggregating data by specific variables, or calculating metrics such as total sales, average order value, or product category distribution.</p> <p>7. Create visualizations, such as bar plots, pie charts, or box plots, to represent the sales data and gain insights into sales trends, customer behavior, or product performance.</p>	
8	<p><b>Interacting with Web APIs</b></p> <p><b>Problem Statement:</b> Analyzing Weather Data from OpenWeatherMap API</p> <p><b>Dataset:</b> Weather data retrieved from OpenWeatherMap API</p> <p><b>Description:</b> The goal is to interact with the OpenWeatherMap API to retrieve weather data for a specific location and perform data modeling and visualization to analyze weather patterns over time.</p> <p><b>Tasks to Perform:</b></p> <ol style="list-style-type: none"> <li>1. Register and obtain API key from OpenWeatherMap.</li> <li>2. Interact with the OpenWeatherMap API using the API key to retrieve weather data for a specific location.</li> <li>3. Extract relevant weather attributes such as temperature, humidity, wind speed, and precipitation from the API response.</li> </ol>	99
	<p>4. Clean and preprocess the retrieved data, handling missing values or inconsistent formats.</p> <p>5. Perform data modeling to analyze weather patterns, such as calculating average temperature, maximum/minimum values, or trends over time.</p> <p>6. Visualize the weather data using appropriate plots, such as line charts, bar plots, or scatter plots, to represent temperature changes, precipitation levels, or wind speed variations.</p> <p>7. Apply data aggregation techniques to summarize weather statistics by specific time periods (e.g., daily, monthly, seasonal).</p> <p>8. Incorporate geographical information, if available, to create maps or geospatial visualizations representing weather patterns across different locations.</p> <p>9. Explore and visualize relationships between weather attributes, such as temperature and humidity, using correlation plots or heatmaps.</p>	

9	<p><b>Data Cleaning and Preparation</b></p> <p><b>Problem Statement:</b> Analyzing Customer Churn in a Telecommunications Company</p> <p><b>Dataset:</b> "Telecom_Customer_Churn.csv"</p> <p><b>Description:</b> The dataset contains information about customers of a telecommunications company and whether they have churned (i.e., discontinued their services). The dataset includes various attributes of the customers, such as their demographics, usage patterns, and account information. The goal is to perform data cleaning and preparation to gain insights into the factors that contribute to customer churn.</p> <p><b>Tasks to Perform:</b></p> <ol style="list-style-type: none"> <li>1. Import the "Telecom_Customer_Churn.csv" dataset.</li> <li>2. Explore the dataset to understand its structure and content.</li> <li>3. Handle missing values in the dataset, deciding on an appropriate strategy.</li> <li>4. Remove any duplicate records from the dataset.</li> <li>5. Check for inconsistent data, such as inconsistent formatting or spelling variations, and standardize it.</li> <li>6. Convert columns to the correct data types as needed.</li> <li>7. Identify and handle outliers in the data.</li> <li>8. Perform feature engineering, creating new features that may be relevant to predicting customer churn.</li> <li>9. Normalize or scale the data if necessary.</li> <li>10. Split the dataset into training and testing sets for further analysis.</li> <li>11. Export the cleaned dataset for future analysis or modeling.</li> </ol>	107
10	<p><b>Data Wrangling</b></p> <p><b>Problem Statement:</b> Data Wrangling on Real Estate Market</p> <p><b>Dataset:</b> "RealEstate_Prices.csv"</p>	116

	<p><b>Description:</b> The dataset contains information about housing prices in a specific real estate market. It includes various attributes such as property characteristics, location, sale prices, and other relevant features. The goal is to perform data wrangling to gain insights into the factors influencing housing prices and prepare the dataset for further analysis or modeling.</p> <p><b>Tasks to Perform:</b></p> <ol style="list-style-type: none"> <li>1. Import the "RealEstate_Prices.csv" dataset. Clean column names by removing spaces, special characters, or renaming them for clarity.</li> <li>2. Handle missing values in the dataset, deciding on an appropriate strategy (e.g., imputation or removal).</li> <li>3. Perform data merging if additional datasets with relevant information are available (e.g., neighborhood demographics or nearby amenities).</li> <li>4. Filter and subset the data based on specific criteria, such as a particular time period, property type, or location.</li> <li>5. Handle categorical variables by encoding them appropriately (e.g., one-hot encoding or label encoding) for further analysis.</li> <li>6. Aggregate the data to calculate summary statistics or derived metrics such as average sale prices by neighborhood or property type.</li> <li>7. Identify and handle outliers or extreme values in the data that may affect the analysis or modeling process</li> </ol>	
11	<p><b>Data Visualization using matplotlib</b></p> <p><b>Problem Statement:</b> Analyzing Air Quality Index (AQI) Trends in a City</p> <p><b>Dataset:</b> "City_Air_Quality.csv"</p> <p><b>Description:</b> The dataset contains information about air quality measurements in a specific city over a period of time. It includes attributes such as date, time, pollutant levels (e.g., PM2.5, PM10, CO), and the Air Quality Index (AQI) values. The goal is to use the matplotlib library to create visualizations that effectively represent the AQI trends and patterns for different pollutants in the city.</p> <p><b>Tasks to Perform:</b></p> <ol style="list-style-type: none"> <li>1. Import the "City_Air_Quality.csv" dataset.</li> <li>2. Explore the dataset to understand its structure and content.</li> <li>3. Identify the relevant variables for visualizing AQI trends, such as date, pollutant levels, and AQI values.</li> <li>4. Create line plots or time series plots to visualize the overall AQI trend over time.</li> <li>5. Plot individual pollutant levels (e.g., PM2.5, PM10, CO) on separate line plots to visualize their trends over time.</li> <li>6. Use bar plots or stacked bar plots to compare the AQI values across different dates or</li> </ol>	134

	<p>time periods.</p> <p>7. Create box plots or violin plots to analyze the distribution of AQI values for different pollutant categories.</p> <p>8. Use scatter plots or bubble charts to explore the relationship between AQI values and pollutant levels.</p> <p>9. Customize the visualizations by adding labels, titles, legends, and appropriate color Schemes</p>	
12	<p><b>Data Aggregation</b></p> <p><b>Problem Statement:</b> Analyzing Sales Performance by Region in a Retail Company</p> <p><b>Dataset:</b> "Retail_Sales_Data.csv"</p> <p><b>Description:</b> The dataset contains information about sales transactions in a retail company. It includes attributes such as transaction date, product category, quantity sold, and sales amount. The goal is to perform data aggregation to analyze the sales performance by region and identify the top-performing regions.</p> <p><b>Tasks to Perform:</b></p> <ol style="list-style-type: none"> <li>1. Import the "Retail_Sales_Data.csv" dataset.</li> <li>2. Explore the dataset to understand its structure and content.</li> <li>3. Identify the relevant variables for aggregating sales data, such as region, sales amount, and product category.</li> <li>4. Group the sales data by region and calculate the total sales amount for each region.</li> <li>5. Create bar plots or pie charts to visualize the sales distribution by region.</li> <li>6. Identify the top-performing regions based on the highest sales amount.</li> <li>7. Group the sales data by region and product category to calculate the total sales amount for each combination.</li> <li>8. Create stacked bar plots or grouped bar plots to compare the sales amounts across different regions and product categories.</li> </ol>	148
13	<p><b>Part III: Mini Project(Mandatory Assignments)</b></p> <p>Mini Project (Mandatory- Group Activity)</p> <p>It is recommended that group of 3 to 5 students should undergo a mini project (considering the Machine Learning and Data modeling and Visualizing concepts) as content beyond syllabus. Some of the problem statements are mentioned below:</p> <ol style="list-style-type: none"> <li>1. Development of a happiness index for schools (including mental health and well-being parameters, among others) with self-assessment facilities.</li> <li>2. Automated Animal Identification and Detection of Species</li> <li>3. Sentimental analysis on Govt. Released Policies</li> <li>4. Identification of Flood Prone Roads</li> <li>5. Identification of Missing Bridges which would increase the connectivity between regions</li> </ol> <p>Note: Instructor can also assign similar problem statements</p> <p>References:</p> <p>For Dataset <a href="https://data.gov.in/">https://data.gov.in/</a></p> <p>For Problem statements: <a href="https://sih.gov.in/sih2022PS">https://sih.gov.in/sih2022PS</a></p>	156

<b>Lab Assignment No.</b>	1A
<b>Title</b>	To use PCA Algorithm for dimensionality reduction. You have a dataset that includes measurements for different variables on wine (alcohol, ash, magnesium, and so on). Apply PCA algorithm & transform this data so that most variations in the measurements of the variables are captured by a small number of principal components so that it is easier to distinguish between red and white wine by inspecting these principal components
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 1 A      (Group A)

➤ **Aim:** To use PCA Algorithm for dimensionality reduction. You have a dataset that includes measurements for different variables on wine (alcohol, ash, magnesium, and so on). Apply PCA algorithm & transform this data so that most variations in the measurements of the variables are captured by a small number of principal components so that it is easier to distinguish between red and white wine by inspecting these principal components

➤ **Outcome:** At end of this experiment, student will be able understand the scheduler, and how its behaviour influences the performance of the system

➤ **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

➤ **Software Requirement:**

Jupyter Notebook/Ubuntu

➤ **Theory:**

### Principal Component Analysis (PCA)

PCA is an unsupervised machine learning algorithm. PCA is mainly used for dimensionality reduction in a dataset consisting of many variables that are highly correlated or lightly correlated with each other while retaining the variation present in the dataset up to a maximum extent.

It is also a great tool for exploratory data analysis for making predictive models.

PCA performs a linear transformation on the data so that most of the variance or information in your high-dimensional dataset is captured by the first few principal components. The first principal component will capture the most variance, followed by the second principal component, and so on.

Each principal component is a linear combination of the original variables. Because all the principal components are orthogonal to each other, there is no redundant information. So, the total variance in the data is defined as the sum of the variances of the individual component. So decide the total number of principal components according to cumulative variance „„explained““ by them.

### **Implementation:**

```
import pandas as pd
from sklearn.decomposition import PCA
```

```

from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

df = pd.read_csv("C:/Users/HP/Dropbox/PC/Downloads/Wine.csv")

df.keys()

```

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

```
df.head(5)
```

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	\
0	14.23	1.71	2.43	15.6	127	2.80	
1	13.20	1.78	2.14	11.2	100	2.65	
2	13.16	2.36	2.67	18.6	101	2.80	
3	14.37	1.95	2.50	16.8	113	3.85	
4	13.24	2.59	2.87	21.0	118	2.80	

	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	Hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	

	OD280	Proline	Customer_Segment
0	3.92	1065	1
1	3.40	1050	1
2	3.17	1185	1
3	3.45	1480	1
4	2.93	735	1

```
df.Customer_Segment.unique()
```

```
array([1, 2, 3], dtype=int64)
```

```
print(df.isnull().sum())      #checking is null
```

	Alcohol	0
Alcohol	0	
Malic_Acid	0	
Ash	0	
Ash_Alcanity	0	
Magnesium	0	
Total_Phenols	0	
Flavanoids	0	
Nonflavanoid_Phenols	0	
Proanthocyanins	0	
Color_Intensity	0	
Hue	0	
OD280	0	

```
Proline      0
Customer_Segment      0
dtype: int64
```

```
X = df.drop('Customer_Segment', axis=1) # Features
y = df['Customer_Segment'] # Target variable
```

```
for col in X.columns:
    sc = StandardScaler() #Standardize features by removing the mean and scaling to
    unit variance.z = (x - u) / s mean=0, Stddeviation=1
    X[col] = sc.fit_transform(X[[col]]) #Fit to data, then transform it. Compute the mean and
    std to be used for later scaling.
```

```
X.head(5)
```

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	\
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	

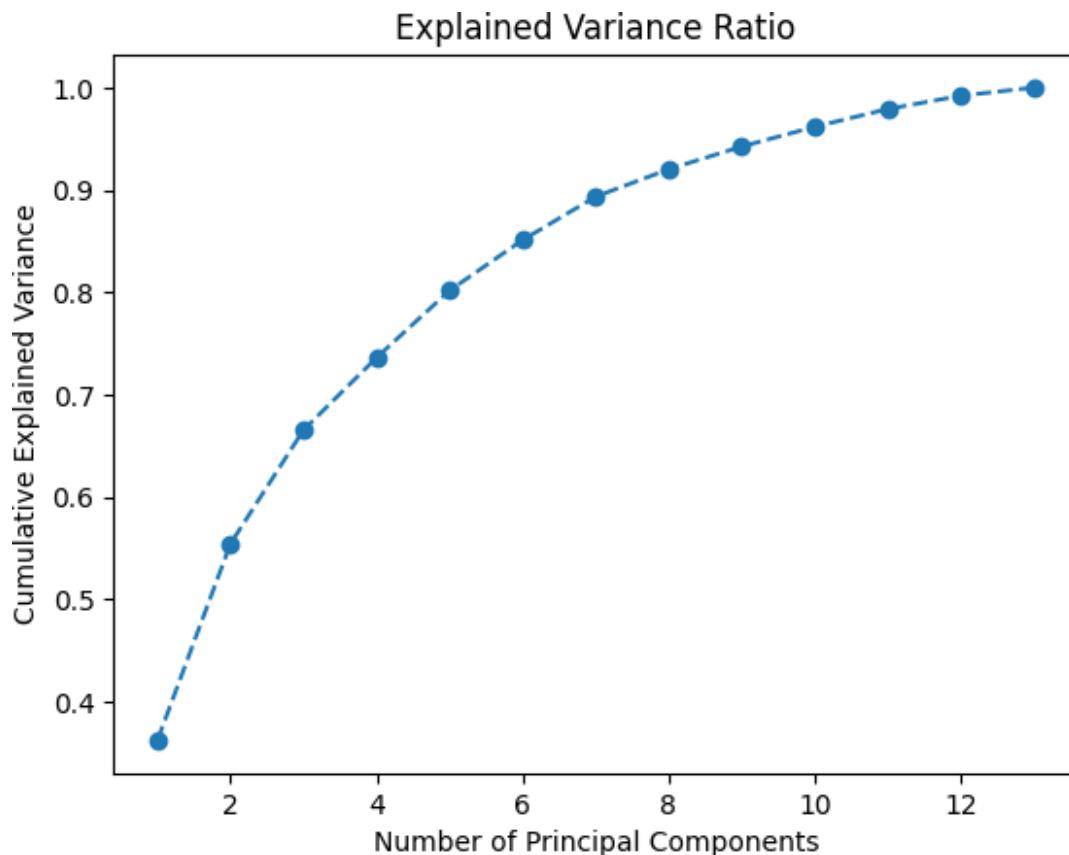
	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	\
0	1.034819	-0.659563	1.224884	0.251717	
1	0.733629	-0.820719	-0.544721	-0.293321	
2	1.215533	-0.498407	2.135968	0.269020	
3	1.466525	-0.981875	1.032155	1.186068	
4	0.663351	0.226796	0.401404	-0.319276	

	Hue	OD280	Proline
0	0.362177	1.847920	1.013009
1	0.406051	1.113449	0.965242
2	0.318304	0.788587	1.395148
3	-0.427544	1.184071	2.334574
4	0.362177	0.449601	-0.037874

```
pca = PCA()
X_pca = pca.fit_transform(X)
```

```
explained_variance_ratio = pca.explained_variance_ratio_
```

```
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio.cumsum(), marker='o',
linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance Ratio')
plt.show()
```



```
n_components = 12 # Choose the desired number of principal components you want to reduce a  
dimension to
```

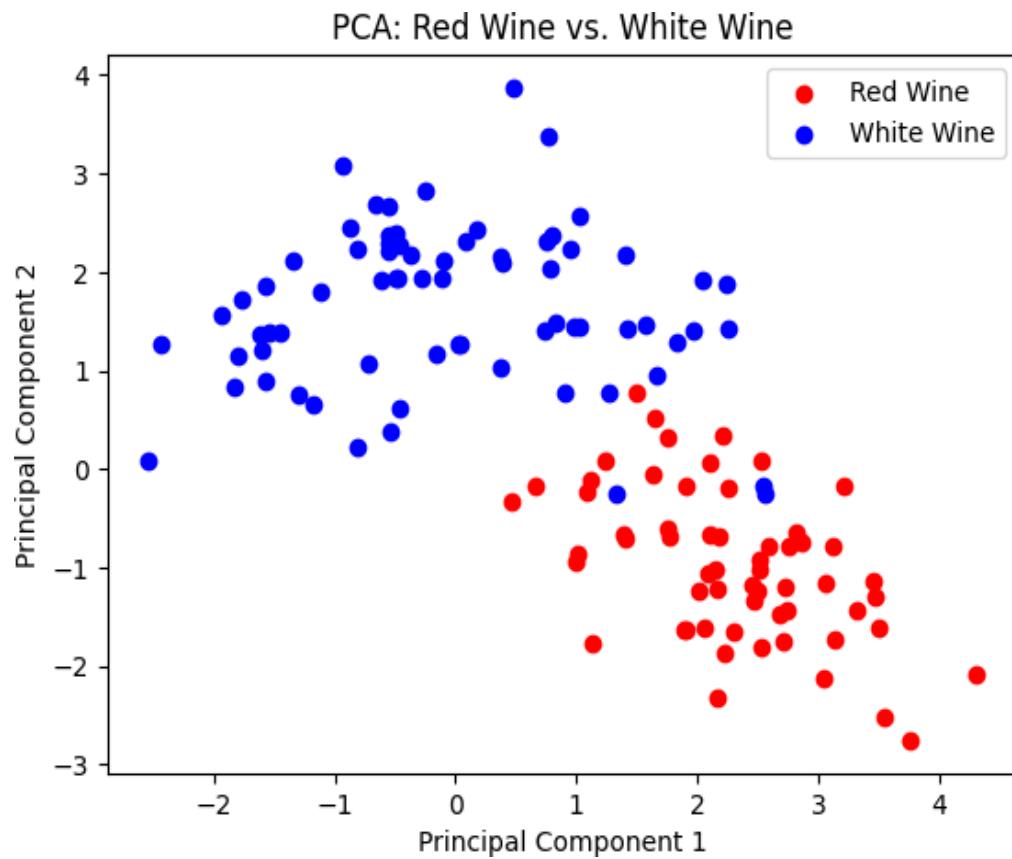
```
pca = PCA(n_components=n_components)  
X_pca = pca.fit_transform(X)
```

```
X_pca.shape
```

```
X.shape
```

```
red_indices = y[y == 1].index  
white_indices = y[y == 2].index
```

```
plt.scatter(X_pca[red_indices, 0], X_pca[red_indices, 1], c='red', label='Red Wine')  
plt.scatter(X_pca[white_indices, 0], X_pca[white_indices, 1], c='blue', label='White Wine')  
plt.xlabel('Principal Component 1')  
plt.ylabel('Principal Component 2')  
plt.legend()  
plt.title('PCA: Red Wine vs. White Wine')  
plt.show()
```



#Conclusion: Here we have reduce the dimention now we can able to apply any algorithm like classification, Regression etc.

<b>Lab Assignment No.</b>	1B
<b>Title</b>	Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link: <a href="https://www.kaggle.com/datasets/uciml/iris">https://www.kaggle.com/datasets/uciml/iris</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 1 B

- **Aim:** Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link:<https://www.kaggle.com/datasets/uciml/iris>

➤ **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

➤ **Software Requirement:**

Jupyter Notebook/Ubuntu

➤ **Theory:**

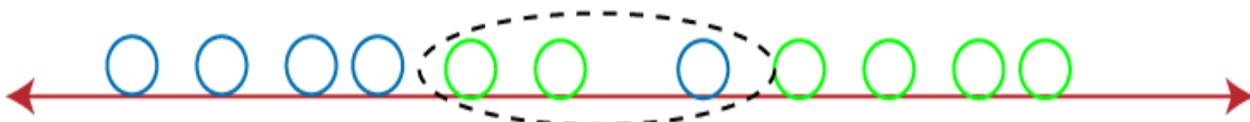
*Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).*

This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs. In this topic, "Linear Discriminant Analysis (LDA) in machine learning", we will discuss the LDA algorithm for classification predictive modeling problems, limitation of logistic regression, representation of linear Discriminant analysis model, how to make a prediction using LDA, how to prepare data for LDA, extensions to LDA and much more. So, let's start with a quick introduction to Linear Discriminant Analysis (LDA) in machine learning.

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

*Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning.* It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.

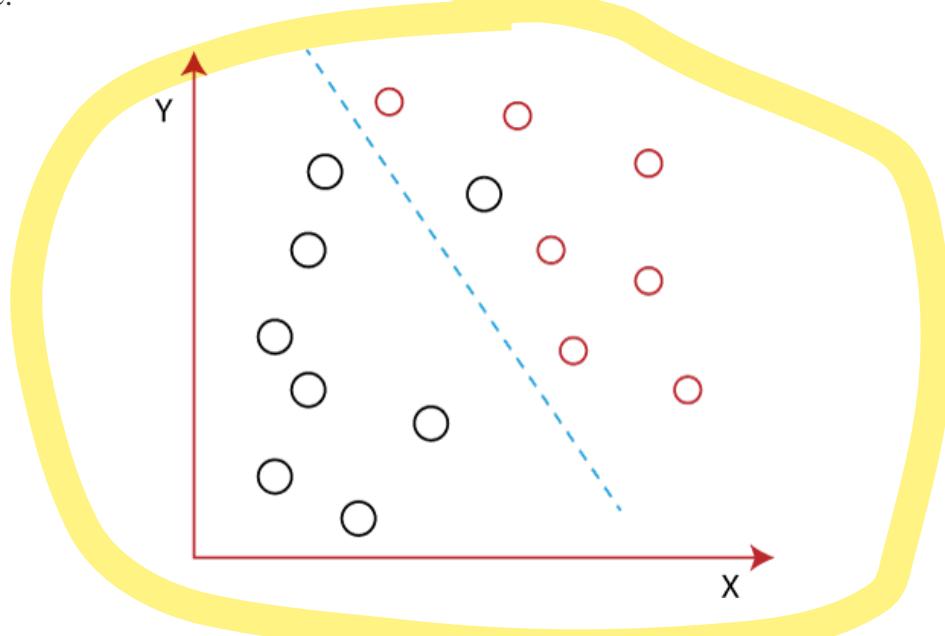


### Overlapping

To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

#### Example:

Let's assume we have to classify two different classes having two sets of data points in a 2-dimensional plane as shown below image:



However, it is impossible to draw a straight line in a 2-d plane that can separate these data points efficiently but using linear Discriminant analysis; we can dimensionally reduce the 2-D plane into the 1-D plane. Using this technique, we can also maximize the separability between multiple classes.

### Implementation:

```
import pandas as pd
```

Reference Link: <https://medium.com/@betulmesci/dimensionality-reduction-with-principal-component-analysis-and-linear-discriminant-analysis-on-iris-dc1731c07fad>

```
df = pd.read_csv("Iris.csv")
```

```
print(df)
```

```

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \
0   1       5.1      3.5       1.4      0.2
1   2       4.9      3.0       1.4      0.2
2   3       4.7      3.2       1.3      0.2
3   4       4.6      3.1       1.5      0.2
4   5       5.0      3.6       1.4      0.2
..  ...
145 146     ...     6.7     ...     5.2     2.3
146 147     ...     6.3     2.5     5.0     1.9
147 148     ...     6.5     3.0     5.2     2.0
148 149     ...     6.2     3.4     5.4     2.3
149 150     ...     5.9     3.0     5.1     1.8

```

	Species				
0	Iris-setosa				
1	Iris-setosa				
2	Iris-setosa				
3	Iris-setosa				
4	Iris-setosa				
..	...				
145	Iris-virginica				
146	Iris-virginica				
147	Iris-virginica				
148	Iris-virginica				
149	Iris-virginica				

[150 rows x 6 columns]

df.Species.unique()

array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

```

X = df.drop(['Id','Species'],axis=1)
y = df['Species']

```

from sklearn.preprocessing import StandardScaler

```

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

```

# Create an instance of LDA
lda = LinearDiscriminantAnalysis(n_components=2)

```

```

# Apply LDA on the scaled features
X_lda = lda.fit_transform(X_scaled, y)

```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, random_state=42)

# Train a logistic regression classifier
classifier = LogisticRegression()
classifier.fit(X_train, y_train)

LogisticRegression()

# Suppose you have a new flower with the following measurements:
new_flower = [[6.7,3.0,5.2,2.3]] # Sepal length, sepal width, petal length, petal width

# Scale the new flower using the same scaler used for training
new_flower_scaled = scaler.transform(new_flower)

# Apply LDA on the scaled new flower
new_flower_lda = lda.transform(new_flower_scaled)

# Predict the species of the new flower
predicted_species = classifier.predict(new_flower_lda)

# Map the predicted label to the actual species
species_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
predicted_species_name = species_mapping[predicted_species[0]]

# Print the predicted species
print("Predicted species:", predicted_species_name)

Predicted species: 2

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
warnings.warn
```

<b>Lab Assignment No.</b>	2A
<b>Title</b>	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and ridge, Lasso regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: <a href="https://www.kaggle.com/datasets/yassersh/uber-fares-dataset">https://www.kaggle.com/datasets/yassersh/uber-fares-dataset</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 2 A

- **Aim:** Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and ridge, Lasso regression models. 5. Evaluate the models and compare their respective scores like R<sup>2</sup>, RMSE, etc. Dataset link: <https://www.kaggle.com/datasets/yasserh/uber-fares-dataset>

➤ **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

➤ **Software Requirement:**

Jupyter Notebook/Ubuntu

➤ **Theory:**

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as temperature, age, salary, price, etc.

### Why do we use Regression Analysis?

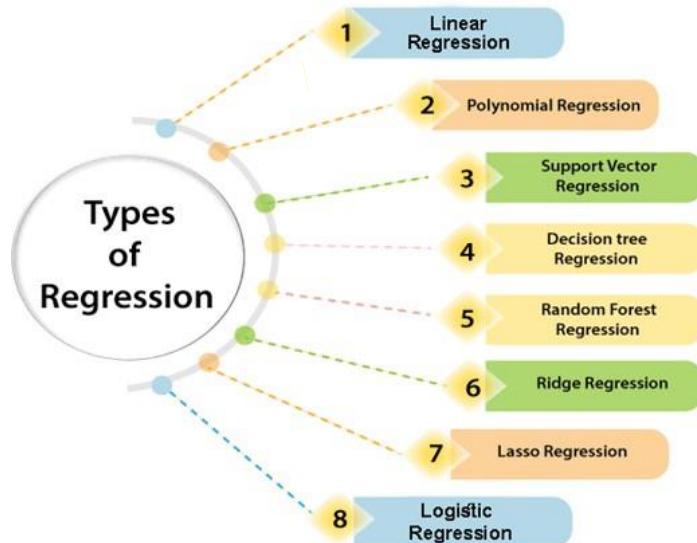
As mentioned above, Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science. Below are some other reasons for using Regression analysis:

- Regression estimates the relationship between the target and the independent variable.
- It is used to find the trends in data.
- It helps to predict real/continuous values.
- By performing the regression, we can confidently determine the most important factor, the least important factor, and how each factor is affecting the other factors.

## Types of Regression

There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables. Here we are discussing some important types of regression which are given below:

- o Linear Regression
- o Logistic Regression
- o Polynomial Regression
- o Support Vector Regression
- o Decision Tree Regression
- o Random Forest Regression
- o Ridge Regression
- o Lasso Regression:



<b>Lab Assignment No.</b>	2B
<b>Title</b>	Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets Dataset link: <a href="https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database">https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 2 B

- **Aim:** Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets Dataset link: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

➤ **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

➤ **Software Requirement:**

Jupyter Notebook/Ubuntu

➤ **Theory:**

Descriptive statistics are brief informational coefficients that summarize a given data set, which can be either a representation of the entire population or a sample of a population. Descriptive statistics are broken down into measures of central tendency and measures of variability (spread). Measures of central tendency include the mean, median, and mode, while measures of variability include standard deviation, variance, minimum and maximum variables, kurtosis, and skewness.

**Types of Descriptive Statistics**

All descriptive statistics are either measures of central tendency or measures of variability, also known as measures of dispersion.

**Central Tendency**

Measures of central tendency focus on the average or middle values of data sets, whereas measures of variability focus on the dispersion of data. These two measures use graphs, tables and general discussions to help people understand the meaning of the analyzed data.

Measures of central tendency describe the center position of a distribution for a data set. A person analyzes the frequency of each data point in the distribution and describes it using the mean, median, or mode, which measures the most common patterns of the analyzed data set.

**Measures of Variability**

Measures of variability (or the measures of spread) aid in analyzing how dispersed the distribution is for a set of data. For example, while the measures of central tendency may give a person the average of a data set, it does not describe how the data is distributed within the set.

So while the average of the data maybe 65 out of 100, there can still be data points at both 1 and 100. Measures of variability help communicate this by describing the shape and spread of the data set. Range, quartiles, absolute deviation, and variance are all examples of measures of variability.

Consider the following data set: 5, 19, 24, 62, 91, 100. The range of that data set is 95, which is calculated by subtracting the lowest number (5) in the data set from the highest (100).

### Distribution

Distribution (or frequency distribution) refers to the quantity of times a data point occurs. Alternatively, it is the measurement of a data point failing to occur. Consider a data set: male, male, female, female, female, other. The distribution of this data can be classified as:

- The number of males in the data set is 2.
- The number of females in the data set is 3.
- The number of individuals identifying as other is 1.
- The number of non-males is 4.

### Univariate vs. Bivariate

In descriptive statistics, univariate data analyzes only one variable. It is used to identify characteristics of a single trait and is not used to analyze any relationships or causations.

For example, imagine a room full of high school students. Say you wanted to gather the average age of the individuals in the room. This univariate data is only dependent on one factor: each person's age. By gathering this one piece of information from each person and dividing by the total number of people, you can determine the average age.

Bivariate data, on the other hand, attempts to link two variables by searching for correlation. Two types of data are collected, and the relationship between the two pieces of information is analyzed together. Because multiple variables are analyzed, this approach may also be referred to as multivariate.

### Descriptive Statistics vs. Inferential Statistics

Descriptive statistics have a different function than inferential statistics, data sets that are used to make decisions or apply characteristics from one data set to another.

Imagine another example where a company sells hot sauce. The company gathers data such as the count of sales, average quantity purchased per transaction, and average sale per day of the week. All of this information is descriptive, as it tells a story of what actually happened in the past. In this case, it is not being used beyond being informational.

Let's say the same company wants to roll out a new hot sauce. It gathers the same sales data above, but it crafts the information to make predictions about what the sales of the new hot sauce will be. The act of using descriptive statistics and applying characteristics to a different data set makes the data set inferential statistics. We are no longer simply summarizing data; we are using it predict what will happen regarding an entirely different body of data (the new hot sauce product).

**Implementation:**

```
import numpy as np
import pandas as pd

df = pd.read_csv("C:/Users/HP/Dropbox/PC/Downloads/diabetes.csv")

df.shape

(768, 9)

df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	
std	3.369578	31.972618	19.355807	15.952218	115.244002	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

**Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation,**

## Skewness and Kurtosis

for column in df.columns:

```
print(f"Column: {column}")
print(f"Frequency:\n{df[column].value_counts()}\n")
print(f"Mean: {df[column].mean()}")
print(f"Median: {df[column].median()}")
print(f"Mode:\n{df[column].mode()}")
print(f"Variance: {df[column].var()}")
print(f"Standard Deviation: {df[column].std()}")
print(f"Skewness: {df[column].skew()}")
print(f"Kurtosis: {df[column].kurt()}")
print(" ----- \n")
```

Column: Pregnancies

Frequency:

1	135
0	111
2	103
3	75
4	68
5	57
6	50
7	45
8	38
9	28
10	24
11	11
13	10
12	9
14	2
15	1
17	1

Name: Pregnancies, dtype: int64

Mean: 3.8450520833333335

Median: 3.0

Mode:

0	1
---	---

Name: Pregnancies, dtype: int64

Variance: 11.35405632062142

Standard Deviation: 3.3695780626988623

Skewness: 0.9016739791518588

Kurtosis: 0.15921977754746486

-----

Column: Glucose

Frequency:

99	17
----	----

```
100 17
111 14
129 14
125 14
...
191 1
177 1
44 1
62 1
190 1
Name: Glucose, Length: 136, dtype: int64
```

Mean: 120.89453125

Median: 117.0

Mode:

```
0 99
1 100
```

Name: Glucose, dtype: int64

Variance: 1022.2483142519557

Standard Deviation: 31.97261819513622

Skewness: 0.17375350179188992

Kurtosis: 0.6407798203735053

---

Column: BloodPressure

Frequency:

```
70 57
74 52
78 45
68 45
72 44
64 43
80 40
76 39
60 37
0 35
62 34
66 30
82 30
88 25
84 23
90 22
86 21
58 21
50 13
56 12
52 11
54 11
```

```
75    8  
92    8  
65    7  
85    6  
94    6  
48    5  
96    4  
44    4  
100   3  
106   3  
98    3  
110   3  
55    2  
108   2  
104   2  
46    2  
30    2  
122   1  
95    1  
102   1  
61    1  
24    1  
38    1  
40    1
```

Name: BloodPressure, dtype: int64

Mean: 69.10546875

Median: 72.0

Mode:

0 70

Name: BloodPressure, dtype: int64

Variance: 374.6472712271838

Standard Deviation: 19.355807170644777

Skewness: -1.8436079833551302

Kurtosis: 5.180156560082496

-----

Column: SkinThickness

Frequency:

0	227	31	19
32	31	19	18
30	27	39	18
27	23	29	17
23	22	40	16
33	20	25	16
28	20	26	16
18	20	22	16

37	16		45	6
41	15		14	6
35	15		44	5
36	14		10	5
15	14		48	4
17	14		47	4
20	13		49	3
24	12		50	3
42	11		8	2
13	11		7	2
21	10		52	2
46	8		54	2
34	8		63	1
12	7		60	1
38	7		56	1
11	6		51	1
43	6		99	1
16	6			

Name: SkinThickness, dtype: int64

Mean: 20.536458333333332

Median: 23.0

Mode:

0 0

Name: SkinThickness, dtype: int64

Variance: 254.47324532811953

Standard Deviation: 15.952217567727677

Skewness: 0.10937249648187608

Kurtosis: -0.520071866153013

-----  
Column: Insulin

Frequency:

0 374

105 11

130 9

140 9

120 8

..

73 1

171 1

255 1

52 1

112 1

Name: Insulin, Length: 186, dtype: int64

Mean: 79.79947916666667

Median: 30.5

Mode:

0 0

Name: Insulin, dtype: int64

Variance: 13281.180077955281

Standard Deviation: 115.24400235133837

Skewness: 2.272250858431574

Kurtosis: 7.2142595543487715

-----

Column: BMI

Frequency:

32.0 13

31.6 12

31.2 12

0.0 11

32.4 10

..

36.7 1

41.8 1

42.6 1

42.8 1

46.3 1

Name: BMI, Length: 248, dtype: int64

Mean: 31.992578124999998

Median: 32.0

Mode:

0 32.0

Name: BMI, dtype: float64

Variance: 62.15998395738257

Standard Deviation: 7.8841603203754405

Skewness: -0.42898158845356543

Kurtosis: 3.290442900816981

-----

Column: DiabetesPedigreeFunction

Frequency:

0.258 6

0.254 6

0.268 5

0.207 5

0.261 5

..

1.353 1

0.655 1

0.092 1

0.926 1

0.171 1

Name: DiabetesPedigreeFunction, Length: 517, dtype: int64

Mean: 0.47187630208333325

Median: 0.3725

Mode:

0 0.254

1 0.258

Name: DiabetesPedigreeFunction, dtype: float64

Variance: 0.10977863787313938

Standard Deviation: 0.33132859501277484

Skewness: 1.919911066307204

Kurtosis: 5.5949535279830584

Column: Age

Frequency:

22	72	51	8
21	63	52	8
25	48	44	8
24	46	58	7
23	38	47	6
28	35	54	6
26	33	49	5
27	32	48	5
29	29	57	5
31	24	53	5
41	22	60	5
30	21	66	4
37	19	63	4
42	18	62	4
33	17	55	4
38	16	67	3
36	16	56	3
32	16	59	3
45	15	65	3
34	14	69	2
46	13	61	2
43	13	72	1
40	13	81	1
39	12	64	1
35	10	70	1
50	8	68	1

Name: Age, dtype: int64

Mean: 33.240885416666664

Median: 29.0

```

Mode:
0 22
Name: Age, dtype: int64
Variance: 138.30304589037365
Standard Deviation: 11.76023154067868
Skewness: 1.1295967011444805
Kurtosis: 0.6431588885398942
-----

```

```

Column: Outcome
Frequency:
0 500
1 268
Name: Outcome, dtype: int64

Mean: 0.3489583333333333
Median: 0.0
Mode:
0 0
Name: Outcome, dtype: int64
Variance: 0.22748261625380098
Standard Deviation: 0.4769513772427971
Skewness: 0.635016643444986
Kurtosis: -1.600929755156027
-----

```

### Bivariate analysis: Linear and logistic regression modeling

from sklearn.linear\_model import LinearRegression, LogisticRegression

```

# Prepare the data
X_linear = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
y_linear = df['Outcome']

# Fit the linear regression model
model_linear = LinearRegression()
model_linear.fit(X_linear, y_linear)

# Print the coefficients
print('Linear Regression Coefficients:')
for feature, coef in zip(X_linear.columns, model_linear.coef_):
    print(f'{feature}: {coef}')

# Make predictions
predictions_linear = model_linear.predict(X_linear)

Linear Regression Coefficients:
Glucose: 0.005932504680360896

```

```
BloodPressure: -0.00227883712542089
SkinThickness: 0.00016697889986787442
Insulin: -0.0002096169514137912
BMI: 0.013310837289280066
DiabetesPedigreeFunction: 0.1376781570786881
Age: 0.005800684345071733
```

*# Prepare the data*

```
X_logistic = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
y_logistic = df['Outcome']
```

*# Fit the logistic regression model*

```
model_logistic = LogisticRegression()
model_logistic.fit(X_logistic, y_logistic)
```

*# Print the coefficients*

```
print('Logistic Regression Coefficients:')
for feature, coef in zip(X_logistic.columns, model_logistic.coef_[0]):
    print(f'{feature}: {coef}')
```

*# Make predictions*

```
predictions_logistic = model_logistic.predict(X_logistic)
```

Logistic Regression Coefficients:

```
Glucose: 0.03454477124790582
BloodPressure: -0.01220824032665116
SkinThickness: 0.0010051963882454211
Insulin: -0.0013499454083243116
BMI: 0.08780751006435426
DiabetesPedigreeFunction: 0.8191678019528903
Age: 0.032699759788267134
```

### Multiple Regression analysis

```
import statsmodels.api as sm
```

*# Split the dataset into the independent variables (X) and the dependent variable (y)*

```
X = df.drop('Outcome', axis=1) # Independent variables
y = df['Outcome'] # Dependent variable
```

*# Add a constant column to the independent variables*

```
X = sm.add_constant(X)
```

*# Fit the multiple regression model*

```
model = sm.OLS(y, X)
results = model.fit()
```

*# Print the regression results*

```
print(results.summary())
```

## OLS Regression Results

---

Dep. Variable: Outcome R-squared: 0.303  
 Model: OLS Adj. R-squared: 0.296  
 Method: Least Squares F-statistic: 41.29  
 Date: Sat, 08 Jul 2023 Prob (F-statistic): 7.36e-55  
 Time: 15:59:17 Log-Likelihood: -381.91  
 No. Observations: 768 AIC: 781.8  
 Df Residuals: 759 BIC: 823.6  
 Df Model: 8  
 Covariance Type: nonrobust

---



---

	coef	std err	t	P> t	[0.025	0.975]
const	-0.8539	0.085	-9.989	0.000	-1.022	-0.686
Pregnancies	0.0206	0.005	4.014	0.000	0.011	0.031
Glucose	0.0059	0.001	11.493	0.000	0.005	0.007
BloodPressure	-0.0023	0.001	-2.873	0.004	-0.004	-0.001
SkinThickness	0.0002	0.001	0.139	0.890	-0.002	0.002
Insulin	-0.0002	0.000	-1.205	0.229	-0.000	0.000
BMI	0.0132	0.002	6.344	0.000	0.009	0.017
DiabetesPedigreeFunction	0.1472	0.045	3.268	0.001	0.059	0.236
Age	0.0026	0.002	1.693	0.091	-0.000	0.006

---

Omnibus:	41.539	Durbin-Watson:	1.982
Prob(Omnibus):	0.000	Jarque-Bera (JB):	31.183
Skew:	0.395	Prob(JB):	1.69e-07
Kurtosis:	2.408	Cond. No.	1.10e+03

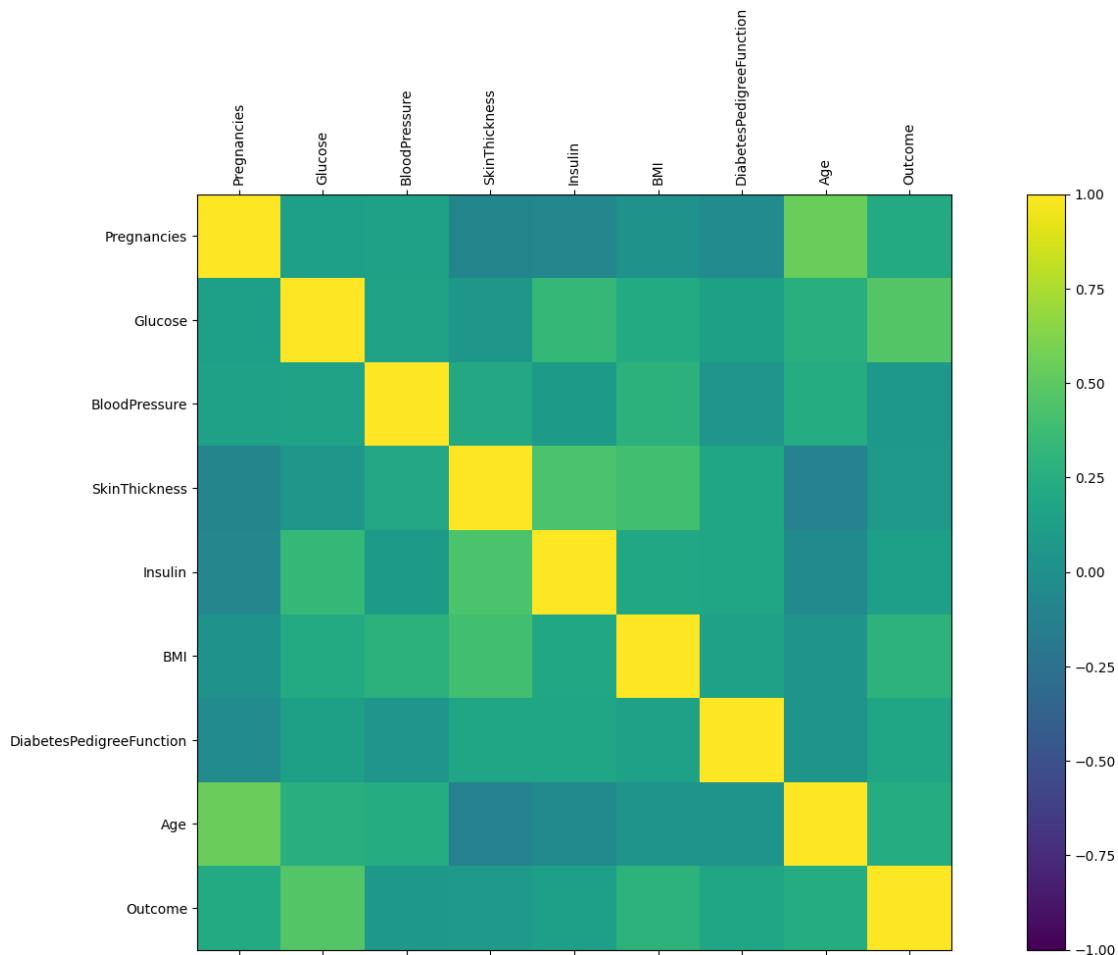
---

Notes:

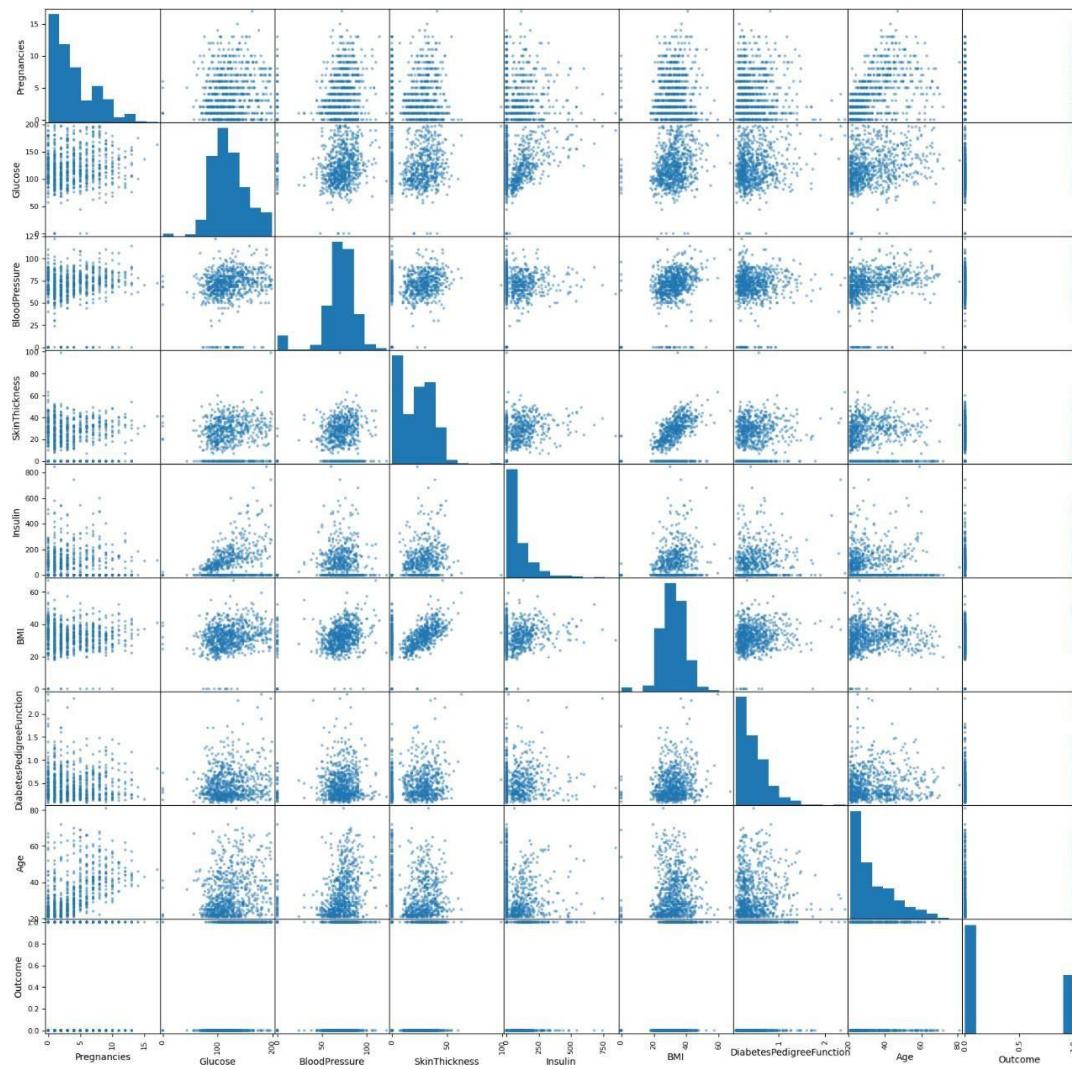
- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.1e+03. This might indicate that there are strong multicollinearity or other numerical problems.

```

fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
names = df.columns
# Rotate x-tick labels by 90 degrees
ax.set_xticklabels(names, rotation=90)
ax.set_yticklabels(names)
pyplot.show()
  
```



```
# Import required package
from pandas.plotting import scatter_matrix
pyplot.rcParams['figure.figsize'] = [20, 20]
# Plotting Scatterplot Matrix
scatter_matrix(df)
pyplot.show()
```



<b>Lab Assignment No.</b>	3A
<b>Title</b>	Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 3 A

- **Aim:** Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).
- **Hardware Requirement:**
  - 6 GB free disk space.
  - 2 GB RAM.
  - 2 GB of RAM, plus additional RAM for virtual machines.
  - 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
  - Virtualization is available with the KVM hypervisor
  - Intel 64 and AMD64 architectures
- **Software Requirement:**  
Jupyter Notebook/Ubuntu
- **Theory:**

### Classification Analysis: Definition

This analysis is a data mining technique used to determine the structure and categories within a given dataset.

Classification analysis is commonly used in machine learning, text analytics, and statistical modelling. Above all, it can help identify patterns or groupings between individual observations, enabling researchers to understand their datasets better and make more accurate predictions.

Classification analysis is used to group or classify objects according to shared characteristics. Moreover, this analysis can be used in many applications, from segmenting customers for marketing campaigns to forecasting stock market trends.

### Classification Analysis Example

- Classifying images

One example of a classification analysis is the use of supervised learning algorithms to classify images. In this case, the algorithm is provided with an image dataset (the training set) that contains labelled images.

The algorithm uses labels to learn how to distinguish between different types of objects in the picture. Once trained, it can then be used to classify new images as belonging to one category or another.

- Customer Segmentation

Another example of classification analysis would be customer segmentation for marketing campaigns. Classification algorithms group customers into segments based on their characteristics and behaviours.

This helps marketers target specific groups with tailored content, offers, and promotions that are more likely to appeal to them.

- Stock Market Prediction

Finally, classification analysis can also be used for stock market prediction. Classification algorithms can identify patterns between past stock prices and other economic indicators, such as interest rates or unemployment figures. By understanding these correlations, analysts can better predict future market trends and make more informed investment decisions.

These are just some examples of how classification analysis can be applied to various scenarios. Unquestionably, classification algorithms can be used to analyse datasets in any domain, from healthcare and finance to agriculture and logistics.

### Classification Analysis Techniques

This analysis is a powerful technique used in data science to analyse and categorise data. Classification techniques are used in many areas, from predicting customer behaviours to finding patterns and trends in large datasets.

This analysis can help businesses make informed decisions about marketing strategies, product development, and more. So, let's delve into the various techniques

#### 1. Supervised Learning

Supervised learning algorithms require labelled data. This means the algorithm is provided with a dataset that has already been categorised or labelled with class labels. The algorithm then uses this label to learn how to distinguish between different class objects in the data. Once trained, it can use its predictive power to classify new datasets.

#### 2. Unsupervised Learning

Unsupervised learning algorithms do not require labelled data. Instead, they use clustering and dimensionality reduction techniques to identify patterns in the dataset without any external guidance. These algorithms help segment customers or identify outlier items in a dataset.

#### 3. Deep Learning

Deep learning is a subset/division of machine learning technologies that use artificial neural networks. These algorithms are capable of learning from large datasets and making complex decisions. Deep learning can be used for tasks such as image classification, natural language processing, and predictive analytics.

Classification algorithms can help uncover patterns in the data that could not be detected using traditional methods. By using classification analysis, businesses can gain valuable insights into their customers' behaviours and preferences, helping them make more informed decisions.

### Implementation:

#### # Import Libraries

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

### Handwritten Digit Recognition

Use the sklearn.dataset load\_digits() method. It loads the handwritten digits dataset. The returned data is in the form of a Dictionary. The 'data' attribute contains a flattened array of 64 (each digit image is of 8\*8 pixels) elements representing the digits.

The 'target' attribute is the 'class' of Digit (0-9) Each individual digit is represented through a flattened 64 digit array numbers of Greyscale values. There are 1797 samples in total and each class or digit has roughly 180 samples.

```
from sklearn.datasets import load_digits
digits = load_digits(n_class=10)
```

digits

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
 [ 0.,  0.,  0., ..., 10.,  0.,  0.],
```

```

[ 0., 0., 0., ..., 16., 9., 0.],
...,
[ 0., 0., 1., ..., 6., 0., 0.],
[ 0., 0., 2., ..., 12., 0., 0.],  

[ 0., 0., 10., ..., 12., 1., 0.]]),
'target': array([0, 1, 2, ..., 8, 9, 8]),
'frame': None,
'feature_names': ['pixel_0_0',
'pixel_0_1',
'pixel_0_2',
'pixel_0_3',
'pixel_0_4',
'pixel_0_5',
'pixel_0_6',
'pixel_0_7',
'pixel_1_0',
'pixel_1_1',
'pixel_1_2',
'pixel_1_3',
'pixel_1_4',
'pixel_1_5',
'pixel_1_6',
'pixel_1_7',
'pixel_2_0',
'pixel_2_1',
'pixel_2_2',
'pixel_2_3',
'pixel_2_4',
'pixel_2_5',
'pixel_2_6',
'pixel_2_7',
'pixel_3_0',
'pixel_3_1',
'pixel_3_2',
'pixel_3_3',
'pixel_3_4',
'pixel_3_5',
'pixel_3_6',
'pixel_3_7',
'pixel_4_0',  

'pixel_4_1',
'pixel_4_2',
'pixel_4_3',
'pixel_4_4',
'pixel_4_5',
'pixel_4_6',
'pixel_4_7',
'pixel_5_0',
'pixel_5_1',
'pixel_5_2',
'pixel_5_3',
'pixel_5_4',
'pixel_5_5',
'pixel_5_6',
'pixel_5_7',
'pixel_6_0',
'pixel_6_1',
'pixel_6_2',
'pixel_6_3',
'pixel_6_4',
'pixel_6_5',
'pixel_6_6',
'pixel_6_7',
'pixel_7_0',
'pixel_7_1',
'pixel_7_2',
'pixel_7_3',
'pixel_7_4',
'pixel_7_5',
'pixel_7_6',
'pixel_7_7],  

'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[ [ 0., 0., 5., ..., 1., 0., 0.],
[ 0., 0., 13., ..., 15., 5., 0.],
[ 0., 3., 15., ..., 11., 8., 0.],
...,
[ 0., 4., 11., ..., 12., 7., 0.],
[ 0., 2., 14., ..., 12., 0., 0.],  


```

```
[ 0.,  0.,  6., ...,  0.,  0.,  0.]],

[[ 0.,  0.,  0., ...,  5.,  0.,  0.],
 [ 0.,  0.,  0., ...,  9.,  0.,  0.],
 [ 0.,  0.,  3., ...,  6.,  0.,  0.],
 ...,
 [ 0.,  0.,  1., ...,  6.,  0.,  0.],
 [ 0.,  0.,  1., ...,  6.,  0.,  0.],
 [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

[[ 0.,  0.,  0., ..., 12.,  0.,  0.],
 [ 0.,  0.,  3., ..., 14.,  0.,  0.],
 [ 0.,  0.,  8., ..., 16.,  0.,  0.],
 ...,
 [ 0.,  9., 16., ...,  0.,  0.,  0.],
 [ 0.,  3., 13., ..., 11.,  5.,  0.],
 [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

...,

[[ 0.,  0.,  1., ...,  1.,  0.,  0.],
 [ 0.,  0., 13., ...,  2.,  1.,  0.],
 [ 0.,  0., 16., ..., 16.,  5.,  0.],
 ...,
 [ 0.,  0., 16., ..., 15.,  0.,  0.],
 [ 0.,  0., 15., ..., 16.,  0.,  0.],
 [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

[[ 0.,  0.,  2., ...,  0.,  0.,  0.],
 [ 0.,  0., 14., ..., 15.,  1.,  0.],
 [ 0.,  4., 16., ..., 16.,  7.,  0.],
 ...,
 [ 0.,  0.,  0., ..., 16.,  2.,  0.],
 [ 0.,  0.,  4., ..., 16.,  2.,  0.],
 [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

[[ 0.,  0., 10., ...,  1.,  0.,  0.],
 [ 0.,  2., 16., ...,  1.,  0.,  0.],
 [ 0.,  0., 15., ..., 15.,  0.,  0.],
 ...,
 [ 0.,  4., 16., ..., 16.,  6.,  0.],
 [ 0.,  8., 16., ..., 16.,  8.,  0.],
 [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),

'DESCRIPTION': '.. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n-----\n-----\n**Data Set Characteristics:**\n  :Number of Instances: 1797\n  :Number of Attributes: 64\n  :Attribute Information: 8x8 image of integer pixels in the range 0..16.\n  :Missing Attribute Values: None\n  :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n  :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits
```

datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\nThe data set contains images of hand-written digits: 10 classes where each class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 into the test set. 32x32 bitmaps are divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994.\n.. topic:: References\n - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition, MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University.\n - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n Linear dimensionality reduction using relevance weighted LDA. School of Electrical and Electronic Engineering Nanyang Technological University.\n 2005.\n - Claudio Gentile. A New Approximate Maximal Margin Classification Algorithm. NIPS. 2000.\n}

```
digits['data'][0].reshape(8,8)
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

```
digits['data'][0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
       0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
digits['images'][1]
```

```
array([[ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.],
       [ 0.,  0.,  0., 11., 16.,  9.,  0.,  0.],
       [ 0.,  0.,  3., 15., 16.,  6.,  0.,  0.],
       [ 0.,  7., 15., 16., 16.,  2.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  3.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
       [ 0.,  0.,  0., 11., 16., 10.,  0.,  0.]])
```

```
digits['target'][0:9]
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```

digits['target'][0]
0
digits.images[0]
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])

```

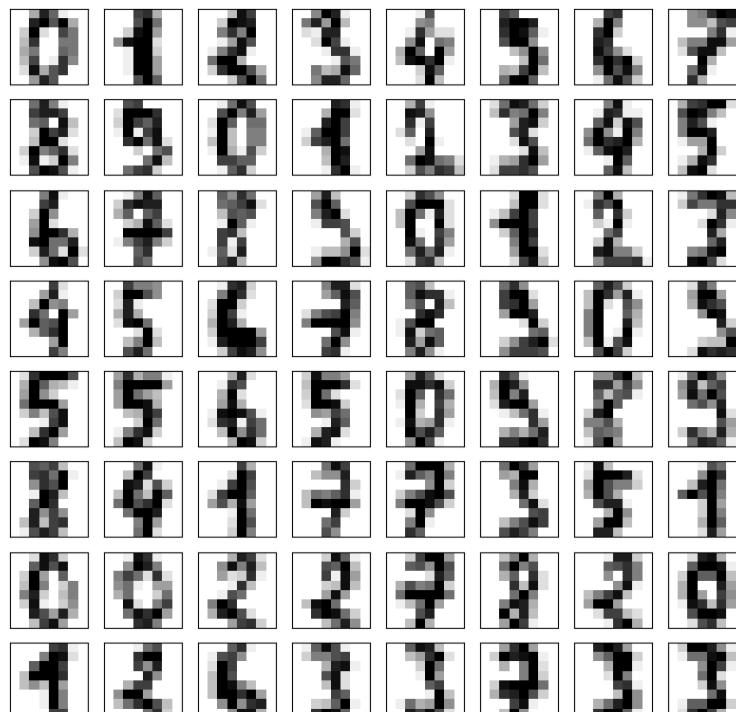
*# Each Digit is represented in digits.images as a matrix of  $8 \times 8 = 64$  pixels. Each of the 64 values represent*

*# a greyscale. The Greyscales are then plotted in the right scale by the imshow method.*

```

fig, ax = plt.subplots(8,8, figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(digits.images[i], cmap='binary')
    axi.set(xticks=[], yticks=[])

```



*# Plotting - Clustering the data points after using Manifold Learning*

```
from sklearn.manifold import Isomap
```

```
iso = Isomap(n_components=2)
```

```

projection = iso.fit_transform(digits.data)      # digits.data - 64 dimensions to 2 dimensions

plt.scatter(projection[:, 0], projection[:, 1], c=digits.target, cmap="viridis")

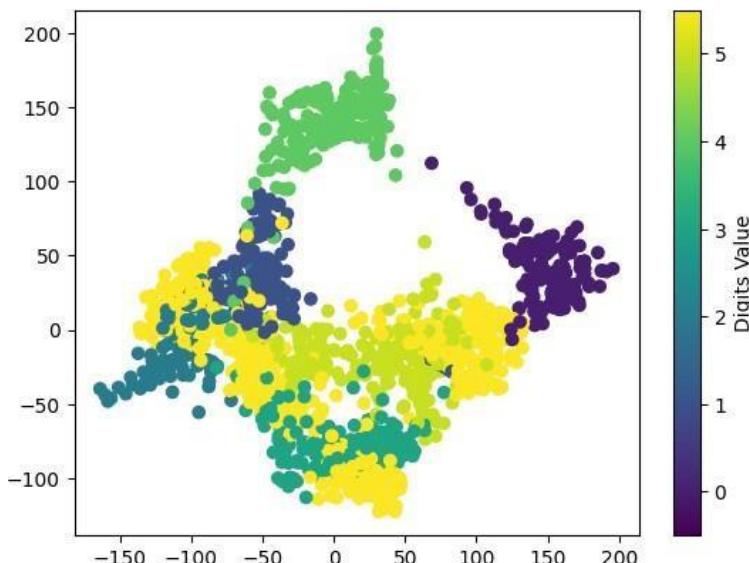
plt.colorbar(ticks=range(10), label='Digits Value')
plt.clim(-0.5, 5.5)

/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_isomap.py:373: UserWarning: The number of
connected components of the neighbors graph is 2 > 1. Completing the graph to fit Isomap might be slow.
Increase the number of neighbors to avoid this issue.

self._fit_transform(X)
/usr/local/lib/python3.10/dist-packages/scipy/sparse/_index.py:103: SparseEfficiencyWarning: Changing
the sparsity structure of a csr_matrix is expensive. lil_matrix is more efficient.

self._set_intXint(row, col, x.flat[0])

```



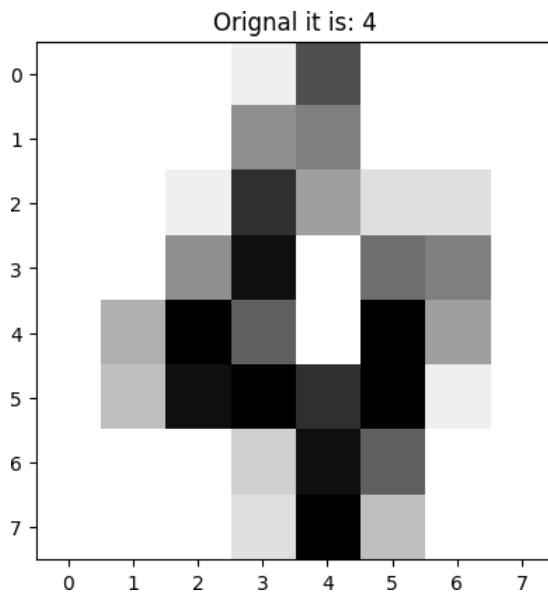
```

print(projection[:, 0][70], projection[:, 1][70])
-56.60683580684862 61.95022367117501

def view_digit(index):
    plt.imshow(digits.images[index] , cmap = plt.cm.gray_r)
    plt.title('Original it is: ' + str(digits.target[index]))
    plt.show()

view_digit(4)

```



### Use the Support Vector Machine Classifier to train the Data

Use part of the data for train and part of the data for test (predicion)

```
main_data = digits['data']
```

```
targets = digits['target']
```

```
from sklearn import svm
```

```
svc = svm.SVC(gamma=0.001 , C = 100)
```

*# GAMMA is a parameter for non linear hyperplanes.*

*# The higher the gamma value it tries to exactly fit the training data set*

*# C is the penalty parameter of the error term.*

*# It controls the trade off between smooth decision boundary and classifying the training points correctly.*

```
svc.fit(main_data[:1500] , targets[:1500])
```

```
predictions = svc.predict(main_data[1501:])
```

```
list(zip(predictions , targets[1501:])))
```

(7, 7),	(8, 8),	(6, 6),
(4, 4),	(4, 4),	(1, 1),
(6, 6),	(3, 3),	(7, 7),
(3, 3),	(1, 1),	(5, 5),
(1, 1),	(4, 4),	(4, 4),
(3, 3),	(0, 0),	(4, 4),
(9, 9),	(5, 5),	(7, 7),
(1, 1),	(3, 3),	(2, 2),
(7, 7),	(6, 6),	(8, 8),
(6, 6),	(9, 9),	(2, 2),

(2, 2),	(0, 0),
(5, 5),	(9, 9),
(7, 7),	(8, 8),
(9, 9),	(9, 9),
(5, 5),	(8, 8),
(4, 4),	(4, 4),
(8, 8),	(1, 1),
(8, 8),	(7, 7),
(4, 4),	(7, 7),
(9, 9),	(3, 3),
(0, 0),	(5, 5),
(8, 8),	(1, 1),
(9, 9),	(0, 0),
(8, 8),	(0, 0),
(0, 0),	(2, 2),
(1, 1),	(2, 2),
(2, 2),	(7, 7),
(3, 3),	(8, 8),
(4, 4),	(2, 2),
(5, 5),	(0, 0),
(6, 6),	(1, 1),
(7, 7),	(2, 2),
(1, 8),	(6, 6),
(9, 9),	(8, 3),
(0, 0),	(3, 3),
(1, 1),	(7, 7),
(2, 2),	(3, 3),
(3, 3),	(3, 3),
(4, 4),	(4, 4),
(5, 5),	(6, 6),
(6, 6),	(6, 6),
(9, 9),	(6, 6),
(0, 0),	(9, 4),
(1, 1),	(9, 9),
(2, 2),	(1, 1),
(3, 3),	(5, 5),
(4, 4),	(0, 0),
(5, 5),	(9, 9),
(6, 6),	(5, 5),
(7, 7),	(2, 2),
(1, 8),	(8, 8),
(9, 9),	(0, 0),
(4, 0),	(1, 1),
(9, 9),	(7, 7),
(5, 5),	(6, 6),
(5, 5),	(3, 3),
(6, 6),	(2, 2),
(5, 5),	(1, 1),

(8, 8),		
(9, 9),	(2, 2),	(3, 3),
(0, 0),	(2, 2),	(9, 9),
(1, 1),	(7, 7),	(1, 1),
(2, 2),	(8, 8),	(7, 7),
(3, 3),	(2, 2),	(6, 6),
(4, 4),	(0, 0),	(8, 8),
(5, 5),	(1, 1),	(4, 4),
(6, 6),	(2, 2),	(5, 3),
(7, 7),	(6, 6),	(1, 1),
(8, 8),	(8, 3),	(4, 4),
(9, 9),	(8, 3),	(0, 0),
(0, 0),	(7, 7),	(5, 5),
(1, 1),	(5, 3),	(3, 3),
(2, 2),	(3, 3),	(6, 6),
(8, 3),	(4, 4),	(9, 9),
(4, 4),	(6, 6),	(6, 6),
(5, 5),	(6, 6),	(1, 1),
(6, 6),	(6, 6),	(7, 7),
(7, 7),	(4, 4),	(5, 5),
(8, 8),	(9, 9),	(4, 4),
(9, 9),	(1, 1),	(4, 4),
(0, 0),	(5, 5),	(7, 7),
(9, 9),	(0, 0),	(2, 2),
(5, 5),	(9, 9),	(8, 8),
(5, 5),	(5, 5),	(2, 2),
(6, 6),	(2, 2),	(2, 2),
(5, 5),	(8, 8),	(5, 5),
(0, 0),	(2, 2),	(7, 7),
(9, 9),	(0, 0),	(9, 9),
(8, 8),	(0, 0),	(5, 5),
(9, 9),	(1, 1),	(4, 4),
(8, 8),	(7, 7),	(8, 8),
(4, 4),	(6, 6),	(8, 8),
(1, 1),	(3, 3),	(4, 4),
(7, 7),	(2, 2),	(9, 9),
(7, 7),	(1, 1),	(0, 0),
(3, 3),	(7, 7),	(8, 8),
(5, 5),	(4, 4),	(9, 9),
(1, 1),	(6, 6),	(8, 8),]
(0, 0),	(3, 3),	

### Create the Confusion Matrix for Performance Evaluation

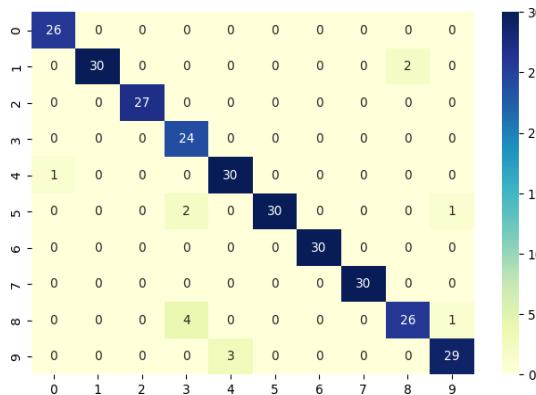
```
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns
```

```
cm = confusion_matrix(predictions, targets[1501:])
```

```
conf_matrix = pd.DataFrame(data = cm)
```

```
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu");
```



cm

```
array([[26, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 30, 0, 0, 0, 0, 0, 0, 2, 0],
       [0, 0, 27, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 24, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 30, 0, 0, 0, 0, 0],
       [0, 0, 0, 2, 0, 30, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 30, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 30, 0, 0, 0],
       [0, 0, 0, 4, 0, 0, 0, 0, 26, 1],
       [0, 0, 0, 3, 0, 0, 0, 0, 0, 29]])
```

### Print the Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(predictions, targets[1501:]))
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	26
1	1.00	0.94	0.97	32
2	1.00	1.00	1.00	27
3	0.80	1.00	0.89	24
4	0.91	0.97	0.94	31
5	1.00	0.91	0.95	33
6	1.00	1.00	1.00	30
7	1.00	1.00	1.00	30
8	0.93	0.84	0.88	31
9	0.94	0.91	0.92	32
Accuracy			0.95	296
macro avg	0.95	0.96	0.95	296
weighted avg	0.96	0.95	0.95	296

<b>Lab Assignment No.</b>	3B
<b>Title</b>	Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 3 B

➤ **Aim:** Implement K-Nearest Neighbours' algorithm on Social network ad dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

➤ **Dataset link:**<https://www.kaggle.com/datasets/rakeshrau/social-network-ads>

➤ **Hardware Requirement:**

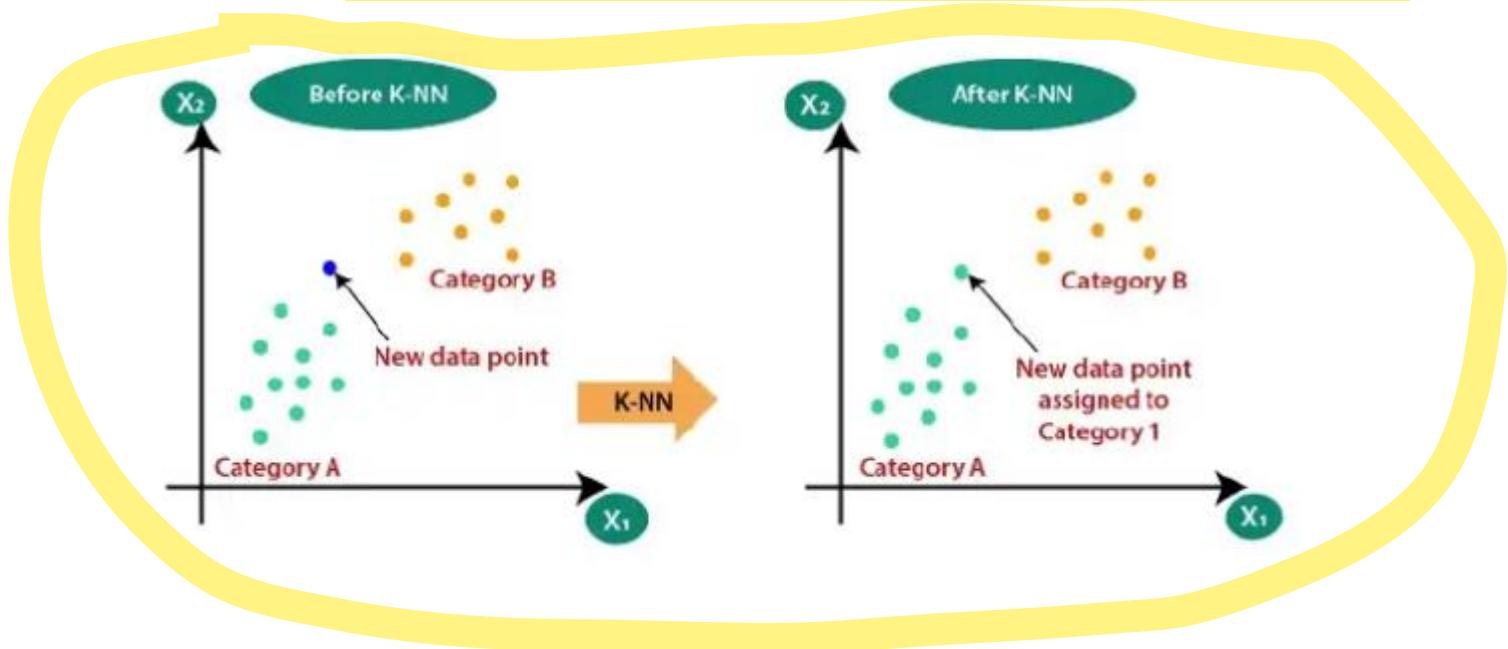
- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

➤ **Software Requirement:**

Jupyter Notebook/Ubuntu

➤ **Theory:**

**K-Nearest Neighbors (KNN) Algorithm:** K-Nearest Neighbors is a supervised machine learning algorithm used for classification and regression tasks. In KNN, an object is classified by a majority vote of its neighbors, with the object being assigned to the class that is most common among its K nearest neighbors (K is a hyperparameter). KNN is a non-parametric and instance-based learning algorithm. Confusion Matrix:



<b>Lab Assignment No.</b>	4A
<b>Title</b>	Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method. Dataset Link: <a href="https://www.kaggle.com/datasets/uciml/iris">https://www.kaggle.com/datasets/uciml/iris</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 4 A

- **Aim:** Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method. Dataset Link: <https://www.kaggle.com/datasets/uciml/iris>

- **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

- **Software Requirement:**

Jupyter Notebook/Ubuntu

- **Theory:**

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by „K“ in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

### Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps –

Step 1 – First, we need to specify the number of clusters, K, need to be generated by this algorithm.

Step 2 – Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.

Step 3 – Now it will compute the cluster centroids.

Step 4 – Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more –

4.1 – First, the sum of squared distance between data points and centroids would be computed.

4.2 – Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

4.3 – At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows Expectation-Maximization approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things –

While working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.

Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids

## Implementation:

Importing the libraries and the data

```
import pandas as pd # Pandas (version : 1.1.5)
import numpy as np # Numpy (version : 1.19.2)
import matplotlib.pyplot as plt # Matplotlib (version : 3.3.2)
from sklearn.cluster import KMeans # Scikit Learn (version : 0.23.2)
import seaborn as sns # Seaborn (version : 0.11.1)
plt.style.use('seaborn')
```

Importing the data from .csv file

First we read the data from the dataset using read\_csv from the pandas library.

```
data = pd.read_csv('data\iris.csv')
```

Viewing the data that we imported to pandas dataframe object

```
data
```

						\
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	3.0	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	
3	4	4.6	3.1	1.5	0.2	
4	5	5.0	3.6	1.4	0.2	
..	...	...	...	...	...	
145	146	6.7	3.0	5.2	2.3	
146	147	6.3	2.5	5.0	1.9	
147	148	6.5	3.0	5.2	2.0	
148	149	6.2	3.4	5.4	2.3	
149	150	5.9	3.0	5.1	1.8	

		Species
0		Iris-setosa
1		Iris-setosa
2		Iris-setosa
3		Iris-setosa
4		Iris-setosa
..	...	
145		Iris-virginica
146		Iris-virginica
147		Iris-virginica
148		Iris-virginica
149		Iris-virginica

[150 rows x 6 columns]

### Viewing and Describing the data

Now we view the Head and Tail of the data using head() and tail() respectively.

data.head()

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
----	---------------	--------------	---------------	--------------	---------

```

1      5.1      3.5      1.4      0.2 Iris-setosa
1  2    4.9      3.0      1.4      0.2 Iris-setosa
2  3    4.7      3.2      1.3      0.2 Iris-setosa
3  4    4.6      3.1      1.5      0.2 Iris-setosa
4  5    5.0      3.6      1.4      0.2 Iris-setosa

```

`data.tail()`

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
145	146	6.7	3.0	5.2	2.3	
146	147	6.3	2.5	5.0	1.9	
147	148	6.5	3.0	5.2	2.0	
148	149	6.2	3.4	5.4	2.3	
149	150	5.9	3.0	5.1	1.8	

Species

```

145 Iris-virginica
146 Iris-virginica
147 Iris-virginica
148 Iris-virginica
149 Iris-virginica

```

Checking the sample size of data - how many samples are there in the dataset using `len()`.

`len(data)`

150

Checking the dimensions/shape of the dataset using `shape`.

`data.shape`

(150, 6)

Viewing Column names of the dataset using `columns`

`data.columns`

```

Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
       dtype='object')

```

for i,col in enumerate(data.columns):

```
print(f'Column number {1+i} is {col}')
```

Column number 1 is Id

Column number 2 is SepalLengthCm

Column number 3 is SepalWidthCm

Column number 4 is PetalLengthCm

Column number 5 is PetalWidthCm

Column number 6 is Species

So, our dataset has 5 columns named:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species.

View datatypes of each column in the dataset using dtype.

```
data.dtypes
```

```
Id           int64
SepalLengthCm  float64
SepalWidthCm   float64
PetalLengthCm  float64
PetalWidthCm   float64
Species        object
dtype: object
```

Gathering Further information about the dataset using info()

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 150 entries, 0 to 149

Data columns (total 6 columns):

#	Column	Non-Null Count	Dtype
0	Id	150 non-null	int64

```

1 SepalLengthCm 150 non-null float64
2 SepalWidthCm 150 non-null float64
3 PetalLengthCm 150 non-null float64
4 PetalWidthCm 150 non-null float64
5 Species 150 non-null object

```

dtypes: float64(4), int64(1), object(1)

memory usage: 7.2+ KB

Describing the data as basic statistics using describe()

data.describe()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Checking the data for inconsistencies and further cleaning the data if needed.

Checking data for missing values using isnull().

data.isnull()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
..	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False

```
148 False False False False False False
149 False False False False False False
```

[150 rows x 6 columns]

Checking summary of missing values

```
data.isnull().sum()
```

```
Id          0
SepalLengthCm  0
SepalWidthCm   0
PetalLengthCm  0
PetalWidthCm   0
Species       0
dtype: int64
```

The 'Id' column has no relevance therefore deleting it would be better.

Deleting 'customer\_id' column using drop().

```
data.drop('Id', axis=1, inplace=True)
```

```
data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

## Modelling

### K - Means Clustering

K-means clustering is a clustering algorithm that aims to partition n observations into k clusters.

Initialisation – K initial “means” (centroids) are generated at random Assignment – K clusters are created by associating each observation with the nearest centroid Update – The centroid of the clusters becomes the new mean, Assignment and Update are repeated iteratively until convergence The end result is that the

sum of squared errors is minimised between points and their respective centroids. We will use KMeans Clustering. At first we will find the optimal clusters based on inertia and using elbow method. The distance between the centroids and the data points should be less.

First we need to check the data for any missing values as it can ruin our model.

```
data.isna().sum()
```

```
SepalLengthCm      0
SepalWidthCm      0
PetalLengthCm     0
PetalWidthCm      0
Species          0
dtype: int64
```

We conclude that we don't have any missing values therefore we can go forward and start the clustering procedure.

We will now view and select the data that we need for clustering.

```
data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Checking the value count of the target column i.e. 'Species' using value\_counts()

```
data['Species'].value_counts()
```

```
Iris-setosa    50
Iris-versicolor 50
```

```
Iris-virginica    50
```

```
Name: Species, dtype: int64
```

Splitting into Training and Target data

Target Data

```
target_data = data.iloc[:,4]
```

```
target_data.head()
```

```
0    Iris-setosa  
1    Iris-setosa  
2    Iris-setosa  
3    Iris-setosa  
4    Iris-setosa
```

```
Name: Species, dtype: object
```

Training data

```
clustering_data = data.iloc[:,[0,1,2,3]]
```

```
clustering_data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Now, we need to visualize the data which we are going to use for the clustering. This will give us a fair idea about the data we're working on.

```
fig, ax = plt.subplots(figsize=(15,7))  
sns.set(font_scale=1.5)  
ax = sns.scatterplot(x=data['SepalLengthCm'],y=data['SepalWidthCm'], s=70, color='#f73434',
```

```

edgecolor='#f73434', linewidth=0.3)

ax.set_ylabel('Sepal Width (in cm)')
ax.set_xlabel('Sepal Length (in cm)')
plt.title('Sepal Length vs Width', fontsize = 20)
plt.show()

```

This gives us a fair Idea and patterns about some of the data.

### Determining No. of Clusters Required

#### The Elbow Method

The Elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes an average score for all clusters. By default, the distortion score is computed, the sum of square distances from each point to its assigned center.

When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. If the line chart looks like an arm, then the “elbow” (the point of inflection on the curve) is the best value of k. The “arm” can be either up or down, but if there is a strong inflection point, it is a good indication that the underlying model fits best at that point.

We use the Elbow Method which uses Within Cluster Sum Of Squares (WCSS) against the the number of clusters (K Value) to figure out the optimal number of clusters value. WCSS measures sum of distances of observations from their cluster centroids which is given by the below formula.

formula

where  $Y_i$  is centroid for observation  $X_i$ . The main goal is to maximize number of clusters and in limiting case each data point becomes its own cluster centroid.

With this simple line of code we get all the inertia value or the within the cluster sum of square.

```

from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):

```

```
km = KMeans(i)
km.fit(clustering_data)
wcss.append(km.inertia_)

np.array(wcss)

array([680.8244      , 152.36870648, 78.94084143, 57.31787321,
       46.53558205, 38.93096305, 34.29998554, 30.21678683,
       28.23999745, 25.95204113])
```

Inertia can be recognized as a measure of how internally coherent clusters are.

Now, we visualize the Elbow Method so that we can determine the number of optimal clusters for our dataset.

```
fig, ax = plt.subplots(figsize=(15,7))
ax = plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.axvline(x=3, ls='--')
plt.ylabel('WCSS')
plt.xlabel('No. of Clusters (k)')
plt.title('The Elbow Method', fontsize = 20)
plt.show()
```

It is clear, that the optimal number of clusters for our data are 3, as the slope of the curve is not steep enough after it. When we observe this curve, we see that last elbow comes at k = 3, it would be difficult to visualize the elbow if we choose the higher range.

## Clustering

Now we will build the model for creating clusters from the dataset. We will use n\_clusters = 3 i.e. 3 clusters as we have determined by the elbow method, which would be optimal for our dataset.

Our data set is for unsupervised learning therefore we will use fit\_predict() Suppose we were working with supervised learning data set we would use fit\_transform()

```
from sklearn.cluster import KMeans
```

```
kms = KMeans(n_clusters=3, init='k-means++')
kms.fit(clustering_data)
```

```
KMeans(n_clusters=3)
```

Now that we have the clusters created, we will enter them into a different column

```
clusters = clustering_data.copy()
clusters['Cluster_Prediction'] = kms.fit_predict(clustering_data)
clusters.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	

	Cluster_Prediction
0	1
1	1
2	1
3	1
4	1

We can also get the centroids of the clusters by the cluster\_centers\_ attribute of KMeans algorithm.

```
kms.cluster_centers_
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006    , 3.418      , 1.464    , 0.244      ],
       [6.85     , 3.07368421, 5.74210526, 2.07105263]])
```

Now we have all the data we need, we just need to plot the data. We will plot the data using scatterplot which will allow us to observe different clusters in different colours.

```
fig, ax = plt.subplots(figsize=(15,7))

plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 0]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 0]['SepalWidthCm'],
            s=70,edgecolor='teal', linewidth=0.3, c='teal', label='Iris-versicolor')

plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 1]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 1]['SepalWidthCm'],
            s=70,edgecolor='lime', linewidth=0.3, c='lime', label='Iris-setosa')

plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 2]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 2]['SepalWidthCm'],
            s=70,edgecolor='magenta', linewidth=0.3, c='magenta', label='Iris-virginica')

plt.scatter(x=kms.cluster_centers_[:, 0], y=kms.cluster_centers_[:, 1], s = 170, c = 'yellow', label = 'Centroids',edgecolor='black', linewidth=0.3)

plt.legend(loc='upper right')

plt.xlim(4,8)

plt.ylim(1.8,4.5)

ax.set_ylabel('Sepal Width (in cm)')

ax.set_xlabel('Sepal Length (in cm)')

plt.title('Clusters', fontsize = 20)

plt.show()
```

<b>Lab Assignment No.</b>	4B
<b>Title</b>	Implement K-Mediod Algorithm on a credit card dataset. Determine the number of clusters using the Silhouette Method. Datasetlink: <a href="https://www.kaggle.com/datasets/arjunbhasin2013/ccdata">https://www.kaggle.com/datasets/arjunbhasin2013/ccdata</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 4 B

### Aim:

To implement the **K-Medoids clustering algorithm** on the Credit Card dataset and determine the optimal number of clusters using the **Silhouette Method**.

### Outcome :

1. Successfully implemented **K-Medoids algorithm** for clustering customer records in the Credit Card dataset.
2. Determine the **optimal number of clusters (k)** using the **Silhouette score**.
3. Derived meaningful insights about customer segmentation based on financial behavior.

### Hardware Requirements:

- **Processor:** Intel i3/i5/i7 or equivalent
- **RAM:** Minimum 4 GB (8 GB recommended)
- **Hard Disk:** 500 MB free space
- **GPU (optional):** For faster computation

### Software Requirements:

- **Operating System:** Windows / Linux / macOS
- **Programming Language:** Python 3.8+
- **Libraries:**
  - numpy
  - pandas
  - scikit-learn
  - scikit-learn-extra (for K-Medoids implementation)
  - matplotlib
  - seaborn

## Theory:

### 1. Clustering

Clustering is an unsupervised machine learning technique used to group data points into clusters such that data points within the same cluster are more similar to each other than to those in other clusters.

### 2. K-Medoids Algorithm

- K-Medoids is a **partitioning clustering algorithm** similar to K-Means but more **robust to noise and outliers**.
- Instead of using the mean (as in K-Means), it chooses **actual data points (medoids)** as the cluster centers.
- Steps:
  1. Initialize **k medoids** randomly.
  2. Assign each data point to the nearest medoid based on a distance metric (e.g., Euclidean distance).
  3. For each cluster, select a data point as the new medoid if it minimizes the **total cost** (sum of distances of points to the medoid).
  4. Repeat until medoids no longer change.

### 3. Silhouette Method

- The **Silhouette Score** measures how well a point fits within its cluster compared to other clusters.
- **Formula:**  $s = \frac{b - a}{\max(a, b)}$   
where:
  - $a$  = average distance of a point to other points in the same cluster.
  - $b$  = average distance of a point to points in the nearest cluster.
- **Score Range:**  $-1 \rightarrow +1$   
  - **+1** → well-clustered
  - **0** → overlapping clusters
  - **-1** → wrong clustering
- The **optimal k** is the one with the highest average Silhouette Score.

**Algorithm: K-Medoids Clustering****Input:**

- Dataset DDD with nnn objects
- Number of clusters kkk

**Output:**

- Set of kkk clusters with their medoids

**Steps:****1. Initialization:**

Select kkk objects randomly from the dataset DDD as the initial medoids.

**2. Assignment Step:**

For each data point  $x_i \in D$ , assign it to the nearest medoid using a distance metric (e.g., Euclidean distance).

**3. Update Step:**

For each cluster:

- Choose the object within the cluster that minimizes the total distance to all other points in the cluster.
- Replace the current medoid with this new object.

**4. Repeat** steps 2 and 3 until the medoids do not change (or a maximum number of iterations is reached).**5. Output** the final clusters and medoids.

<b>Lab Assignment No.</b>	5A
<b>Title</b>	Implement Random Forest Classifier model to predict the safety of the car. Dataset link: <a href="https://www.kaggle.com/datasets/elikplim/car-evaluation-dataset">https://www.kaggle.com/datasets/elikplim/car-evaluation-dataset</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 5 A

### Aim:

To build a **Random Forest Classification model** that predicts the “safety” (or the class / label) of a car based on its attributes (e.g. price, maintenance, doors, persons, lug\_boot, safety).

### Outcome:

- A trained Random Forest model with good predictive accuracy.
- Performance metrics (accuracy, precision, recall, F1-score, confusion matrix) on test data.
- Feature importance analysis (which car attributes are most predictive of safety).
- Insights from misclassifications and possible improvements.

### Hardware Requirements:

- Processor: Any modern CPU (Intel / AMD)
- RAM: 4 GB minimum (8 GB recommended)
- Disk: Enough to store dataset + libraries (a few hundred MB)
- (Optional) GPU: Not needed (RF is CPU-based, unless you use GPU-accelerated libraries, but basic is fine on CPU)

### Software Requirements:

- OS: Windows / Linux / macOS
- Python 3.7+
- Libraries:
  - numpy
  - pandas
  - scikit-learn
  - matplotlib / seaborn (for plotting)
  - (Optional) joblib or pickle (for model saving)

### Theory:

#### 5.1 Random Forest Classifier

- Random Forest is an **ensemble** of decision trees.

- Each tree is trained on a bootstrap sample of the data (sampling with replacement).
- At each node split, only a random subset of features is considered for the best split (this introduces diversity).
- Final prediction is by majority voting among all trees (for classification).
- It tends to reduce overfitting (compared to a single decision tree) and improves generalization.

### Advantages:

- Robust to overfitting (often)
- Works well with categorical and numerical features
- Can measure feature importance
- Handles missing data (to some extent) and scales well

### Hyperparameters:

- n\_estimators (number of trees)
- max\_depth (maximum depth of each tree)
- min\_samples\_split, min\_samples\_leaf
- max\_features (number (or fraction) of features to consider for each split)
- bootstrap (whether bootstrap sampling is used)

### Algorithm (High Level)

**Input:** Car evaluation dataset with features and target class

**Output:** A trained Random Forest model and performance on test set

1. **Load data**
2. **Preprocess / Encode** categorical features → transform them to numeric (e.g., via LabelEncoder, OneHotEncoder)
3. **Split** data into training and testing sets

4. **Train** Random Forest classifier on training data
5. **Predict** on test data
6. Compute **performance metrics** (accuracy, precision, recall, F1, confusion matrix)
7. **Analyze** feature importances
8. (Optional) **Hyperparameter tuning** (e.g. via GridSearchCV or RandomizedSearchCV)  
(Optional) **Save** the model

<b>Lab Assignment No.</b>	5B
<b>Title</b>	Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient Tree Boosting (GBM), XGBoost classification on Iris dataset and compare the performance of three models using different evaluation measures. Dataset Link <a href="https://www.kaggle.com/datasets/uciml/iris">https://www.kaggle.com/datasets/uciml/iris</a>
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 5 B

### Aim:

Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient Tree Boosting (GBM), XGBoost classification on Iris dataset and compare the performance of three models using different evaluation measures. Dataset Link <https://www.kaggle.com/datasets/uciml/iris>

### Hardware Requirement:

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

### Software Requirement:

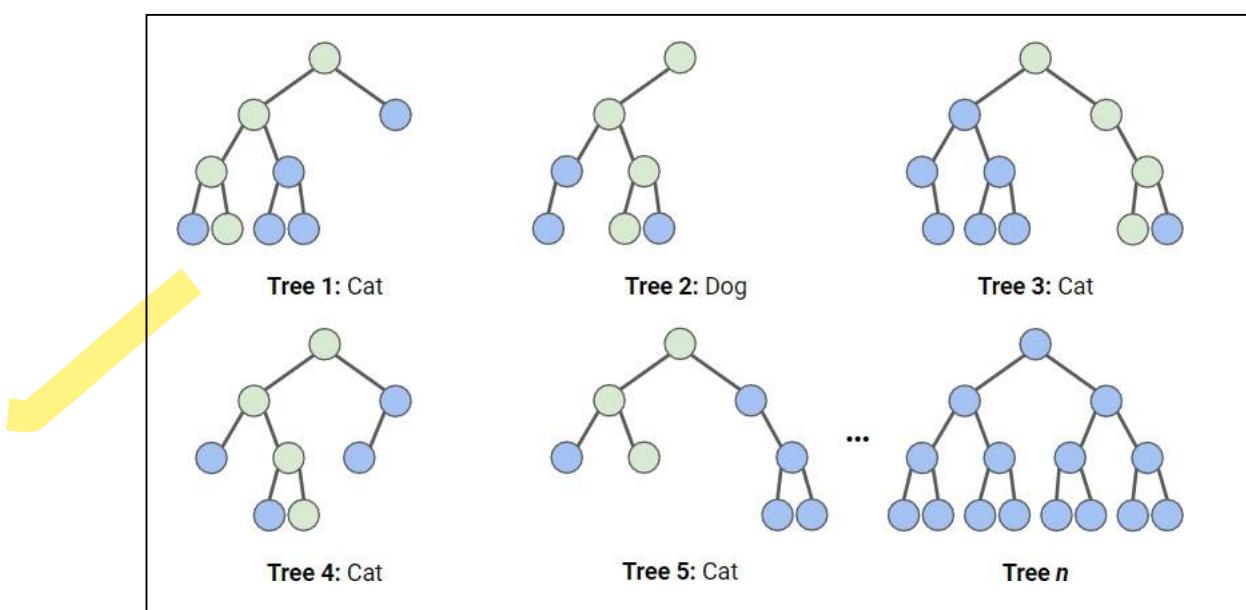
Jupyter Notebook/Ubuntu

### Theory:

Imagine you have a complex problem to solve, and you gather a group of experts from different fields to provide their input. Each expert provides their opinion based on their expertise and experience. Then, the experts would vote to arrive at a final decision.

In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result. (For regression, predictions use an averaging technique instead.)

In the diagram below, we have a random forest with n decision trees, and we've shown the first 5, along with their predictions (either "Dog" or "Cat"). Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction. Looking at the first 5 trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest would count the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.



### Implementation:

```

import pandas as pd
from sklearn.datasets import load_digits
digits = load_digits()

dir(digits)

['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']

%matplotlib inline
import matplotlib.pyplot as plt

```

```

plt.gray()
for i in range(4):
    plt.matshow(digits.images[i])

```

<Figure size 640x480 with 0 Axes>

```

df = pd.DataFrame(digits.data)
df.head()

```

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	\
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

```

      57  58  59   60    61  62  63
0  0.0 6.0 13.0 10.0  0.0 0.0 0.0
1  0.0 0.0 11.0 16.0 10.0 0.0 0.0
2  0.0 0.0  3.0 11.0 16.0 9.0 0.0
3  0.0 7.0 13.0 13.0  9.0 0.0 0.0
4  0.0 0.0  2.0 16.0  4.0 0.0 0.0

```

[5 rows x 64 columns]

```
df['target'] = digits.target
```

```
df[0:12]
```

```

      0   1   2   3   4   5   6   7   8   9 ... 55 56 57 \
0  0.0 0.0 5.0 13.0 9.0 1.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
1  0.0 0.0 0.0 12.0 13.0 5.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
2  0.0 0.0 0.0 4.0 15.0 12.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
3  0.0 0.0 7.0 15.0 13.0 1.0 0.0 0.0 0.0 8.0 ... 0.0 0.0 0.0
4  0.0 0.0 0.0 1.0 11.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
5  0.0 0.0 12.0 10.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
6  0.0 0.0 0.0 12.0 13.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
7  0.0 0.0 7.0 8.0 13.0 16.0 15.0 1.0 0.0 0.0 ... 0.0 0.0 0.0
8  0.0 0.0 9.0 14.0 8.0 1.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
9  0.0 0.0 11.0 12.0 0.0 0.0 0.0 0.0 0.0 2.0 ... 0.0 0.0 0.0
10 0.0 0.0 1.0 9.0 15.0 11.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0
11 0.0 0.0 0.0 14.0 13.0 1.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0

```

	58	59	60	61	62	63	target
0	6.0	13.0	10.0	0.0	0.0	0.0	0
1	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	3.0	11.0	16.0	9.0	0.0	2
3	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	2.0	16.0	4.0	0.0	0.0	4
5	9.0	16.0	16.0	10.0	0.0	0.0	5
6	1.0	9.0	15.0	11.0	3.0	0.0	6
7	13.0	5.0	0.0	0.0	0.0	0.0	7
8	11.0	16.0	15.0	11.0	1.0	0.0	8
9	9.0	12.0	13.0	3.0	0.0	0.0	9
10	1.0	10.0	13.0	3.0	0.0	0.0	0
11	0.0	1.0	13.0	16.0	1.0	0.0	1

[12 rows x 65 columns]

Train and the model and prediction

```
X = df.drop('target',axis='columns')
y = df.target
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

RandomForestClassifier(n_estimators=20)

model.score(X_test, y_test)

0.9805555555555555
```

y\_predicted = model.predict(X\_test)

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

```
array([[32, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 30, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 32, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 37, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 35, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 41, 1, 0, 0, 1],
       [0, 0, 0, 0, 1, 0, 35, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 52, 0, 2],
       [1, 0, 0, 0, 0, 0, 0, 0, 32, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 27]])
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Text(95.7222222222221, 0.5, 'Truth')

<b>Lab Assignment No.</b>	6A
<b>Title</b>	Reinforcement Learning Implement Reinforcement Learning using an example of a maze environment that the agent needs to explore.
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 6 A****Aim:**

To implement Reinforcement Learning (Q-learning) for an agent to explore and solve a grid-maze environment — learn a policy that reaches the goal while avoiding walls and penalties.

**Outcome**

- Agent learns an optimal (or near-optimal) policy to navigate from start to goal.
- Training curves (episode reward vs episodes) showing learning.
- Visualization of learned policy (arrows/paths) and sample episode traces.

**Hardware Requirements**

- CPU: Modern dual-core CPU is sufficient.
- RAM: 2 GB minimum (4 GB recommended).
- Disk: ~100 MB free.

**Software Requirements**

- Python 3.7+
- Libraries:
  - numpy
  - matplotlib
  - seaborn (optional)
  - gym not required for a simple custom maze (we'll implement a tiny env)
  - (Optional) pandas for logging

**Theory**

- **Markov Decision Process (MDP):** states, actions, rewards, transition dynamics, discount factor  $\gamma$ .
- **Q-learning:** off-policy TD control algorithm. Update rule:

$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

where  $\alpha$  = learning rate,  $\gamma$  = discount factor.

- Exploration via  $\epsilon$ -greedy.

### Algorithm (Q-learning) — Pseudocode

1. Initialize  $Q(s,a)$  arbitrarily (e.g., zeros) for all states and actions  $a$ .
2. For episode = 1..N:
  - Initialize state  $s$ .
  - For each step:
    - Choose action  $a$  from  $s$  using  $\epsilon$ -greedy( $Q$ ).
    - Execute  $a$ , observe reward  $r$  and next state  $s'$ .
    - $Q[s,a] \leftarrow Q[s,a] + \alpha * (r + \gamma * \max_a Q[s',a] - Q[s,a])$
    - $s \leftarrow s'$
    - If  $s'$  is terminal: break
    - Decay  $\epsilon$  (optional).
3. Return  $Q$  and derive policy  $\pi(s) = \text{argmax}_a Q[s,a]$ .

### Implementation plan

- Define a small grid maze (0=free, 1=wall, G=goal, S=start).
- Discrete state indices (e.g., row \* ncols + col).
- Actions = [up, right, down, left].
- Reward design:
  - Step cost =  $-1$  (encourage shortest path)
  - Hitting wall =  $-5$  (or remain in same state)
  - Reaching goal =  $+50$  (terminal)

- Train with Q-learning and plot episode rewards & steps.

Visualize policy: arrows on grid or path from start.

<b>Lab Assignment No.</b>	6B
<b>Title</b>	Solve the Taxi problem using reinforcement learning where the agent acts as a taxi driver to pick up a passenger at one location and then drop the passenger off at their destination.
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 6 B****Aim:**

Solve the Taxi problem using RL (Q-learning). The taxi must learn to pick up a passenger from a source cell and drop them at a destination cell in a gridworld.

**Outcome:**

- Agent learns a policy to successfully pick up and drop passengers.
- Evaluation: average reward, success rate, steps to completion over evaluation episodes.
- Visualize training performance and sample run (rendered environment).

**Hardware Requirements:**

- Same as Maze: CPU OK, 4 GB RAM recommended.

**Software Requirements:**

- Python 3.7+
- gym (OpenAI Gym) — pip install gym (for Taxi-v3)
- numpy, matplotlib
- (Optional) stable-baselines3 for advanced agents (requires gymnasium / new APIs)

**Theory:**

- Taxi-v3 is a discrete MDP: states encode taxi position, passenger location, and destination.
- Actions: 6 discrete actions (south, north, east, west, pickup, dropoff).
- Q-learning applies directly due to small finite state/action space.
- Reward scheme: -1 per step, +20 for successful dropoff, -10 for illegal pickup/dropoff.

**Algorithm:**

Same as maze Q-learning; state space is env.observation\_space.n, actions env.action\_space.n. Use  $\epsilon$ -greedy exploration and update Q-table with standard Q-learning rule.

## Implementation :

- Use OpenAI Gym Taxi-v3.
- Initialize Q-table of shape ( $n_{\text{states}} \times n_{\text{actions}}$ ).
- Train for many episodes (e.g., 10k).
- Decay  $\epsilon$ .
- Save and evaluate best Q-table.

Python code (Taxi Q-learning)

```
# Taxi-v3 Q-learning
import gym
import numpy as np
import random
import matplotlib.pyplot as plt

env = gym.make('Taxi-v3') # make sure gym is installed

n_states = env.observation_space.n
n_actions = env.action_space.n

Q = np.zeros((n_states, n_actions))

# hyperparameters
alpha = 0.8
gamma = 0.95
epsilon = 1.0
min_epsilon = 0.01
decay = 0.9995
episodes = 20000
max_steps = 200

rewards = []
success_count = 0

for ep in range(episodes):
    state = env.reset()
    total_reward = 0
    done = False
    for step in range(max_steps):
        if random.random() < epsilon:
            action = env.action_space.sample()
        else:
            action = int(np.argmax(Q[state]))
        next_state, reward, done, info = env.step(action)
        total_reward += reward
        if done:
            break
    rewards.append(total_reward)
    if total_reward == 260:
        success_count += 1
    epsilon = max(min_epsilon, epsilon * decay)

plt.plot(rewards)
plt.xlabel('Episodes')
plt.ylabel('Total Reward')
plt.title('Taxi-v3 Q-learning Performance')
plt.show()
```

```
next_state, reward, done, _ = env.step(action)
Q[state, action] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[state, action])
state = next_state
total_reward += reward
if done:
    if reward == 20: # successful dropoff reward
        success_count += 1
        break
rewards.append(total_reward)
epsilon = max(min_epsilon, epsilon * decay)

# Evaluation
print("Training finished.")
print("Average reward (last 1000):", np.mean(rewards[-1000:]))
print("Successes during training:", success_count)

# Plot rewards
plt.plot(rewards)
plt.title('Episode rewards')
plt.show()

# Play a few episodes with greedy policy
for ep in range(5):
    state = env.reset()
    done = False
    total = 0
    steps = 0
    while not done and steps < 200:
        action = int(np.argmax(Q[state]))
        state, reward, done, _ = env.step(action)
        total += reward
        steps += 1
        env.render()
    print(f'Episode {ep} finished in {steps} steps with reward {total}')
    print("-"*40)
```

<b>Lab Assignment No.</b>	6C
<b>Title</b>	Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks a. Setting up the environment b. Defining the Tic-Tac-Toe game c. Building the reinforcement learning model d. Training the model e. Testing the model
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

## EXPERIMENT NO. 6 C

### Aim:

Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks

- a. Setting up the environment
- b. Defining the Tic-Tac-Toe game
- c. Building the reinforcement learning model
- d. Training the model
- e. Testing the model

### Hardware Requirement:

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

### Software Requirement:

Jupyter Notebook/Ubuntu

### Theory:

In reinforcement learning, developers devise a method of rewarding desired behaviors and punishing negative behaviors. This method assigns positive values to the desired actions to encourage the agent and negative values to undesired behaviors. This programs the agent to seek long-term and maximum overall reward to achieve an optimal solution.

These long-term goals help prevent the agent from stalling on lesser goals. With time, the agent learns to avoid the negative and seek the positive. This learning method has been adopted in artificial intelligence (AI) as a way of directing unsupervised machine learning through rewards and penalties.

Common reinforcement learning algorithms

Rather than referring to a specific algorithm, the field of reinforcement learning is made up of several algorithms that take somewhat different approaches. The differences are mainly due to their strategies for exploring their environments.

- State-action-reward-state-action (SARSA). This reinforcement learning algorithm starts by giving the agent what's known as a *policy*. The policy is essentially a probability that tells it the odds of certain actions resulting in rewards, or beneficial states.
- Q-learning. This approach to reinforcement learning takes the opposite approach. The agent receives no policy, meaning its exploration of its environment is more self-directed.
- Deep Q-Networks. These algorithms utilize neural networks in addition to reinforcement learning techniques. They utilize the self-directed environment exploration of reinforcement learning. Future actions are based on a random sample of past beneficial actions learned by the neural network.

## Implementation:

```

import numpy as np

class TicTacToeEnvironment:
    def __init__(self):
        self.state = [0] * 9
        self.is_terminal = False
    def reset(self):
        self.state = [0] * 9
        self.is_terminal = False

    def get_available_moves(self):
        return [i for i, mark in enumerate(self.state) if mark == 0]

    def make_move(self, move, player_mark):
        self.state[move] = player_mark

    def check_win(self, player_mark):
        winning_states = [
            [0, 1, 2], [3, 4, 5], [6, 7, 8], # rows
            [0, 3, 6], [1, 4, 7], [2, 5, 8], # columns
            [0, 4, 8], [2, 4, 6] # diagonals
        ]
        for state_indices in winning_states:
            if all(self.state[i] == player_mark for i in state_indices):
                self.is_terminal = True
                return True
        return False

```

```
def is_draw(self):
    return 0 not in self.state

class QLearningAgent:
    def __init__(self, learning_rate=0.9, discount_factor=0.9, exploration_rate=0.3):
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.q_table = np.zeros((3**9, 9))

    def get_state_index(self, state):
        state_index = 0
        for i, mark in enumerate(state):
            state_index += (3 ** i) * (mark + 1)
        return state_index

    def choose_action(self, state, available_moves):
        state_index = self.get_state_index(state)
        if np.random.random() < self.exploration_rate:
            return np.random.choice(available_moves)
        else:
            return np.argmax(self.q_table[state_index, available_moves])

    def update_q_table(self, state, action, next_state, reward):
        state_index = self.get_state_index(state)
        next_state_index = self.get_state_index(next_state) if next_state is not None else None
        max_q_value = np.max(self.q_table[next_state_index]) if next_state is not None else 0
        self.q_table[state_index, action] = (1 - self.learning_rate) * self.q_table[state_index, action] + \
            self.learning_rate * (reward + self.discount_factor * max_q_value)

def evaluate_agents(agent1, agent2, num_episodes=1000):
    environment = TicTacToeEnvironment()
    agent1_wins = 0
    agent2_wins = 0
    draws = 0

    for _ in range(num_episodes):
        environment.reset()
        current_agent = agent1
        while not environment.is_terminal():
            available_moves = environment.get_available_moves()
```

```
current_state = environment.state.copy()
action = current_agent.choose_action(current_state, available_moves)
environment.make_move(action, 1 if current_agent == agent1 else -1)

if environment.check_win(1 if current_agent == agent1 else -1):
    current_agent.update_q_table(current_state, action, None, 10)
    if current_agent == agent1:
        agent1_wins += 1
    else:
        agent2_wins += 1
    break

elif environment.is_draw():
    current_agent.update_q_table(current_state, action, None, 0)
    draws += 1
    break

next_state = environment.state.copy()
reward = 0
if environment.check_win(1 if current_agent == agent1 else -1):
    reward = -10
current_agent.update_q_table(current_state, action, next_state, reward)

current_agent = agent2 if current_agent == agent1 else agent1

return agent1_wins, agent2_wins, draws

# Create agents
agent1 = QLearningAgent()
agent2 = QLearningAgent()

# Evaluate agents
agent1_wins, agent2_wins, draws = evaluate_agents(agent1, agent2)

# Print results
print(f"Agent 1 wins: {agent1_wins}")
print(f"Agent 2 wins: {agent2_wins}")
print(f"Draws: {draws}")

Agent 1 wins: 458
Agent 2 wins: 470
Draws: 72
```

**TicTacToeEnvironment:**

This class represents the Tic-Tac-Toe game environment. It maintains the current state of the game, checks for a win or draw, and provides methods to reset the game and make moves.

The `__init__` method initializes the game state and sets the terminal flag to False.

The `reset` method resets the game state and the terminal flag.

The `get_available_moves` method returns a list of indices representing the available moves in the current game state.

The `make_move` method updates the game state by placing a player's mark at the specified move index.

The `check_win` method checks if a player has won the game by examining the current state.

The `is_draw` method checks if the game has ended in a draw.

**QLearningAgent:**

This class represents the Q-learning agent. It learns to play Tic-Tac-Toe by updating a Q-table based on the rewards received during gameplay.

The `__init__` method initializes the learning rate, discount factor, exploration rate, and the Q-table.

The `get_state_index` method converts the current game state into a unique index for indexing the Q-table.

The `choose_action` method selects the action (move) to be taken based on the current game state and the exploration-exploitation tradeoff using the epsilon-greedy policy.

The `update_q_table` method updates the Q-table based on the current state, action, next state, and the reward received.

**evaluate\_agents:**

This function performs the evaluation of two Q-learning agents by playing multiple episodes of Tic-Tac-Toe games.

It takes the two agents and the number of episodes to play as input.

In each episode, the environment is reset, and the agents take turns making moves until the game is over (either a win or a draw).

The agents update their Q-tables based on the rewards received during the episode.

The function keeps track of the wins and draws for each agent and returns the counts.

**Main code:**

The main code creates two Q-learning agents, `agent1` and `agent2`, using the `QLearningAgent` class.

The `evaluate_agents` function is called to evaluate the agents by playing a specified number of episodes.

The results (number of wins and draws) for each agent are printed.

The Q-learning algorithm involves the following steps:

The agents choose their moves based on the current game state and the exploration-exploitation policy.

The environment updates the game state based on the chosen moves.

The environment checks if the game has ended (win or draw).

The agents update their Q-tables based on the rewards received.

The agents continue playing until the specified number of episodes is completed.

<b>Lab Assignment No.</b>	7
<b>Title</b>	Analyzing Sales Data from Multiple File Formats
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 7      (Group B)****Aim:**

Analyzing Sales Data from Multiple File Formats

**Problem Statement:**

Analyzing Sales Data from Multiple File Formats

**Dataset: Sales data in multiple file formats (e.g., CSV, Excel, JSON)****Description:**

The goal is to load and analyze sales data from different file formats, including CSV, Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset.

**Tasks to Perform:**

Obtain sales data files in various formats, such as CSV, Excel, and JSON.

1. Load the sales data from each file format into the appropriate data structures or dataframes.

2. Explore the structure and content of the loaded data, identifying any inconsistencies, missing values, or data quality issues.

3. Perform data cleaning operations, such as handling missing values, removing duplicates, or correcting inconsistencies.

4. Convert the data into a unified format, such as a common dataframe or data structure, to enable seamless analysis.

5. Perform data transformation tasks, such as merging multiple datasets, splitting columns, or deriving new variables.

6. Analyze the sales data by performing descriptive statistics, aggregating data by specific variables, or calculating metrics such as total sales, average order value, or product category distribution.

7. Create visualizations, such as bar plots, pie charts, or box plots, to represent the sales data and gain insights into sales trends, customer behavior, or product performance.

**Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

**Software Requirement:**

Jupyter Notebook/Ubuntu

## Implementation:

```

import requests
import pandas as pd
import datetime
# Set your OpenWeatherMap API key
api_key = 'fb365aa6104829b44455572365ff3b4e'
Get the lat(itude) and lon(gitude) from here.
# Set the location for which you want to retrieve weather data
lat = 18.184135
lon = 74.610764
#https://openweathermap.org/api/one-call-3#how How to use api call
# Construct the API URL
api_url = f"http://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={api_key}"
# Send a GET request to the API
response = requests.get(api_url)
weather_data = response.json()
weather_data.keys()
#pass response to weather_data object(dictionary)
dict_keys(['cod', 'message', 'cnt', 'list', 'city'])
weather_data['list'][0]
{'dt': 1690189200,
'main': {'temp': 298.21,
'feels_like': 298.81,
'temp_min': 298.1,
'temp_max': 298.21,
'pressure': 1006,
'sea_level': 1006,
'grnd_level': 942,
'humidity': 78,
'temp_kf': 0.11},
'weather': [{"id": 804,
'main': 'Clouds',
'description': 'overcast clouds',
'icon': '04d'}],
'clouds': {'all': 100},
'wind': {'speed': 6.85, 'deg': 258, 'gust': 12.9},
'vesibility': 10000,
'pop': 0.59,
'sys': {'pod': 'd'},
'dt_txt': '2023-07-24 09:00:00'}

len(weather_data['list'])
40
weather_data['list'][0]['weather'][0]['description']
{"type": "string"}
#getting the data from dictionary and taking into one variable
# Extract relevant weather attributes using list comprehension
temperatures = [item['main']['temp'] for item in weather_data['list']]
#it will extract all values (40)
and putting into one variable
timestamps = [pd.to_datetime(item['dt'], unit='s') for item in weather_data['list']]
temperature = [item['main']['temp'] for item in weather_data['list']]

```

```

humidity = [item['main']['humidity'] for item in weather_data['list']]
wind_speed = [item['wind']['speed'] for item in weather_data['list']]
weather_description = [item['weather'][0]['description'] for item in weather_data['list']]
# Create a pandas DataFrame with the extracted weather data
weather_df = pd.DataFrame({
    'Timestamp': timestamps,
    'Temperature': temperatures,
    'humidity': humidity,
    'wind_speed': wind_speed,
    'weather_description': weather_description,
})
# Set the Timestamp column as the DataFrame's index
weather_df.set_index('Timestamp', inplace=True)
max_temp = weather_df['Temperature'].max()
max_temp
298.9
min_temp = weather_df['Temperature'].min()
min_temp
294.92
# Clean and preprocess the data
# Handling missing values
weather_df.fillna(0, inplace=True) # Replace missing values with 0 or appropriate value
# Handling inconsistent format (if applicable)
weather_df['Temperature'] = weather_df['Temperature'].apply(lambda x: x - 273.15 if isinstance(x, float)
else x) # Convert temperature from Kelvin to Celsius

# Print the cleaned and preprocessed data
print(weather_df)
   Temperature  humidity  wind_speed weather_description
Timestamp
2023-07-24 09:00:00    25.06     78      6.85  overcast clouds
2023-07-24 12:00:00    24.52     81      6.92  light rain
2023-07-24 15:00:00    23.73     84      7.18  light rain
2023-07-24 18:00:00    23.69     83      6.44  light rain
2023-07-24 21:00:00    23.06     85      5.54  light rain
2023-07-25 00:00:00    22.28     92      4.57  moderate rain
2023-07-25 03:00:00    22.46     92      3.95  moderate rain
2023-07-25 06:00:00    22.98     90      6.10  moderate rain
2023-07-25 09:00:00    24.55     79      6.46  light rain
2023-07-25 12:00:00    23.53     84      5.00  light rain
2023-07-25 15:00:00    22.87     88      5.00  overcast clouds
2023-07-25 18:00:00    22.77     89      3.93  overcast clouds
2023-07-25 21:00:00    22.56     84      5.47  overcast clouds
2023-07-26 00:00:00    22.35     87      3.97  overcast clouds
2023-07-26 03:00:00    23.05     85      3.47  light rain
2023-07-26 06:00:00    23.34     85      3.84  light rain
2023-07-26 09:00:00    23.08     89      4.16  light rain
2023-07-26 12:00:00    24.09     83      5.52  light rain
2023-07-26 15:00:00    23.10     87      5.59  light rain
2023-07-26 18:00:00    22.43     91      5.42  light rain
2023-07-26 21:00:00    22.29     92      5.17  light rain
2023-07-27 00:00:00    22.53     90      5.31  light rain
2023-07-27 03:00:00    22.78     88      4.30  light rain
2023-07-27 06:00:00    22.83     90      5.19  moderate rain
2023-07-27 09:00:00    22.57     91      6.65  moderate rain
2023-07-27 12:00:00    22.28     91      5.27  moderate rain

```

2023-07-27 15:00:00	22.03	93	5.12	light rain
2023-07-27 18:00:00	21.82	92	4.65	light rain
2023-07-27 21:00:00	21.77	90	5.27	light rain
2023-07-28 00:00:00	22.01	88	5.41	light rain
2023-07-28 03:00:00	23.30	81	6.19	overcast clouds
2023-07-28 06:00:00	25.19	72	7.19	light rain
2023-07-28 09:00:00	24.95	76	7.22	light rain
2023-07-28 12:00:00	24.72	75	6.93	overcast clouds
2023-07-28 15:00:00	23.41	83	5.12	overcast clouds
2023-07-28 18:00:00	22.76	86	4.56	overcast clouds
2023-07-28 21:00:00	22.63	87	4.15	overcast clouds
2023-07-29 00:00:00	22.74	84	4.35	overcast clouds
2023-07-29 03:00:00	23.87	77	6.16	overcast clouds
2023-07-29 06:00:00	25.75	66	7.23	overcast clouds

```

import matplotlib.pyplot as plt
daily_mean_temp = weather_df['Temperature'].resample('D').mean()
daily_mean_humidity = weather_df['humidity'].resample('D').mean()
daily_mean_wind_speed = weather_df['wind_speed'].resample('D').mean()
# Plot the mean daily temperature over time (Line plot)
plt.figure(figsize=(10, 6))
daily_mean_temp.plot(color='red', linestyle='-', marker='o')
plt.title('Mean Daily Temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()
# Plot the mean daily humidity over time (Bar plot)
plt.figure(figsize=(10, 6))
daily_mean_humidity.plot(kind='bar', color='blue')
plt.title('Mean Daily Humidity')
plt.xlabel('Date')
plt.ylabel('Humidity (%)')
plt.grid(True)
plt.show()
# Plot the relationship between temperature and wind speed (Scatter plot)
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['wind_speed'], color='green')
plt.title('Temperature vs. Wind Speed')
plt.xlabel('Temperature (°C)')
plt.ylabel('Wind Speed (m/s)')
plt.grid(True)
plt.show()
###Heatmap
import seaborn as sns
heatmap_data = weather_df[['Temperature', 'humidity']]
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm')
plt.title('Temperature vs Humidity Heatmap')
plt.show()
# Create a scatter plot to visualize the relationship between temperature and humidity
plt.scatter(weather_df['Temperature'], weather_df['humidity'])
plt.xlabel('Temperature (°C)')
plt.ylabel('Humidity (%)')
plt.title('Temperature vs Humidity Scatter Plot')
plt.show()
###Geospatial Map
import requests

```

```

import pandas as pd
import geopandas as gpd
import folium

# Set your OpenWeatherMap API key
api_key = 'fb365aa6104829b44455572365ff3b4e'

# Specify the locations for which you want to retrieve weather data
locations = ['London', 'Paris', 'New York']

weather_df = pd.DataFrame()

# Retrieve weather data for each location
for location in locations:
    # Construct the API URL
    api_url = f'http://api.openweathermap.org/data/2.5/weather?q={location}&appid={api_key}'

    # Send a GET request to the API
    response = requests.get(api_url)
    weather_data = response.json()

    # Extract relevant weather attributes
    temperature = weather_data['main']['temp']
    humidity = weather_data['main']['humidity']
    wind_speed = weather_data['wind']['speed']
    latitude = weather_data['coord']['lat']
    longitude = weather_data['coord']['lon']

    # Create a DataFrame for the location's weather data
    location_df = pd.DataFrame({
        'Location': [location],
        'Temperature': [temperature],
        'Humidity': [humidity],
        'Wind Speed': [wind_speed],
        'Latitude': [latitude],
        'Longitude': [longitude]
    })

    # Append the location's weather data to the main DataFrame
    weather_df = weather_df.append(location_df, ignore_index=True)

<ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will
be removed from pandas in a future version. Use pandas.concat instead.
    weather_df = weather_df.append(location_df, ignore_index=True)

<ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will
be removed from pandas in a future version. Use pandas.concat instead.
    weather_df = weather_df.append(location_df, ignore_index=True)
weather_df

   Location  Temperature  Humidity  Wind Speed  Latitude  Longitude
0  London      289.02      88       3.60    51.5085   -0.1257
1  Paris       290.96      83       6.17    48.8534    2.3488
2  New York    296.82      61       4.47    40.7143   -74.0060

# Load a world map shapefile using geopandas
world_map = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))

```

```
# Rename the column used for merging in the world map DataFrame
world_map.rename(columns={'name': 'Location'}, inplace=True)

# Merge the weather data with the world map based on location
weather_map = world_map.merge(weather_df, on='Location')

# Create a folium map centered around the mean latitude and longitude of all locations
map_center = [weather_df['Latitude'].mean(), weather_df['Longitude'].mean()]
weather_map_folium = folium.Map(location=map_center, zoom_start=2)

# Add weather markers to the folium map
for index, row in weather_map.iterrows():
    location = [row['Latitude'], row['Longitude']]
    temperature = row['Temperature']
    marker_text = f'Temperature: {temperature} K'
    folium.Marker(location, popup=marker_text, icon=folium.Icon(icon='cloud',
color='red')).add_to(weather_map_folium)
# display the folium map
weather_map_folium
<ipython-input-19-c9bd718791be>:2: FutureWarning: The geopandas.dataset module is deprecated and
will be removed in GeoPandas 1.0. You can get the original 'naturalearth_cities' data from
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.
    world_map = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
<folium.folium.Map at 0x7f242a56f430>
type(weather_map_folium)
folium.folium.Map
```

<b>Lab Assignment No.</b>	8
<b>Title</b>	Interacting with Web APIs Analyzing Weather Data from OpenWeatherMap API
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 8      (Group B)****Aim:**

Interacting with Web APIs

**Problem Statement:**

Analyzing Weather Data from OpenWeatherMap API

**Dataset:** Weather data retrieved from OpenWeatherMap API

**Description:**

The goal is to interact with the OpenWeatherMap API to retrieve weather data for a specific location and perform data modeling and visualization to analyze weather patterns over time.

Tasks to Perform:

1. Register and obtain API key from OpenWeatherMap.
2. Interact with the OpenWeatherMap API using the API key to retrieve weather data for a specific location.
3. Extract relevant weather attributes such as temperature, humidity, wind speed, and precipitation from the API response.
4. Clean and preprocess the retrieved data, handling missing values or inconsistent formats.
5. Perform data modeling to analyze weather patterns, such as calculating average temperature, maximum/minimum values, or trends over time.
6. Visualize the weather data using appropriate plots, such as line charts, bar plots, or scatter plots, to represent temperature changes, precipitation levels, or wind speed variations.
7. Apply data aggregation techniques to summarize weather statistics by specific time periods (e.g., daily, monthly, seasonal).
8. Incorporate geographical information, if available, to create maps or geospatial visualizations representing weather patterns across different locations.
9. Explore and visualize relationships between weather attributes, such as temperature and humidity, using correlation plots or heatmaps.

**Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

**Software Requirement:**

Jupyter Notebook/Ubuntu

**Implementation:**

```
import requests  
import pandas as pd  
import datetime  
  
# Set your OpenWeatherMap API key  
api_key = 'fb365aa6104829b44455572365ff3b4e'
```

Get the lat(itude) and lon(gitude) from [here](#).

```
# Set the location for which you want to retrieve weather data  
lat = 18.184135  
lon = 74.610764
```

```
#https://openweathermap.org/api/one-call-3#how    How to use api call  
# Construct the API URL  
api_url = f"http://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={api_key}"
```

```
# Send a GET request to the API  
response = requests.get(api_url)  
weather_data = response.json()      #pass response to weather_data object(dictionary)
```

```
weather_data.keys()
```

```
dict_keys(['cod', 'message', 'cnt', 'list', 'city'])
```

```
weather_data['list'][0]
```

```
{'dt': 1690189200,  
'main': {'temp': 298.21,  
'feels_like': 298.81,  
'temp_min': 298.1,  
'temp_max': 298.21,  
'pressure': 1006,  
'sea_level': 1006,  
'grnd_level': 942,  
'humidity': 78,  
'temp_kf': 0.11},  
'weather': [{"id": 804,  
'main': 'Clouds',  
'description': 'overcast clouds',  
'icon': '04d'}],  
'clouds': {'all': 100},
```

```
'wind': {'speed': 6.85, 'deg': 258, 'gust': 12.9},  
'visibility': 10000,  
'pop': 0.59,  
'sys': {'pod': 'd'},  
'dt_txt': '2023-07-24 09:00:00'}  
len(weather_data['list'])
```

40

```
weather_data['list'][0]['weather'][0]['description']
```

```
{"type": "string"}
```

```
#getting the data from dictionary and taking into one variable  
# Extract relevant weather attributes using list comprehension  
temperatures = [item['main']['temp'] for item in weather_data['list']] #it will extract all values (40)  
and putting into one variable
```

```
timestamps = [pd.to_datetime(item['dt'], unit='s') for item in weather_data['list']]
```

```
temperature = [item['main']['temp'] for item in weather_data['list']]
```

```
humidity = [item['main']['humidity'] for item in weather_data['list']]
```

```
wind_speed = [item['wind']['speed'] for item in weather_data['list']]
```

```
weather_description = [item['weather'][0]['description'] for item in weather_data['list']]
```

```
# Create a pandas DataFrame with the extracted weather data
```

```
weather_df = pd.DataFrame({  
    'Timestamp': timestamps,  
    'Temperature': temperatures,  
    'humidity': humidity,  
    'wind_speed': wind_speed,  
    'weather_description': weather_description,  
})
```

```
# Set the Timestamp column as the DataFrame's index
```

```
weather_df.set_index('Timestamp', inplace=True)
```

```
max_temp = weather_df['Temperature'].max()
```

```
max_temp
```

298.9

```
min_temp = weather_df['Temperature'].min()
```

```
min_temp
```

294.92

```
# Clean and preprocess the data
```

```
# Handling missing values
```

```
weather_df.fillna(0, inplace=True) # Replace missing values with 0 or appropriate value
```

```
# Handling inconsistent format (if applicable)
weather_df['Temperature'] = weather_df['Temperature'].apply(lambda x: x - 273.15 if isinstance(x, float) else x) # Convert temperature from Kelvin to Celsius
```

```
# Print the cleaned and preprocessed data
print(weather_df)
```

Timestamp	Temperature	humidity	wind_speed	weather_description
2023-07-24 09:00:00	25.06	78	6.85	overcast clouds
2023-07-24 12:00:00	24.52	81	6.92	light rain
2023-07-24 15:00:00	23.73	84	7.18	light rain
2023-07-24 18:00:00	23.69	83	6.44	light rain
2023-07-24 21:00:00	23.06	85	5.54	light rain
2023-07-25 00:00:00	22.28	92	4.57	moderate rain
2023-07-25 03:00:00	22.46	92	3.95	moderate rain
2023-07-25 06:00:00	22.98	90	6.10	moderate rain
2023-07-25 09:00:00	24.55	79	6.46	light rain
2023-07-25 12:00:00	23.53	84	5.00	light rain
2023-07-25 15:00:00	22.87	88	5.00	overcast clouds
2023-07-25 18:00:00	22.77	89	3.93	overcast clouds
2023-07-25 21:00:00	22.56	84	5.47	overcast clouds
2023-07-26 00:00:00	22.35	87	3.97	overcast clouds
2023-07-26 03:00:00	23.05	85	3.47	light rain
2023-07-26 06:00:00	23.34	85	3.84	light rain
2023-07-26 09:00:00	23.08	89	4.16	light rain
2023-07-26 12:00:00	24.09	83	5.52	light rain
2023-07-26 15:00:00	23.10	87	5.59	light rain
2023-07-26 18:00:00	22.43	91	5.42	light rain
2023-07-26 21:00:00	22.29	92	5.17	light rain
2023-07-27 00:00:00	22.53	90	5.31	light rain
2023-07-27 03:00:00	22.78	88	4.30	light rain
2023-07-27 06:00:00	22.83	90	5.19	moderate rain
2023-07-27 09:00:00	22.57	91	6.65	moderate rain
2023-07-27 12:00:00	22.28	91	5.27	moderate rain
2023-07-27 15:00:00	22.03	93	5.12	light rain
2023-07-27 18:00:00	21.82	92	4.65	light rain
2023-07-27 21:00:00	21.77	90	5.27	light rain
2023-07-28 00:00:00	22.01	88	5.41	light rain
2023-07-28 03:00:00	23.30	81	6.19	overcast clouds
2023-07-28 06:00:00	25.19	72	7.19	light rain
2023-07-28 09:00:00	24.95	76	7.22	light rain
2023-07-28 12:00:00	24.72	75	6.93	overcast clouds
2023-07-28 15:00:00	23.41	83	5.12	overcast clouds
2023-07-28 18:00:00	22.76	86	4.56	overcast clouds
2023-07-28 21:00:00	22.63	87	4.15	overcast clouds
2023-07-29 00:00:00	22.74	84	4.35	overcast clouds
2023-07-29 03:00:00	23.87	77	6.16	overcast clouds
2023-07-29 06:00:00	25.75	66	7.23	overcast clouds

```
import matplotlib.pyplot as plt

daily_mean_temp = weather_df['Temperature'].resample('D').mean()
daily_mean_humidity = weather_df['humidity'].resample('D').mean()
daily_mean_wind_speed = weather_df['wind_speed'].resample('D').mean()

# Plot the mean daily temperature over time (Line plot)
plt.figure(figsize=(10, 6))
daily_mean_temp.plot(color='red', linestyle='-', marker='o')
plt.title('Mean Daily Temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()

# Plot the mean daily humidity over time (Bar plot)
plt.figure(figsize=(10, 6))
daily_mean_humidity.plot(kind='bar', color='blue')
plt.title('Mean Daily Humidity')
plt.xlabel('Date')
plt.ylabel('Humidity (%)')
plt.grid(True)
plt.show()

# Plot the relationship between temperature and wind speed (Scatter plot)
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['wind_speed'], color='green')
plt.title('Temperature vs. Wind Speed')
plt.xlabel('Temperature (°C)')
plt.ylabel('Wind Speed (m/s)')
plt.grid(True)
plt.show()

###Heatmap

import seaborn as sns

heatmap_data = weather_df[['Temperature', 'humidity']]
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm')
plt.title('Temperature vs Humidity Heatmap')
plt.show()

# Create a scatter plot to visualize the relationship between temperature and humidity
plt.scatter(weather_df['Temperature'], weather_df['humidity'])
plt.xlabel('Temperature (°C)')
plt.ylabel('Humidity (%)')
```

```
plt.title('Temperature vs Humidity Scatter Plot')
plt.show()

###Geospatial Map

import requests
import pandas as pd
import geopandas as gpd
import folium

# Set your OpenWeatherMap API key
api_key = 'fb365aa6104829b44455572365ff3b4e'

# Specify the locations for which you want to retrieve weather data
locations = ['London', 'Paris', 'New York']

weather_df = pd.DataFrame()

# Retrieve weather data for each location
for location in locations:
    # Construct the API URL
    api_url = f'http://api.openweathermap.org/data/2.5/weather?q={location}&appid={api_key}'

    # Send a GET request to the API
    response = requests.get(api_url)
    weather_data = response.json()
    # Extract relevant weather attributes
    temperature = weather_data['main']['temp']
    humidity = weather_data['main']['humidity']
    wind_speed = weather_data['wind']['speed']
    latitude = weather_data['coord']['lat']
    longitude = weather_data['coord']['lon']
    # Create a DataFrame for the location's weather data
    location_df = pd.DataFrame({
        'Location': [location],
        'Temperature': [temperature],
        'Humidity': [humidity],
        'Wind Speed': [wind_speed],
        'Latitude': [latitude],
        'Longitude': [longitude]
    })

    # Append the location's weather data to the main DataFrame
    weather_df = weather_df.append(location_df, ignore_index=True)

<ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will
be removed from pandas in a future version. Use pandas.concat instead.
    weather_df = weather_df.append(location_df, ignore_index=True)
```

```
<ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will
be removed from pandas in a future version. Use pandas.concat instead.
weather_df = weather_df.append(location_df, ignore_index=True)
<ipython-input-17-68826faaad0a>:41: FutureWarning: The frame.append method is deprecated and will
be removed from pandas in a future version. Use pandas.concat instead.
weather_df = weather_df.append(location_df, ignore_index=True)
```

```
weather_df
```

	Location	Temperature	Humidity	Wind Speed	Latitude	Longitude
0	London	289.02	88	3.60	51.5085	-0.1257
1	Paris	290.96	83	6.17	48.8534	2.3488
2	New York	296.82	61	4.47	40.7143	-74.0060

```
# Load a world map shapefile using geopandas
world_map = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
```

```
# Rename the column used for merging in the world map DataFrame
world_map.rename(columns={'name': 'Location'}, inplace=True)
```

```
# Merge the weather data with the world map based on location
weather_map = world_map.merge(weather_df, on='Location')
```

```
# Create a folium map centered around the mean latitude and longitude of all locations
map_center = [weather_df['Latitude'].mean(), weather_df['Longitude'].mean()]
weather_map_folium = folium.Map(location=map_center, zoom_start=2)
```

```
# Add weather markers to the folium map
for index, row in weather_map.iterrows():
```

```
    location = [row['Latitude'], row['Longitude']]
    temperature = row['Temperature']
    marker_text = f'Temperature: {temperature} K'
    folium.Marker(location, popup=marker_text, icon=folium.Icon(icon='cloud',
color='red')).add_to(weather_map_folium)
# display the folium map
weather_map_folium
```

```
<ipython-input-19-c9bd718791be>:2: FutureWarning: The geopandas.dataset module is deprecated and
will be removed in GeoPandas 1.0. You can get the original 'naturalearth_cities' data from
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.
```

```
world_map = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
```

```
<folium.folium.Map at 0x7f242a56f430>
```

```
type(weather_map_folium)
```

```
folium.folium.Map
```

<b>Lab Assignment No.</b>	9
<b>Title</b>	Analyzing Customer Churn in a Telecommunications Company
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 9                    (Group B)****Aim:**

Data Cleaning and Preparation

**Problem Statement:**

Analyzing Customer Churn in a Telecommunications Company

**Dataset:** "Telecom\_Customer\_Churn.csv"

**Description:**

The dataset contains information about customers of a telecommunications company and whether they have churned (i.e., discontinued their services). The dataset includes various attributes of the customers, such as their demographics, usage patterns, and account information. The goal is to perform data cleaning and preparation to gain insights into the factors that contribute to customer churn.

**Steps to Perform:**

1. Import the "Telecom\_Customer\_Churn.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Handle missing values in the dataset, deciding on an appropriate strategy.
4. Remove any duplicate records from the dataset.
5. Check for inconsistent data, such as inconsistent formatting or spelling variations, and standardize it.
6. Convert columns to the correct data types as needed.
7. Identify and handle outliers in the data.

**Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

**Software Requirement:**

Jupyter Notebook/Ubuntu

**Implementation:**

Import necessary libraries

```
import pandas as pd                    #data manipulation
```

```

import numpy as np          #numerical computations
from sklearn.model_selection import train_test_split      # scikit-learn for machine learning models
split the dataset into training and testing sets for model evaluation
from sklearn import metrics           #evaluating the performance of machine learning models
Load the dataset

data = pd.read_csv("Telecom_Customer_Churn.csv")
print(data.index)

RangeIndex(start=0, stop=7043, step=1)

```

Explore the dataset

```
print(data)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	\
0	7590-VHVEG	Female	0	Yes	No	1	
1	5575-GNVDE	Male	0	No	No	34	
2	3668-QPYBK	Male	0	No	No	2	
3	7795-CFOCW	Male	0	No	No	45	
4	9237-HQITU	Female	0	No	No	2	
...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	
7039	2234-XADUH	Female	0	Yes	Yes	72	
7040	4801-JZAZL	Female	0	Yes	Yes	11	
7041	8361-LTMKD	Male	1	Yes	No	4	
7042	3186-AJIEK	Male	0	No	No	66	

	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	\
0	No	No	phone service	DSL	No	...
1	Yes		No	DSL	Yes	...
2	Yes		No	DSL	Yes	...
3	No	No	phone service	DSL	Yes	...
4	Yes		No	Fiber optic	No	...
...	...	...	...	...	...	...
7038	Yes		Yes	DSL	Yes	...
7039	Yes		Yes	Fiber optic	No	...
7040	No	No	phone service	DSL	Yes	...
7041	Yes		Yes	Fiber optic	No	...
7042	Yes		No	Fiber optic	Yes	...

	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	\
0	No	No	No	No	Month-to-month	
1	Yes	No	No	No	One year	
2	No	No	No	No	Month-to-month	
3	Yes	Yes		No	No	One year
4	No	No	No	No	Month-to-month	
...	...	...	...	...	...	...
7038	Yes	Yes	Yes	Yes	One year	
7039	Yes	No	Yes	Yes	One year	

7040	No	No	No	No	Month-to-month
7041	No	No	No	No	Month-to-month
7042	Yes	Yes	Yes	Yes	Two year

	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	\
0	Yes	Electronic check	29.85	29.85	
1	No	Mailed check	56.95	1889.5	
2	Yes	Mailed check	53.85	108.15	
3	No	Bank transfer (automatic)	42.30	1840.75	
4	Yes	Electronic check	70.70	151.65	
...	...	...	...	...	
7038	Yes	Mailed check	84.80	1990.5	
7039	Yes	Credit card (automatic)	103.20	7362.9	
7040	Yes	Electronic check	29.60	346.45	
7041	Yes	Mailed check	74.40	306.6	
7042	Yes	Bank transfer (automatic)	105.65	6844.5	

	Churn
0	No
1	No
2	Yes
3	No
4	Yes
...	...
7038	No
7039	No
7040	No
7041	Yes
7042	No

[7043 rows x 21 columns]

```
print(data.columns)
```

```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

```
data.shape
```

```
(7043, 21)
```

```
print(data.head())
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	\
0	7590-VHVEG	Female	0	Yes	No	1	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	

```

3 7795-CFOCW Male 0 No No 45 No
4 9237-HQITU Female 0 No No 2 Yes

```

```

MultipleLines InternetService OnlineSecurity ... DeviceProtection \
0 No phone service DSL No ... No
1 No DSL Yes ... Yes
2 No DSL Yes ... No
3 No phone service DSL Yes ... Yes
4 No Fiber optic No ... No

```

```

TechSupport StreamingTV StreamingMovies Contract PaperlessBilling \
0 No No No Month-to-month Yes
1 No No No One year No
2 No No No Month-to-month Yes
3 Yes No No No One year No
4 No No No Month-to-month Yes

```

```

PaymentMethod MonthlyCharges TotalCharges Churn
0 Electronic check 29.85 29.85 No
1 Mailed check 56.95 1889.5 No
2 Mailed check 53.85 108.15 Yes
3 Bank transfer (automatic) 42.30 1840.75 No
4 Electronic check 70.70 151.65 Yes

```

[5 rows x 21 columns]

```
print(data.tail())
```

```

customerID gender SeniorCitizen Partner Dependents tenure \
7038 6840-RESVB Male 0 Yes Yes 24
7039 2234-XADUH Female 0 Yes Yes 72
7040 4801-JZAZL Female 0 Yes Yes 11
7041 8361-LTMKD Male 1 Yes No 4
7042 3186-AJIEK Male 0 No No 66

```

```

PhoneService MultipleLines InternetService OnlineSecurity ... \
7038 Yes Yes DSL Yes ...
7039 Yes Yes Fiber optic No ...
7040 No No phone service DSL Yes ...
7041 Yes Yes Fiber optic No ...
7042 Yes No Fiber optic Yes ...

```

```

DeviceProtection TechSupport StreamingTV StreamingMovies Contract \
7038 Yes Yes Yes Yes One year
7039 Yes No Yes Yes One year
7040 No No No No Month-to-month
7041 No No No No Month-to-month
7042 Yes Yes Yes Yes Two year

```

	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	\
7038	Yes	Mailed check	84.80	1990.5	
7039	Yes	Credit card (automatic)	103.20	7362.9	
7040	Yes	Electronic check	29.60	346.45	
7041	Yes	Mailed check	74.40	306.6	
7042	Yes	Bank transfer (automatic)	105.65	6844.5	

	Churn
7038	No
7039	No
7040	No
7041	Yes
7042	No

[5 rows x 21 columns]

```
# to know unique values
data.nunique()
```

```
customerID      7043
gender          2
SeniorCitizen   2
Partner         2
Dependents     2
tenure          73
PhoneService    2
MultipleLines   3
InternetService 3
OnlineSecurity  3
OnlineBackup    3
DeviceProtection 3
TechSupport    3
StreamingTV    3
StreamingMovies 3
Contract        3
PaperlessBilling 2
PaymentMethod   4
MonthlyCharges  1585
TotalCharges    6531
Churn          2
dtype: int64
```

### Handle Missing Values

# data.isna().sum() is used to count the number of missing values (NaN values) in each column of a pandas DataFrame called data.

```
data.isna().sum()
```

```

customerID      0
gender          0
SeniorCitizen   0
Partner         0
Dependents     0
tenure          0
PhoneService    0
MultipleLines   0
InternetService 0
OnlineSecurity  0
OnlineBackup    0
DeviceProtection 0
TechSupport     0
StreamingTV    0
StreamingMovies 0
Contract        0
PaperlessBilling 0
PaymentMethod   0
MonthlyCharges  0
TotalCharges    0
Churn           0
dtype: int64

```

# isna() and isnull() are essentially the same method in Pandas, and they both return a boolean mask of the same shape as the input object, indicating where missing values (NaN or None) are present.  
`data.isnull().sum()`

```

customerID      0
gender          0
SeniorCitizen   0
Partner         0
Dependents     0
tenure          0
PhoneService    0
MultipleLines   0
InternetService 0
OnlineSecurity  0
OnlineBackup    0
DeviceProtection 0
TechSupport     0
StreamingTV    0
StreamingMovies 0
Contract        0
PaperlessBilling 0
PaymentMethod   0
MonthlyCharges  0
TotalCharges    0
Churn           0

```

```
dtype: int64
```

Remove Duplicate Records

```
# Check the number of rows before removing duplicates
print("Number of rows before removing duplicates:", len(data))
```

Number of rows before removing duplicates: 7043

```
# Remove duplicate records
data_cleaned = data.drop_duplicates()
```

```
# Check the number of rows after removing duplicates
print("Number of rows after removing duplicates:", len(data_cleaned))
```

Number of rows after removing duplicates: 7043

```
data.describe()
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

#Measure of frequency distribution

```
unique, counts = np.unique(data['tenure'], return_counts=True)
print(unique, counts)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72] [ 11 613 238 200 176 133 110 131 123 119 116 99 117 109 76 99 80 87
 97 73 71 63 90 85 94 79 79 72 57 72 72 65 69 64 65 88
 50 65 59 56 64 70 65 65 51 61 74 68 64 66 68 68 80 70
 68 64 80 65 67 60 76 76 70 72 80 76 89 98 100 95 119 170
 362]
```

#Measure of frequency distribution

```
unique, counts = np.unique(data['MonthlyCharges'], return_counts=True)
print(unique, counts)
```

```
[ 18.25 18.4 18.55 ... 118.6 118.65 118.75] [1 1 1 ... 2 1 1]
```

#Measure of frequency distribution

```
unique, counts = np.unique(data['TotalCharges'], return_counts=True)
print(unique, counts)

[ '' 100.2 '100.25' ... '999.45' '999.8' '999.9'] [11 1 1 ... 1 1 1]

# sns.pairplot(data) creates a grid of pairwise plots of the variables in a dataset, which can help you
quickly visualize the relationships between different pairs of variables.
import seaborn as sns      #Seaborn library for data visualization
sns.pairplot(data)

<seaborn.axisgrid.PairGrid at 0x7fb9cc97a680>
```

#### Check for Outliers

```
#checking boxplot for Fare column
import matplotlib.pyplot as plt      #pyplot module from the Matplotlib library
plt.boxplot(data['tenure'])
plt.show()

plt.boxplot(data['MonthlyCharges'])
plt.show()
```

#### Split the Data

```
X = data.drop("Churn", axis=1)
y = data["Churn"]
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

X\_train.shape

(5634, 20)

y\_train.shape

(5634,)

X\_test.shape

(1409, 20)

y\_test.shape

(1409,)

#### Export the cleaned data

```
# Export the cleaned dataset to a CSV file
data.to_csv("Cleaned_Telecom_Customer_Churn.csv", index=False)
```

<b>Lab Assignment No.</b>	10
<b>Title</b>	Data Wrangling on Real Estate Market
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 10      (Group B)****Aim:**

Data Wrangling

**Problem Statement:**

Data Wrangling on Real Estate Market

**Dataset:** "RealEstate\_Prices.csv"

**Description:**

The dataset contains information about housing prices in a specific real estate market. It includes various attributes such as property characteristics, location, sale prices, and other relevant features. The goal is to perform data wrangling to gain insights into the factors influencing housing prices and prepare the dataset for further analysis or modeling.

**Steps to Perform:**

1. Import the "RealEstate\_Prices.csv" dataset. Clean column names by removing spaces, special characters, or renaming them for clarity.
2. Handle missing values in the dataset, deciding on an appropriate strategy (e.g., imputation or removal).
3. Perform data merging if additional datasets with relevant information are available (e.g., neighborhood demographics or nearby amenities).
4. Filter and subset the data based on specific criteria, such as a particular time period, property type, or location.
5. Handle categorical variables by encoding them appropriately (e.g., one-hot encoding or label encoding) for further analysis.
6. Aggregate the data to calculate summary statistics or derived metrics such as average sale prices by neighborhood or property type.
7. Identify and handle outliers or extreme values in the data that may affect the analysis or modeling process.

**Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

**Software Requirement:**

Jupyter Notebook/Ubuntu

**Implementation:**

```
import pandas as pd
```

```
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20,10)
```

Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging.

```
df1 = pd.read_csv("/content/Bengaluru_House_Data.csv")
df1.head()
```

	area_type	availability	location	size	\
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	

	society	total_sqft	bath	balcony	price
0	Coomee	1056	2.0	1.0	39.07
1	Theanmp	2600	5.0	3.0	120.00
2	NaN	1440	2.0	3.0	62.00
3	Soiewre	1521	3.0	1.0	95.00
4	NaN	1200	2.0	1.0	51.00

```
df1.shape
```

```
(13320, 9)
```

```
df1.columns
```

```
Index(['area_type', 'availability', 'location', 'size', 'society',
       'total_sqft', 'bath', 'balcony', 'price'],
      dtype='object')
```

```
df1['area_type']
```

```
0      Super built-up Area
1          Plot Area
2      Built-up Area
3      Super built-up Area
4      Super built-up Area
...
13315      Built-up Area
13316 Super built-up Area
13317      Built-up Area
13318 Super built-up Area
13319 Super built-up Area
Name: area_type, Length: 13320, dtype: object
```

```
df1['area_type'].unique()
```

```
array(['Super built-up Area', 'Plot Area', 'Built-up Area',
       'Carpet Area'], dtype=object)
```

```
df1['area_type'].value_counts()
```

```
Super built-up Area    8790
Built-up Area        2418
Plot Area            2025
Carpet Area           87
Name: area_type, dtype: int64
```

Drop features that are not required to build our model

```
df2 = df1.drop(['area_type','society','balcony','availability'],axis='columns')
df2.shape
```

```
(13320, 5)
```

```
df2.isnull().sum()
```

```
location      1
size         16
total_sqft     0
bath          73
price          0
dtype: int64
```

```
df2.shape
```

```
(13320, 5)
```

```
df3 = df2.dropna()
df3.isnull().sum()
```

```
location      0
size          0
total_sqft    0
bath          0
price          0
dtype: int64
```

```
df3.shape
```

```
(13246, 5)
```

```
df3['size'].unique()
```

```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)
```

```
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))
```

<ipython-input-15-4c4c73fbe7f4>:1: 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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))
```

```
df3.head()
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00	4
2	Uttarahalli	3 BHK	1440	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00	3
4	Kothanur	2 BHK	1200	2.0	51.00	2

```
df3.bhk.unique()
```

```
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
       13, 18])
```

```
df3[df3.bhk>20]
```

	location	size	total_sqft	bath	price	bhk
1718	2 Electronic City Phase II	27 BHK	8000	27.0	230.0	27
4684	Munnekollal	43 Bedroom	2400	40.0	660.0	43

```
df3.total_sqft.unique()
```

```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

## Explore total\_sqft feature

```
def is_float(x):
```

```
    try:
```

```
        float(x)
```

```
    except:
```

```
        return False
```

```
    return True
```

```
df3[~df3['total_sqft'].apply(is_float)].head(10)
```

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2
410	Kengeri	1 BHK 34.46Sq. Meter	1.0	18.500		1
549	Hennur Road	2 BHK	1195 - 1440	2.0	63.770	2
648	Arekere	9 Bedroom	4125Perch	9.0	265.000	9
661	Yelahanka	2 BHK	1120 - 1145	2.0	48.130	2
672	Bettahalsoor	4 Bedroom	3090 - 5002	4.0	445.000	4

```
def convert_sqft_to_num(x):
```

```
    tokens = x.split('-')
```

```
    if len(tokens) == 2:
```

```
        return (float(tokens[0])+float(tokens[1]))/2
```

```
    try:
```

```
        return float(x)
```

```
    except:
```

```
        return None
```

```
convert_sqft_to_num('2100 - 2850')
```

---

2475.0

```
convert_sqft_to_num('34.46Sq. Meter')
```

```
df4 = df3.copy()
df4.total_sqft = df4.total_sqft.apply(convert_sqft_to_num)
df4
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3
4	Kothanur	2 BHK	1200.0	2.0	51.00	2
...	...	...	...	...	...	...
13315	Whitefield	5 Bedroom	3453.0	4.0	231.00	5
13316	Richards Town	4 BHK	3600.0	5.0	400.00	4
13317	Raja Rajeshwari Nagar	2 BHK	1141.0	2.0	60.00	2
13318	Padmanabhanagar	4 BHK	4689.0	4.0	488.00	4
13319	Doddathoguru	1 BHK	550.0	1.0	17.00	1

[13246 rows x 6 columns]

```
df4 = df4[df4.total_sqft.notnull()]
df4
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3
4	Kothanur	2 BHK	1200.0	2.0	51.00	2
...	...	...	...	...	...	...

13315	Whitefield	5 Bedroom	3453.0	4.0	231.00	5
13316	Richards Town	4 BHK	3600.0	5.0	400.00	4
13317	Raja Rajeshwari Nagar	2 BHK	1141.0	2.0	60.00	2
13318	Padmanabhanagar	4 BHK	4689.0	4.0	488.00	4
13319	Doddathoguru	1 BHK	550.0	1.0	17.00	1

[13200 rows x 6 columns]

For below row, it shows total\_sqft as 2475 which is an average of the range 2100-2850

df4.loc[30]

location	Yelahanka
size	4 BHK
total_sqft	2475.0
bath	4.0
price	186.0
bhk	4

Name: 30, dtype: object

$(2100 + 2850)/2$

2475.0

Add new feature called price per square feet

```
df5 = df4.copy()
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
df5.head()
```

	location	size	total_sqft	bath	price	bhk	\
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	

```
3     Lingadheeranahalli    3 BHK      1521.0  3.0  95.00  3
4       Kothanur        2 BHK      1200.0  2.0  51.00  2
```

```
price_per_sqft
```

```
0   3699.810606
1   4615.384615
2   4305.555556
3   6245.890861
4   4250.000000
```

```
df5_stats = df5['price_per_sqft'].describe()
```

```
df5_stats
```

```
count 1.320000e+04
```

```
mean  7.920759e+03
```

```
std   1.067272e+05
```

```
min   2.678298e+02
```

```
25%   4.267701e+03
```

```
50%   5.438331e+03
```

```
75%   7.317073e+03
```

```
max   1.200000e+07
```

```
Name: price_per_sqft, dtype: float64
```

```
df5.to_csv("bhp.csv",index=False)
```

Examine locations which is a categorical variable. We need to apply dimensionality reduction technique here to reduce number of locations

```
len(df5.location.unique())
```

```
1298
```

```
df5.location = df5.location.apply(lambda x: x.strip())
```

```
location_stats = df5['location'].value_counts(ascending=False)
```

```
location_stats
```

```
Whitefield      533
Sarjapur Road  392
Electronic City 304
Kanakpura Road   264
Thanisandra     235
...
Rajanna Layout    1
Subramanyanagar  1
Lakshmipura Vidyapura 1
Malur Hosur Road 1
Abshot Layout     1
```

```
Name: location, Length: 1287, dtype: int64
```

```
len(location_stats[location_stats>10])
```

```
240
```

```
len(location_stats)
```

```
1287
```

```
len(location_stats[location_stats<=10])
```

```
1047
```

Any location having less than 10 data points should be tagged as "other" location. This way number of categories can be reduced by huge amount. Later on when we do one hot encoding, it will help us with having fewer dummy columns

```
location_stats_less_than_10 = location_stats[location_stats<=10]
```

```
location_stats_less_than_10
```

```
BTM 1st Stage      10
```

```

Gunjur Palya      10
Nagappa Reddy Layout    10
Sector 1 HSR Layout    10
Thyagaraja Nagar      10
..
Rajanna Layout      1
Subramanyanagar      1
Lakshmipura Vidyaanyapura   1
Malur Hosur Road     1
Abshot Layout        1
Name: location, Length: 1047, dtype: int64

```

```
len(df5.location.unique())
```

1287

```

df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(df5.location.unique())

```

241

```
df5.head(10)
```

	location	size	total_sqft	bath	price	bhk	\
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	
5	Whitefield	2 BHK	1170.0	2.0	38.00	2	
6	Old Airport Road	4 BHK	2732.0	4.0	204.00	4	
7	Rajaji Nagar	4 BHK	3300.0	4.0	600.00	4	
8	Marathahalli	3 BHK	1310.0	3.0	63.25	3	
9	other	6 Bedroom	1020.0	6.0	370.00	6	

```
price_per_sqft
```

```
0    3699.810606
1    4615.384615
2    4305.555556
3    6245.890861
4    4250.000000
5    3247.863248
6    7467.057101
7    18181.818182
8    4828.244275
9    36274.509804
```

normally square ft per bedroom is 300 (i.e. 2 bhk apartment is minimum 600 sqft

```
df5[df5.total_sqft/df5.bhk<300].head()
```

	location	size	total_sqft	bath	price	bhk	\
9	other	6 Bedroom	1020.0	6.0	370.0	6	
45		HSR Layout	8 Bedroom	600.0	9.0	200.0	8
58	Murugeshpalya	6 Bedroom	1407.0	4.0	150.0	6	
68	Devarachikkahalli	8 Bedroom	1350.0	7.0	85.0	8	
70	other	3 Bedroom	500.0	3.0	100.0	3	

```
price_per_sqft
```

```
9    36274.509804
45   33333.333333
58   10660.980810
68   6296.296296
70   20000.000000
```

Check above data points. We have 6 bhk apartment with 1020 sqft. Another one is 8 bhk and total sqft is 600. These are clear data errors that can be removed safely

```
df5.shape
```

```
(13200, 7)
```

```
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
```

```
df6.shape
```

```
(12456, 7)
```

```
df6.columns
```

```
Index(['location', 'size', 'total_sqft', 'bath', 'price', 'bhk',
       'price_per_sqft'],
      dtype='object')
```

```
plt.boxplot(df6['total_sqft'])
plt.show()
```

```
Q1 = np.percentile(df6['total_sqft'], 25.) # 25th percentile of the data of the given feature
```

```
Q3 = np.percentile(df6['total_sqft'], 75.) # 75th percentile of the data of the given feature
```

```
IQR = Q3-Q1 #Interquartile Range
```

```
ll = Q1 - (1.5*IQR)
```

```
ul = Q3 + (1.5*IQR)
```

```
upper_outliers = df6[df6['total_sqft'] > ul].index.tolist()
```

```
lower_outliers = df6[df6['total_sqft'] < ll].index.tolist()
```

```
bad_indices = list(set(upper_outliers + lower_outliers))
```

```
drop = True
```

```
if drop:
```

```
    df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
<ipython-input-51-c46bdd7d51e2>:11: SettingWithCopyWarning:
```

```
  A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas->

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
plt.boxplot(df6['bath'])
```

```
plt.show()
```

```
Q1 = np.percentile(df6['bath'], 25.) # 25th percentile of the data of the given feature
```

```
Q3 = np.percentile(df6['bath'], 75.) # 75th percentile of the data of the given feature
```

```
IQR = Q3-Q1 #Interquartile Range
```

```
ll = Q1 - (1.5*IQR)
```

```
ul = Q3 + (1.5*IQR)
```

```
upper_outliers = df6[df6['bath'] > ul].index.tolist()
```

```
lower_outliers = df6[df6['bath'] < ll].index.tolist()
```

```
bad_indices = list(set(upper_outliers + lower_outliers))
```

```
drop = True
```

```
if drop:
```

```
    df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
<ipython-input-54-cdb575bb4e89>:11: SettingWithCopyWarning:
```

```
  A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
plt.boxplot(df6['price'])
```

```
plt.show()
```

```
Q1 = np.percentile(df6['price'], 25.) # 25th percentile of the data of the given feature
```

```
Q3 = np.percentile(df6['price'], 75.) # 75th percentile of the data of the given feature
```

```
IQR = Q3-Q1 #Interquartile Range
```

```
ll = Q1 - (1.5*IQR)
```

```
ul = Q3 + (1.5*IQR)
```

```
upper_outliers = df6[df6['price'] > ul].index.tolist()
lower_outliers = df6[df6['price'] < ll].index.tolist()
bad_indices = list(set(upper_outliers + lower_outliers))
drop = True
if drop:
    df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

<ipython-input-56-e0f097c1f625>:11: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
plt.boxplot(df6['bhk'])
```

```
plt.show()
```

```
Q1 = np.percentile(df6['bhk'], 25.) # 25th percentile of the data of the given feature
```

```
Q3 = np.percentile(df6['bhk'], 75.) # 75th percentile of the data of the given feature
```

```
IQR = Q3-Q1 #Interquartile Range
```

```
ll = Q1 - (1.5*IQR)
```

```
ul = Q3 + (1.5*IQR)
```

```
upper_outliers = df6[df6['bhk'] > ul].index.tolist()
```

```
lower_outliers = df6[df6['bhk'] < ll].index.tolist()
```

```
bad_indices = list(set(upper_outliers + lower_outliers))
```

```
drop = True
```

```
if drop:
```

```
    df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

<ipython-input-58-c12c1120f543>:11: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
plt.boxplot(df6['price_per_sqft'])
```

```
plt.show()
```

```
Q1 = np.percentile(df6['price_per_sqft'], 25.) # 25th percentile of the data of the given feature
```

```
Q3 = np.percentile(df6['price_per_sqft'], 75.) # 75th percentile of the data of the given feature
```

```
IQR = Q3-Q1 #Interquartile Range
```

```
ll = Q1 - (1.5*IQR)
```

```
ul = Q3 + (1.5*IQR)
```

```
upper_outliers = df6[df6['price_per_sqft'] > ul].index.tolist()
```

```
lower_outliers = df6[df6['price_per_sqft'] < ll].index.tolist()
```

```
bad_indices = list(set(upper_outliers + lower_outliers))
```

```
drop = True
```

```
if drop:
```

```
    df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
<ipython-input-60-d349eb2f1f03>:11: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df6.drop(bad_indices, inplace = True, errors = 'ignore')
```

```
df6.shape
```

```
(10090, 7)
```

```
X = df6.drop(['price'],axis='columns')
```

```
X.head(3)
```

	location	size	total_sqft	bath	bhk	price_per_sqft
--	----------	------	------------	------	-----	----------------

0	Electronic City Phase II	2	BHK	1056.0	2.0	2	3699.810606
---	--------------------------	---	-----	--------	-----	---	-------------

```
X.shape
```

```
(10090, 6)
```

```
y = df6.price
```

```
y.head(3)
```

```
0      39.07
```

```
2      62.00
```

```
3      95.00
```

```
Name: price, dtype: float64
```

```
len(y)
```

```
10090
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=10)
```

```
X_train.shape
```

```
(8072, 6)
```

```
y_train.shape
```

```
(8072,)
```

```
X_test.shape
```

```
(2018, 6)
```

```
y_test.shape
```

```
(2018,)
```

<b>Lab Assignment No.</b>	11
<b>Title</b>	Data Visualization using matplotlib
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 11      (Group B)****Aim:**

Data Visualization using matplotlib

**Problem Statement:**

Analyzing Air Quality Index (AQI) Trends in a City

**Dataset:** "City\_Air\_Quality.csv"

**Description:**

The dataset contains information about air quality measurements in a specific city over a period of time. It includes attributes such as date, time, pollutant levels (e.g., PM2.5, PM10, CO), and the Air Quality Index (AQI) values. The goal is to use the matplotlib library to create visualizations that effectively represent the AQI trends and patterns for different pollutants in the city.

**Steps to Perform:**

1. Import the "City\_Air\_Quality.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Identify the relevant variables for visualizing AQI trends, such as date, pollutant levels, and AQI values.
4. Create line plots or time series plots to visualize the overall AQI trend over time.
5. Plot individual pollutant levels (e.g., PM2.5, PM10, CO) on separate line plots to visualize their trends over time.
6. Use bar plots or stacked bar plots to compare the AQI values across different dates or time periods.
7. Create box plots or violin plots to analyze the distribution of AQI values for different pollutant categories.
8. Use scatter plots or bubble charts to explore the relationship between AQI values and pollutant levels.
9. Customize the visualizations by adding labels, titles, legends, and appropriate color schemes.

**Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

**Software Requirement:**

Jupyter Notebook/Ubuntu

## Implementation:

```
import numpy as np import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.impute import SimpleImputer  
  
%matplotlib inline  
data = pd.read_csv("data.csv")  
print(data.index)  
RangeIndex(start=0, stop=49005, step=1)  
sns.set(style="ticks", rc = { 'figure.figsize':(20,15)})  
# Suppressing update warnings  
import warnings  
warnings.filterwarnings('ignore')  
Checking the dataset  
We can see that there are quite a number of NaNs in the dataset. To proceed with the EDA, we must handle these NaNs by either removing them or filling them. I will be doing both.  
# checking the original dataset  
print(data.isnull().sum())  
print(data.shape)  
data.info()  
stn_code           15764  
sampling_date      0  
State              0  
location            0  
Agency             16355  
Type               994  
so2                1312  
no2                858  
Rspm               2696  
Spm                28659  
location_monitoring_station 2537
```

```
pm2_5          49005
date           1
dtype: int64
(49005, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49005 entries, 0 to 49004
Data columns (total 13 columns):
 #  Column            Non-Null Count Dtype  
 --- 
 0  stn_code          33241 non-null   float64 
 1  sampling_date     49005 non-null   object  
 2  state              49005 non-null   object  
 3  location           49005 non-null   object  
 4  agency              32650 non-null   object  
 5  type                48011 non-null   object  
 6  so2                 47693 non-null   float64 
 7  no2                 48147 non-null   float64 
 8  rspm                46309 non-null   float64 
 9  spm                  20346 non-null   float64 
 10 location_monitoring_station 46468 non-null   object  
 11 pm2_5               0 non-null      float64 
 12 date                49004 non-null   object  
dtypes: float64(6), object(7)
memory usage: 4.9+ MB
```

### Cleaning the dataset

Removing NaNs Looking at the dataset head, we can conclude that the following columns:

1. stn\_code
1. agency
2. sampling\_date
3. location\_monitoring\_agency

do not add much to the dataset in terms of information that can't already be extracted from other columns. Therefore, we drop these columns.

Since date also has missing values, we will drop the rows containing these values as they're of little use as well.  
Cleaning values Since the geographical nomenclature has changed over time, we change it here as well to correspond to more accurate insights.

The type column

Currently, the type column has several names for the same type and therefore, it is better to clean it up and make it more uniform.

```
# Cleaning up the data
```

```
# cleaning up name changes
```

```
data.state = data.state.replace({'Uttaranchal':'Uttarakhand'})
```

```
data.state[data.location == "Jamshedpur"] = data.state[data.location ==
```

```
'Jamshedpur'].replace({"Bihar":"Jharkhand"})
```

```
#changing types to uniform format
```

```
types = {
```

```
    "Residential": "R",
```

```
    "Residential and others": "RO",
```

```
    "Residential, Rural and other Areas": "RRO",
```

```
    "Industrial Area": "I",
```

```
    "Industrial Areas": "I",
```

```
    "Industrial": "I",
```

```
    "Sensitive Area": "S",
```

```
    "Sensitive Areas": "S",
```

```
    "Sensitive": "S",
```

```
    np.nan: "RRO"
```

```
}
```

```
data.type = data.type.replace(types)
```

```
data.head()
```

```

state location type so2 no2 rspm spm pm2_5 date
0 Andhra Pradesh Hyderabad RRO 4.8 17.4 NaN NaN      NaN 1990-02-01
1 Andhra Pradesh Hyderabad I 3.1 7.0 NaN NaN      NaN 1990-02-01
2 Andhra Pradesh Hyderabad RRO 6.2 28.5 NaN NaN      NaN 1990-02-01
3 Andhra Pradesh Hyderabad RRO 6.3 14.7 NaN NaN      NaN 1990-03-01
4 Andhra Pradesh Hyderabad I 4.7 7.5 NaN NaN      NaN 1990-03-01

```

# defining columns of importance, which shall be used regularly

```
VALUE_COLS = ['so2', 'no2', 'rspm', 'spm', 'pm2_5']
```

Filling NaNs Since our pollutants column contain a lot of NaNs, we must fill them to have consistent data. If we drop the rows containing NaNs, we will be left with nothing.

I use the SimpleImputer from sklearn.imputer (v0.20.2) to fill the missing values in every column with the mean.

# invoking SimpleImputer to fill missing values

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
data[VALUE_COLS] = imputer.fit_transform(data[VALUE_COLS])
```

---

ValueError Traceback (most recent call last)

<ipython-input-16-7a53965e699d> in <cell line: 3>()

1 # invoking SimpleImputer to fill missing values

2 imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

---> 3 data[VALUE\_COLS] = imputer.fit\_transform(data[VALUE\_COLS])

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in \_\_setitem\_\_(self, key, value)

3966 self.\_setitem\_frame(key, value)

3967 elif isinstance(key, (Series, np.ndarray, list, Index)):

-> 3968 self.\_setitem\_array(key, value)

3969 elif isinstance(value, DataFrame):

3970 self.\_set\_item\_frame\_value(key, value)

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in \_setitem\_array(self, key, value)

4017

4018 elif isinstance(value, np.ndarray) and value.ndim == 2:

```
-> 4019      self._iset_not_inplace(key, value)
4020
4021      elif np.ndim(value) > 1:

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in _iset_not_inplace(self, key, value)
4044      if self.columns.is_unique:
4045          if np.shape(value)[-1] != len(key):
-> 4046              raise ValueError("Columns must be same length as key")
4047
4048      for i, col in enumerate(key):
```

ValueError: Columns must be same length as key

# checking to see if the dataset has any null values left over and the format

```
print(data.isnull().sum())
```

```
data.tail()
```

State	0
location	0
Type	0
so2	1312
no2	858
Rspm	2696
Spm	28659
pm2_5	49005
Date	1

```
dtype: int64
```

	state	location	type	so2	no2	rspm	spm	pm2_5	\
49000	Chandigarh	Chandigarh	RO	6.0	15.0	47.0	125.0	NaN	
49001	Chandigarh	Chandigarh	RO	NaN	12.0	54.0	161.0	NaN	
49002	Chandigarh	Chandigarh	RO	NaN	10.0	116.0	196.0	NaN	
49003	Chandigarh	Chandigarh	RO	NaN	9.0	38.0	154.0	NaN	
49004	Chandigarh	Chandigarh	RO	10.0	27.0	43.0	152.0	NaN	

```
date
49000 2005-03-23
49001 2005-03-25
49002 2005-03-28
49003 2005-03-30
49004      NaN
```

Plotting pollutant levels as yearly averages for states

```
# defining a function that plots SO2, NO2, RSPM and SPM yearly average levels for a given state
# since data is available monthly, it was resampled to a year and averaged to obtain yearly averages
# years for which no data was collected has not been imputed

def plot_for_state(state):

    fig, ax = plt.subplots(2,2, figsize=(20,12))

    fig.suptitle(state, size=20)

    state = aqi[aqi.state == state]

    state = state.reset_index().set_index('date')[VALUE_COLS].resample('Y').mean()

    state.so2.plot(legend=True, ax=ax[0][0], title="so2")
    ax[0][0].set_ylabel("so2 (\u00b5g/m3)")
    ax[0][0].set_xlabel("Year")

    state.no2.plot(legend=True, ax=ax[0][1], title="no2")
    ax[0][1].set_ylabel("no2 (\u00b5g/m3)")
    ax[0][1].set_xlabel("Year")

    state.rspm.plot(legend=True, ax=ax[1][0], title="rspm")
    ax[1][0].set_ylabel("RSPM (PM10 \u00b5g/m3)")
    ax[1][0].set_xlabel("Year")

    state.spm.plot(legend=True, ax=ax[1][1], title="spm")
    ax[1][1].set_ylabel("SPM (PM10 \u00b5g/m3)")
    ax[1][1].set_xlabel("Year")
```

```
plot_for_state("Uttar Pradesh")
```

Plotting Uttar Pradesh, we see that SO<sub>2</sub> levels have fallen in the state while NO<sub>2</sub> levels have risen. Information about RSPM and SPM can't be concluded since a lot of data is missing.

Plotting highest and lowest ranking states

```
# defining a function to find and plot the top 10 and bottom 10 states for a given indicator (defaults to SO2)
```

```
def top_and_bottom_10_states(indicator="so2"):
```

```
    fig, ax = plt.subplots(2,1, figsize=(20, 12))
```

```
    ind = data[[indicator, 'state']].groupby('state',
```

```
    as_index=False).median().sort_values(by=indicator, ascending=False)
```

```
    top10 = sns.barplot(x='state', y=indicator, data=ind[:10], ax=ax[0], color='red')
```

```
    top10.set_title("Top 10 states by {} (1991-2016)".format(indicator))
```

```
    top10.set_ylabel("so2 ( $\mu\text{g}/\text{m}^3$ )")
```

```
    top10.set_xlabel("State")
```

```
    bottom10 = sns.barplot(x='state', y=indicator, data=ind[-10:], ax=ax[1], color='green')
```

```
    bottom10.set_title("Bottom 10 states by {} (1991-2016)".format(indicator))
```

```
    bottom10.set_ylabel("so2 ( $\mu\text{g}/\text{m}^3$ )")
```

```
    bottom10.set_xlabel("State")
```

```
top_and_bottom_10_states("so2")
```

```
top_and_bottom_10_states("no2")
```

Plotting for SO<sub>2</sub>, we can see that the top state is Uttarakhand, while the bottom state is Meghalaya.

Plotting for NO<sub>2</sub>, we can see that the top state is West Bengal, while the bottom state is Mizoram.

Plotting the highest ever recorded levels

```
# defining a function to find the highest ever recorded levels for a given indicator (defaults to SO2) by state
```

```
# sidenote: mostly outliers
```

```
def highest_levels_recorded(indicator="so2"):
```

```
    plt.figure(figsize=(20,10))
```

```
    ind = data[[indicator, 'location', 'state', 'date']].groupby('state', as_index=False).max()
```

```
    highest = sns.barplot(x='state', y=indicator, data=ind)
```

```
    highest.set_title("Highest ever {} levels recorded by state".format(indicator))
```

```
plt.xticks(rotation=90)  
highest_levels_recorded("no2")  
highest_levels_recorded("rspm")
```

Plotting for NO2, we can see that Rajasthan recorded the highest ever NO2 level. Plotting for RSPM, we can see that Uttar Pradesh recorded the highest ever RSPM level.

Plotting yearly trends

```
# defining a function to plot the yearly trend values for a given indicator (defaults to SO2) and state (defaults to overall)
```

```
def yearly_trend(state="", indicator="so2", ):
```

```
    plt.figure(figsize=(20,12))  
    data['year'] = data.date.dt.year
```

```
    if state is "":
```

```
        year_wise = data[[indicator, 'year', 'state']].groupby('year', as_index=False).median()
```

```
        trend = sns.pointplot(x='year', y=indicator, data=year_wise)
```

```
        trend.set_title('Yearly trend of {}'.format(indicator))
```

```
    else:
```

```
        year_wise = data[[indicator, 'year', 'state']].groupby(['state','year']).median().loc[state].reset_index()
```

```
        trend = sns.pointplot(x='year', y=indicator, data=year_wise)
```

```
        trend.set_title('Yearly trend of {} for {}'.format(indicator, state))
```

```
yearly_trend()
```

```
yearly_trend("Bihar", "no2")
```

---

```
AttributeError           Traceback (most recent call last)
```

```
<ipython-input-42-e79267482a54> in <cell line: 1>()
```

```
----> 1 yearly_trend()
```

```
2 yearly_trend("Bihar", "no2")
```

```
<ipython-input-30-93f123e178ba> in yearly_trend(state, indicator)
```

```
2 def yearly_trend(state="", indicator="so2", ):
```

```
3     plt.figure(figsize=(20,12))
```

```
----> 4         data['year'] = data.date.dt.year
```

```
5   if state is "":
6       year_wise = data[[indicator, 'year', 'state']].groupby('year', as_index=False).median()
```

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in \_\_getattr\_\_(self, name)

```
5900     ):
5901         return self[name]
-> 5902         return object.__getattribute__(self, name)
5903
5904     def __setattr__(self, name: str, value) -> None:
```

/usr/local/lib/python3.10/dist-packages/pandas/core/Accessor.py in \_\_get\_\_(self, obj, cls)

```
180         # we're accessing the attribute of the class, i.e., Dataset.geo
181         return self._accessor
--> 182     accessor_obj = self._accessor(obj)
183     # Replace the property with the accessor object. Inspired by:
184     # https://www.pydanny.com/cached-property.html
```

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/Accessors.py in \_\_new\_\_(cls, data)

```
510         return PeriodProperties(data, orig)
511
--> 512     raise AttributeError("Can only use .dt accessor with datetimelike values")
```

AttributeError: Can only use .dt accessor with datetimelike values

<Figure size 2000x1200 with 0 Axes>

Plotting for SO2, we can see the yearly trend for sulphur dioxide levels in the country. Plotting for NO2 in West Bengal, we can see the yearly trend.

Plotting a heatmap for a particular indicator

```
# defining a function to plot a heatmap for yearly median average for a given indicator (defaults to SO2)
def indicator_by_state_and_year(indicator="so2"):
    plt.figure(figsize=(20, 20))
    hmap = sns.heatmap(
```

```
data=data.pivot_table(values=indicator, index='state', columns='year', aggfunc='median', margins=True),
    annot=True, linewidths=.5, cbar=True, square=True, cmap='inferno', cbar_kws={'label': "Annual
Average"})
```

```
hmap.set_title("{} by state and year".format(indicator))
indicator_by_state_and_year('no2')
```

```
-----  
KeyError           Traceback (most recent call last)  
<ipython-input-35-39c9f3640fe4> in <cell line: 1>()  
----> 1 indicator_by_state_and_year('no2')
```

```
<ipython-input-34-3c4f9130ffd5> in indicator_by_state_and_year(indicator)
      3     plt.figure(figsize=(20, 20))
      4     hmap = sns.heatmap(
----> 5     data=data.pivot_table(values=indicator, index='state', columns='year', aggfunc='median',
margins=True),
      6             annot=True, linewidths=.5, cbar=True, square=True, cmap='inferno', cbar_kws={'label':
"Annual Average"})
```

```
7
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in pivot_table(self, values, index, columns,
aggfunc, fill_value, margins, dropna, margins_name, observed, sort)
```

```
8729     from pandas.core.reshape.pivot import pivot_table
8730
-> 8731     return pivot_table(
8732         self,
8733         values=values,
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/reshape/pivot.py in pivot_table(data, values, index, columns,
aggfunc, fill_value, margins, dropna, margins_name, observed, sort)
```

```
95     return table.__finalize__(data, method="pivot_table")
```

96

```
--> 97     table = __internal_pivot_table(  
98         data,  
99         values,
```

/usr/local/lib/python3.10/dist-packages/pandas/core/reshape/pivot.py in \_\_internal\_pivot\_table(data, values, index, columns, aggfunc, fill\_value, margins, dropna, margins\_name, observed, sort)

164 values = list(values)

165

```
--> 166     grouped = data.groupby(keys, observed=observed, sort=sort)
```

167 msg = (

168 "pivot\_table dropped a column because it failed to aggregate. This behavior "

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in groupby(self, by, axis, level, as\_index, sort, group\_keys, squeeze, observed, dropna)

8400 axis = self.\_get\_axis\_number(axis)

8401

```
-> 8402     return DataFrameGroupBy(
```

8403 obj=self,

8404 keys=by,

/usr/local/lib/python3.10/dist-packages/pandas/core/groupby/groupby.py in \_\_init\_\_(self, obj, keys, axis, level, grouper, exclusions, selection, as\_index, sort, group\_keys, squeeze, observed, mutated, dropna)

963 from pandas.core.groupby.grouper import get\_grouper

964

```
--> 965     grouper, exclusions, obj = get_grouper(
```

966 obj,

967 keys,

/usr/local/lib/python3.10/dist-packages/pandas/core/groupby/grouper.py in get\_grouper(obj, key, axis, level, sort, observed, mutated, validate, dropna)

```
886      in_axis, level, gpr = False, gpr, None
887      else:
--> 888      raise KeyError(gpr)
889      elif isinstance(gpr, Grouper) and gpr.key is not None:
890          # Add key to exclusions
```

KeyError: 'year'

<Figure size 2000x2000 with 0 Axes>

Plotting pollutant average by type

# defining a function to plot pollutant averages by type for a given indicator

```
def type_avg(indicator=""):
```

```
    type_avg = data[VALUE_COLS + ['type', 'date']].groupby("type").mean()
    if indicator is not "":
        t = type_avg[indicator].plot(kind='bar')
        plt.xticks(rotation = 0)
        plt.title("Pollutant average by type for {}".format(indicator))
    else:
        t = type_avg.plot(kind='bar')
        plt.xticks(rotation = 0)
        plt.title("Pollutant average by type")
```

```
type_avg('so2')
```

Plotting pollutant averages by locations/state

# defining a function to plot pollutant averages for a given indicator (defaults to SO2) by locations in a given state

```
def location_avgs(state, indicator="so2"):
```

```
    locs = data[VALUE_COLS + ['state', 'location', 'date']].groupby(['state', 'location']).mean()
    state_avgs = locs.loc[state].reset_index()
    sns.barplot(x='location', y=indicator, data=state_avgs)
    plt.title("Location-wise average for {} in {}".format(indicator, state))
    plt.xticks(rotation = 90)
```

```
location_avgs("Bihar", "no2")
```

<b>Lab Assignment No.</b>	12
<b>Title</b>	Analyzing Sales Performance by Region in a Retail Company
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 12      (Group B)****Aim:**

Data Aggregation

**Problem Statement:**

Analyzing Sales Performance by Region in a Retail Company

**Dataset:** "Retail\_Sales\_Data.csv"

**Description:**

The dataset contains information about sales transactions in a retail company. It includes attributes such as transaction date, product category, quantity sold, and sales amount. The goal is to perform data aggregation to analyze the sales performance by region and identify the top-performing regions.

**Steps to Perform:**

1. Import the "Retail\_Sales\_Data.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Identify the relevant variables for aggregating sales data, such as region, sales amount, and product category.
4. Group the sales data by region and calculate the total sales amount for each region.
5. Create bar plots or pie charts to visualize the sales distribution by region.
6. Identify the top-performing regions based on the highest sales amount.
7. Group the sales data by region and product category to calculate the total sales amount for each combination.
8. Create stacked bar plots or grouped bar plots to compare the sales amounts across different regions and product categories.

**Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

**Software Requirement:**

Jupyter Notebook/Ubuntu

**Implementation:**

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

Data Aggregation is important for deriving granular insights about individual customers and for better understanding their perception and expectations regarding the product.

Regardless of the size and type, every business organization needs valuable data and insights to combat the day-to-day challenges of the competitive market. If a business wants to thrive in the market, then it must understand its target audience and customer preferences, and in this, big data plays a vital role.

What is Data Aggregation?

### About Dataset

dataset contains shopping information from 10 different shopping malls between 2021 and 2023. We have gathered data from various age groups and genders to provide a comprehensive view of shopping habits in Istanbul. The dataset includes essential information such as invoice numbers, customer IDs, age, gender, payment methods, product categories, quantity, price, order dates, and shopping mall locations.

Attribute Information:

invoice\_no: Invoice number. Nominal. A combination of the letter 'T' and a 6-digit integer uniquely assigned to each operation.

customer\_id: Customer number. Nominal. A combination of the letter 'C' and a 6-digit integer uniquely assigned to each operation.

gender: String variable of the customer's gender.

age: Positive Integer variable of the customers age.

category: String variable of the category of the purchased product.

quantity: The quantities of each product (item) per transaction. Numeric.

price: Unit price. Numeric. Product price per unit in Turkish Liras (TL).

payment\_method: String variable of the payment method (cash, credit card or debit card) used for the transaction.

invoice\_date: Invoice date. The day when a transaction was generated.

shopping\_mall: String variable of the name of the shopping mall where the transaction was made.  
 # dataset source: <https://www.kaggle.com/datasets/mehmetahiraslan/customer-shopping-dataset>

Metrocity	4193	4193	4193	4193	4193	4193
-----------	------	------	------	------	------	------

```
#df = pd.read_csv("/content/customer_shopping_data.csv")
```

```
df= pd.read_csv("/content/customer_shopping_data.csv")
```

```
df.head()
```

	invoice_no	customer_id	gender	age	category	quantity	price
0	I138884	C241288	Female	28	Clothing	5.0	1500.40
1	I317333	C111565	Male	21	Shoes	3.0	1800.51
2	I127801	C266599	Male	20	Clothing	1.0	300.08
3	I173702	C988172	Female	66	Shoes	5.0	3000.85
4	I337046	C189076	Female	53	Books	4.0	60.60

payment\_method invoice\_date shopping\_mall

	payment_method	invoice_date	shopping_mall
0	Credit Card	5/8/2022	Kanyon
1	Debit Card	12/12/2021	Forum Istanbul
2	Cash	9/11/2021	Metrocity
3	Credit Card	16/05/2021	Metropol AVM
4	Cash	24/10/2021	Kanyon

# To check the count of records grouped by region/branch of the mall

```
df.groupby("shopping_mall").count()
```

shopping_mall	invoice_no	customer_id	gender	age	category	quantity
Cevahir AVM	1349	1349	1349	1349	1349	1349
Emaar Square Mall	1341	1341	1341	1341	1341	1341
Forum Istanbul	1343	1343	1343	1343	1343	1343
Istinye Park	2709	2709	2709	2709	2709	2709
Kanyon	5481	5481	5481	5481	5481	5481
Mall of Istanbul	5588	5588	5588	5588	5588	5588

Metropol AVM	2856	2856	2856	2856	2856	2856
Viaport Outlet	1389	1389	1389	1389	1389	1389
Zorlu Center	1392	1392	1392	1392	1392	1392

price payment\_method invoice\_date

shopping\_mall

Cevahir AVM	1349	1349	1349
Emaar Square Mall	1341	1341	1341
Forum Istanbul	1343	1343	1343
Istinye Park	2709	2709	2709
Kanyon	5481	5481	5481
Mall of Istanbul	5588	5588	5588
Metrocity	4193	4193	4193
Metropol AVM	2856	2856	2856
Viaport Outlet	1389	1389	1389
Zorlu Center	1392	1392	1392

# To check the count of records grouped by the product categories

df.groupby("category").count()

category	invoice_no	customer_id	gender	age	quantity	price	\
Books	1397	1397	1397	1397	1397	1397	
Clothin	1	1	1	1	0	0	
Clothing	9433	9433	9433	9433	9433	9433	
Cosmetics	4224	4224	4224	4224	4224	4224	
Food & Beverage	4158	4158	4158	4158	4158	4158	
Shoes	2773	2773	2773	2773	2773	2773	
Souvenir	1402	1402	1402	1402	1402	1402	
Technology	1435	1435	1435	1435	1435	1435	

Toys	2819	2819	2819	2819	2819	2819
------	------	------	------	------	------	------

payment\_method invoice\_date shopping\_mall

category

Books	1397	1397	1397
Clothin	0	0	0
Clothing	9433	9433	9433
Cosmetics	4224	4224	4224
Food & Beverage	4158	4158	4158
Shoes	2773	2773	2773
Souvenir	1402	1402	1402
Technology	1435	1435	1435
Toys	2819	2819	2819

# total sales for each mall branch

```
branch_sales = df.groupby("shopping_mall").sum()
```

<ipython-input-13-64840580634c>:3: FutureWarning: The default value of numeric\_only in  
DataFrameGroupBy.sum is deprecated. In a future version, numeric\_only will default to False. Either specify  
numeric\_only or select only columns which should be valid for the function.

```
branch_sales = df.groupby("shopping_mall").sum()
```

# total sales for each category of product

```
category_sales = df.groupby("category").sum()
```

<ipython-input-14-732f2a6af039>:3: FutureWarning: The default value of numeric\_only in  
DataFrameGroupBy.sum is deprecated. In a future version, numeric\_only will default to False. Either specify  
numeric\_only or select only columns which should be valid for the function.

```
category_sales = df.groupby("category").sum()
```

In the above two cells, the sum method will return sums for all numeric values. For some attributes such as age, this sum is not relevant.

#to get the top performing branches

```
branch_sales.sort_values(by = "price", ascending = False)
```

	age	quantity	price
shopping_mall			
Mall of Istanbul	243751	16680.0	3874873.68
Kanyon	237767	16464.0	3774006.38
Metrocity	183003	12585.0	2799049.70
Metropol AVM	123899	8530.0	1886384.39
Istinye Park	118686	8202.0	1874608.87
Viaport Outlet	59666	4107.0	989716.52
Zorlu Center	60844	4181.0	983379.89
Emaar Square Mall	58286	4008.0	927215.95
Cevahir AVM	57069	4059.0	913555.36
Forum Istanbul	58716	4063.0	895712.68

# to get the top selling categories

```
category_sales.sort_values(by = "price", ascending = False)
```

	age	quantity	price
category			
Clothing	1497054	103558	31075684.64
Shoes	436027	30217	18135336.89
Technology	216669	15021	15772050.00
Cosmetics	657937	45465	1848606.90
Toys	437032	30321	1086704.64
Food & Beverage	640605	44277	231568.71
Books	216882	14982	226977.30

Souvenir 216922 14871 174436.83

# to get total sales for each combination of branch and product\_category

```
combined_branch_category_sales = df.groupby(["shopping_mall", "category"]).sum()
```

<ipython-input-16-994273aad95b>:3: FutureWarning: The default value of numeric\_only in  
DataFrameGroupBy.sum is deprecated. In a future version, numeric\_only will default to False. Either specify  
numeric\_only or select only columns which should be valid for the function.

```
combined_branch_category_sales = df.groupby(["shopping_mall", "category"]).sum()
```

```
combined_branch_category_sales
```

# pie chart for sales by branch

```
plt.pie(branch_sales["price"], labels = branch_sales.index)
```

```
plt.show()
```

# pie chart for sales by product category

```
plt.pie(category_sales["price"], labels = category_sales.index)
```

```
plt.show()
```

```
combined_pivot = df.pivot_table(index="shopping_mall", columns="category", values="price", aggfunc="sum")
```

# grouped bar chart for sales of different categories at different branches

```
combined_pivot.plot(kind="bar", figsize=(10, 6))
```

```
plt.show()
```

<b>Lab Assignment No.</b>	13
<b>Title</b>	Mini Project
<b>Roll No.</b>	
<b>Class</b>	BE
<b>Date of Completion</b>	
<b>Subject</b>	Computer Laboratory-I
<b>Assessment Marks</b>	
<b>Assessor's Sign</b>	

**EXPERIMENT NO. 13      (Group C)****Aim:**

To design and develop a mini-project in groups (5 students) by applying concepts of Machine Learning, Data Modeling, and Data Visualization on real-world problem statements, demonstrating skills beyond syllabus requirements.

**Concept:**

Mini projects bridge the gap between theory and practice. By working collaboratively, students learn how to:

- Apply ML and AI concepts to real-world datasets.
- Use data preprocessing, feature engineering, and model building techniques.
- Evaluate models using appropriate performance metrics.
- Visualize outcomes for better interpretation and decision-making.
- Strengthen teamwork, project management, and presentation skills.

**Theory:**

1. Machine Learning in Real-World Problems
  - ML helps to extract patterns and predictions from large datasets.
  - Techniques like classification, regression, clustering, and NLP (Natural Language Processing) can be applied depending on the project domain.
2. Data Modeling
  - Data collection from sources like [data.gov.in](http://data.gov.in) or open datasets.
  - Cleaning and preprocessing of data to remove noise and inconsistencies.
  - Feature selection and engineering for accurate predictions.
  - Building models using algorithms such as Decision Trees, Random Forest, Logistic Regression, KNN, or Neural Networks.
3. Visualization
  - Data visualization tools (Matplotlib, Seaborn, Plotly, PowerBI, Tableau) help in interpreting data trends.
  - Interactive dashboards for stakeholders to make informed decisions.
4. Problem Statement Examples
  - Happiness Index for Schools → Combining academic, mental health, extracurricular, and well-being data to generate a score.
  - Automated Animal Identification → Using Computer Vision (CNNs) for animal species recognition.
  - Sentiment Analysis on Govt. Policies → NLP models to analyze positive/negative impact of policies.
  - Flood Prone Roads Identification → Using rainfall, terrain, and historical flood data with GIS integration.

- Missing Bridges for Connectivity → Graph theory and geospatial analysis to suggest infrastructure improvements.

## Instructions to Students

Prepare:

- Project Report with aim, objectives, methodology, results, and conclusion.
- Code documentation (well-commented Python/Jupyter Notebook).
- Presentation/Poster for viva.

## Topics For Mini Projects:

### 1. Happiness Index for Schools

**Data Source:** Browse education and well-being data on [data.gov.in](https://data.gov.in) — look for school-level reports, attendance statistics, student surveys, or well-being indicators.

### 2. Sentiment Analysis of Government Policies

**Datasets:** Use general sentiment data like the Twitter Sentiment Dataset (tweets + labels) on Kaggle Kaggle. For Indic context, explore L3Cube-MahaSent-MD for Marathi domain-specific sentiment analysis

### 3. Fake News Detection

**Datasets:** Widely available on Kaggle: “Fake News Detection” datasets (news text + real/fake labels). Search Kaggle for current versions.

### 4. Crime Pattern Prediction in Cities

**Datasets:** Look into state police portals or data.gov.in for open crime statistics datasets; e.g., NCRB crime data.

### 5. Identifying Missing Bridges / Roads for Connectivity

**Data Source:** Use GIS data from OpenStreetMap and government infrastructure datasets on data.gov.in.

### 6. Flood-Prone Road Identification

**Datasets:** National or state-level rainfall and flood data from data.gov.in, plus elevation/terrain data, and road network data via OpenStreetMap.

### 7. Air Quality Prediction (AQI Forecasting)

**Data Source:** City-level historical air quality data available on data.gov.in or through central or local pollution control boards (like CPCB).

### 8. Forest Fire Risk Prediction

**Datasets:** Satellite datasets from ISRO’s Bhuvan platform; forest fire datasets available on NASA FIRMS (external).

### 9. Drought Prediction using Rainfall & Soil Data

**Data Source:** India Meteorological Department (IMD) rainfall datasets; agriculture and soil moisture data via data.gov.in.

### 10. Waste Management Optimization

**Datasets:** Smart city portals or municipal waste management systems—check state data portals for IoT-generated waste bin data.

**11. Mental Health/Stress Detection via Social Media**

**Datasets:** Social media text (Twitter/X) datasets—e.g., Kaggle sentiment datasets; create your own scraped dataset with ethical considerations.

**12. Disease Outbreak Prediction (e.g., Dengue)**

**Data Source:** Health ministry open datasets (case counts), and weather data from data.gov.in for modeling.

**13. Smart Diet Recommendation System**

**Datasets:** Nutrition datasets from government health portals; user lifestyle data can be simulated.

**14. Sleep Quality Prediction from Wearables**

**Datasets:** Public wearable data (if any available), else generate synthetic data or use open datasets like the MHealth (monitoring health) dataset.

**15. Depression Detection via Sentiment/Text**

**Datasets:** General sentiment datasets (Kaggle) or health-related text datasets; L3Cube Marathi dataset may help if focusing on Marathi content

**16. Automated Traffic Congestion Prediction**

**Datasets:** City traffic data from Municipal Transport Authority; Google Traffic APIs, or city smart city portals.

**17. Accident-Prone Zone Identification**

**Datasets:** Accident data from transport departments or traffic police; look for open datasets in state portals or data.gov.in.

**18. Smart Parking Prediction System**

**Datasets:** Municipal smart parking data or apps; Kaggle may have parking datasets globally.

**19. Public Transport Demand Forecasting**

**Datasets:** Transport authorities' usage logs, ticketing data; city/state transport data portals.

**20. Energy Consumption Forecasting in Smart Homes**

**Datasets:** Open thermal/electricity usage datasets (e.g., UCI Energy datasets) or Indian power consumption stats from data.gov.in.

**Conclusion**

By completing this mini-project we gain hands-on experience in applying ML & data visualization concepts to real-world problems.