Second Semester MCA Degree Examination, June/July 2025

Machine Learning & Data Analytics using Python

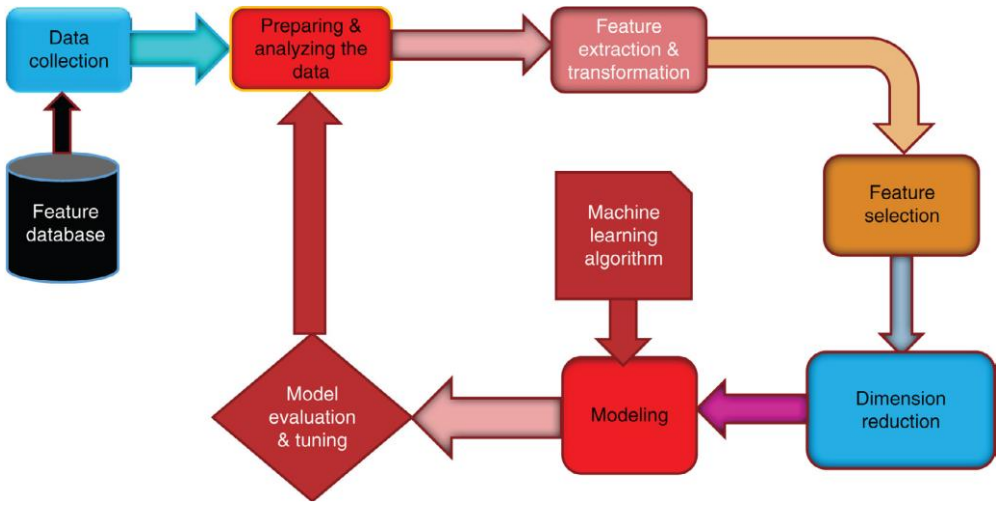
Time: 3 hrs. Max. Marks: 100

Q.1 a.Define Machine Learning. Describe its types with examples.

Introduction to Machine Learning:

Machine learning is about extracting knowledge from data. It is a research field at the intersection of statistics, artificial intelligence, and computer science and is also known as predictive analytics or statistical learning. To be intelligent, a system that is in a changing environment should have the ability to learn. Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer pro- gram to optimize the parameters of the model using the training data or past experience.

A machine learning framework is mainly composed of elements associated with data retrieval and extraction, preparation, modeling, evaluation, and deployment.



The most successful kinds of machine learning algorithms are those that automate decision-making processes by generalizing from known examples. In this setting, which is known as *supervised learning*, the user provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired out- put given an input. In particular, the algorithm is able to create an output for an input it has never seen before without any help from a human. Going back to our example of spam classification, using machine learning, the user provides the algorithm with a large number of emails (which are the input), together with information about whether any of these emails are spam (which is the desired output). Given a new email, the algorithm will then produce a prediction as to whether the new email is spam.

Machine learning algorithms that learn from input/output pairs are called supervised

learning algorithms because a "teacher" provides supervision to the algorithms in the form of the desired outputs for each example that they learn from. While creating a dataset of inputs and outputs is often a laborious manual process, supervised learning algorithms are well understood and their performance is easy to measure. If your application can be formulated as a supervised learning problem, and you are able to create a dataset that includes the desired outcome, machine learning will likely be able to solve your problem.

Examples of supervised machine learning tasks include:

Identifying the zip code from handwritten digits on an envelope

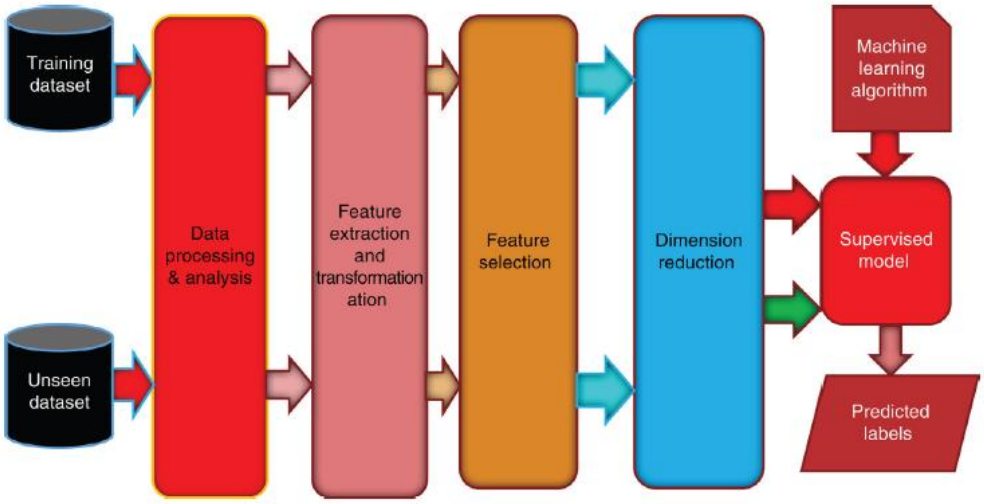
Here the input is a scan of the handwriting, and the desired output is the actual digits in the zip code. To create a dataset for building a machine learning model, you need to collect many envelopes. Then you can read the zip codes yourself and store the digits as your desired outcomes.

Determining whether a tumor is benign based on a medical image

Here the input is the image, and the output is whether the tumor is benign. To create a dataset for building a model, you need a database of medical images. You also need an expert opinion, so a doctor needs to look at all of the images and decide which tumors are benign and which are not. It might even be necessary to do additional diagnosis beyond the content of the image to determine whether the tumor in the image is cancerous or not.

Detecting fraudulent activity in credit card transactions

Here the input is a record of the credit card transaction, and the output is whether it is likely to be fraudulent or not. Assuming that you are the entity dis- tributing the credit cards, collecting a dataset means storing all transactions and recording if a user reports any transaction as fraudulent.



Supervised machine learning framework.

Unsupervised algorithms are the other type of algorithm that we will cover in this book. In unsupervised learning, only the input data is known, and no known output data is given to the algorithm. While there are many successful applications of these methods, they are usually harder to understand and evaluate.

Examples of unsupervised learning include:

Identifying topics in a set of blog posts

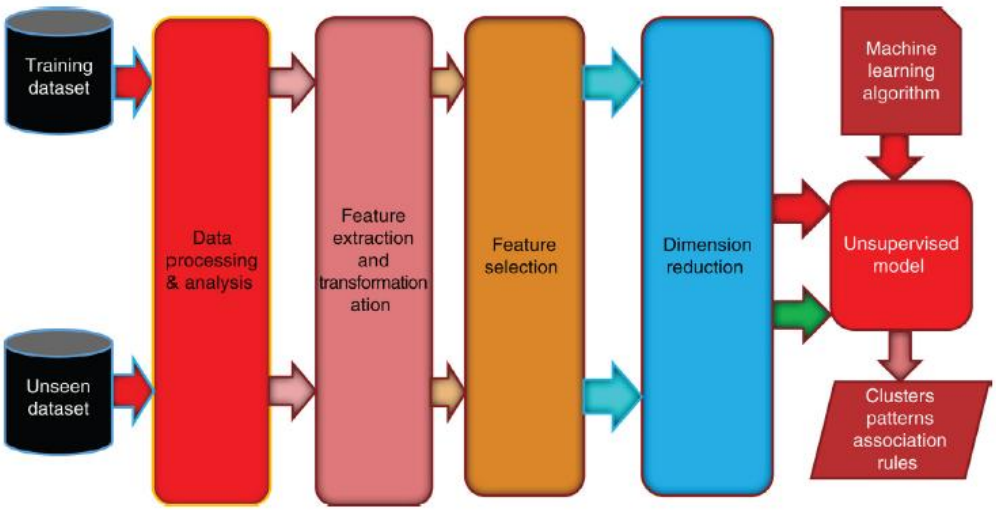
If you have a large collection of text data, you might want to summarize it and find prevalent themes in it. You might not know beforehand what these topics are, or how many topics there might be. Therefore, there are no known outputs.

Segmenting customers into groups with similar preferences

Given a set of customer records, you might want to identify which customers are similar, and whether there are groups of customers with similar preferences. For a shopping site, these might be "parents," "bookworms," or "gamers." Because you don't know in advance what these groups might be, or even how many there are, you have no known outputs.

Detecting abnormal access patterns to a website

To identify abuse or bugs, it is often helpful to find access patterns that are differ- ent from the norm. Each abnormal pattern might be very different, and you might not have any recorded instances of abnormal behavior. Because in this example you only observe traffic, and you don't know what constitutes normal and abnormal behavior, this is an unsupervised problem.



Unsupervised machine learning framework.

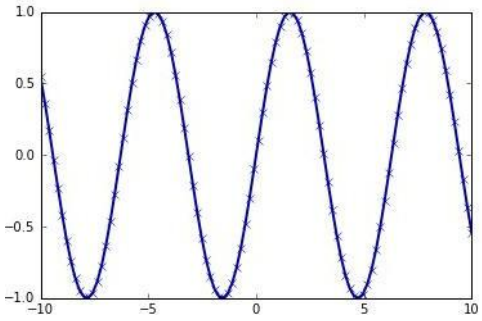
b. Illustrate the data visualization using matplotlib with three different charts.

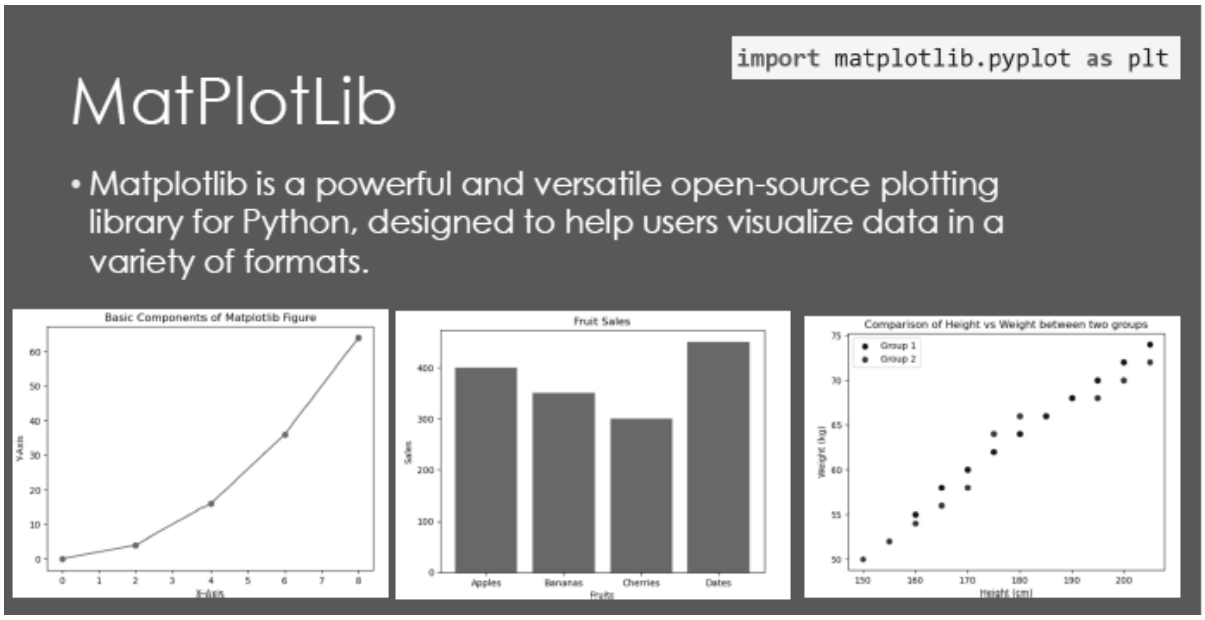
matplotlib is the primary scientific plotting library in Python. It provides functions for making publication-quality visualizations such as line charts, histograms, scatter plots, and so on. Visualizing your data and different aspects of your analysis can give you important insights, and we will be using matplotlib for all our visualizations. When working inside the Jupyter Notebook, you can show figures directly in the browser by using the %matplotlib notebook and %matplotlib inline commands. We recommend using %matplotlib notebook, which provides an interactive envi- ronment (though we are using

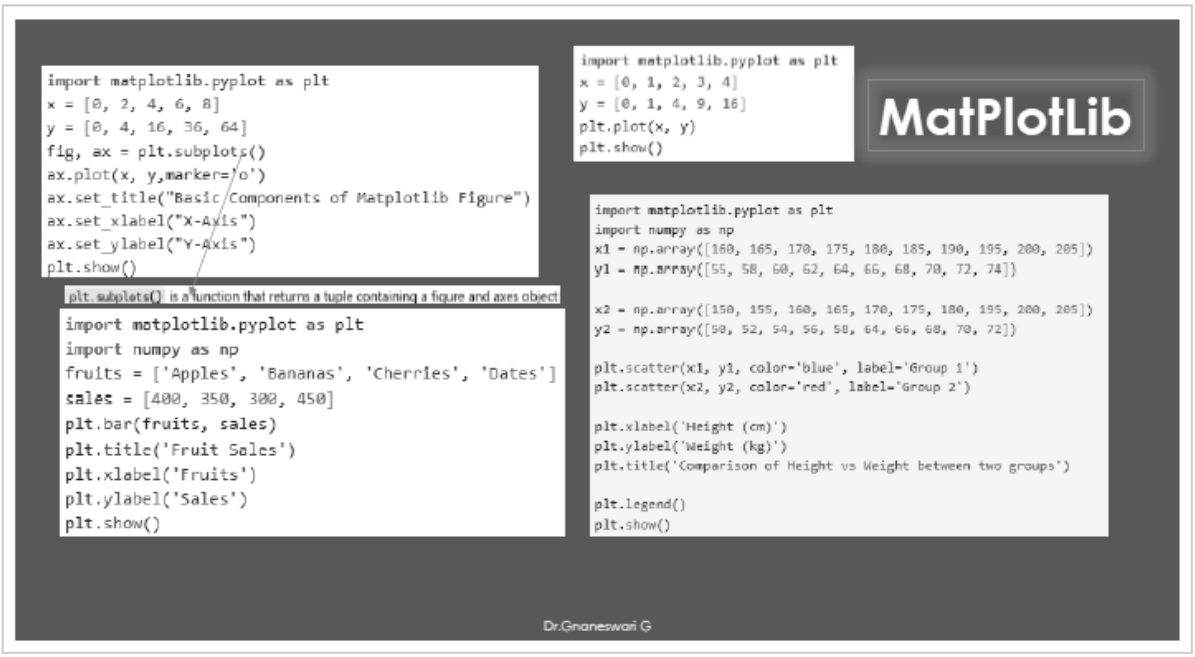
%matplotlib inline to produce this book). For example, this code produces the plot in Figure 1-1:

In[6]:

```
%matplotlib inline

import matplotlib.pyplot as plt


# Generate a sequence of numbers from -10 to 10 with 100 steps in between

x = np.linspace(-10, 10, 100)

# Create a second array using sine

y = np.sin(x)

# The plot function makes a line chart of one array against another

plt.plot(x, y, marker="x")
```







Q.2 a. Demonstrate the use of NumPy and Pandas for reading a csv file and manipulate the data with appropriate examples.

NumPy

NumPy is one of the fundamental packages for scientific computing in Python. It contains functionality for multidimensional arrays, high-level mathematical func- tions such as linear algebra operations and the Fourier transform, and pseudorandom number generators.

In scikit-learn, the NumPy array is the fundamental data structure. scikit-learn takes in data in the form of NumPy arrays. Any data you're using will have to be con- verted to a NumPy array. The core functionality of NumPy is the ndarray class, a multidimensional (*n*-dimensional) array. All elements of the array must be of the same type. A NumPy array looks like this:

In[2]:

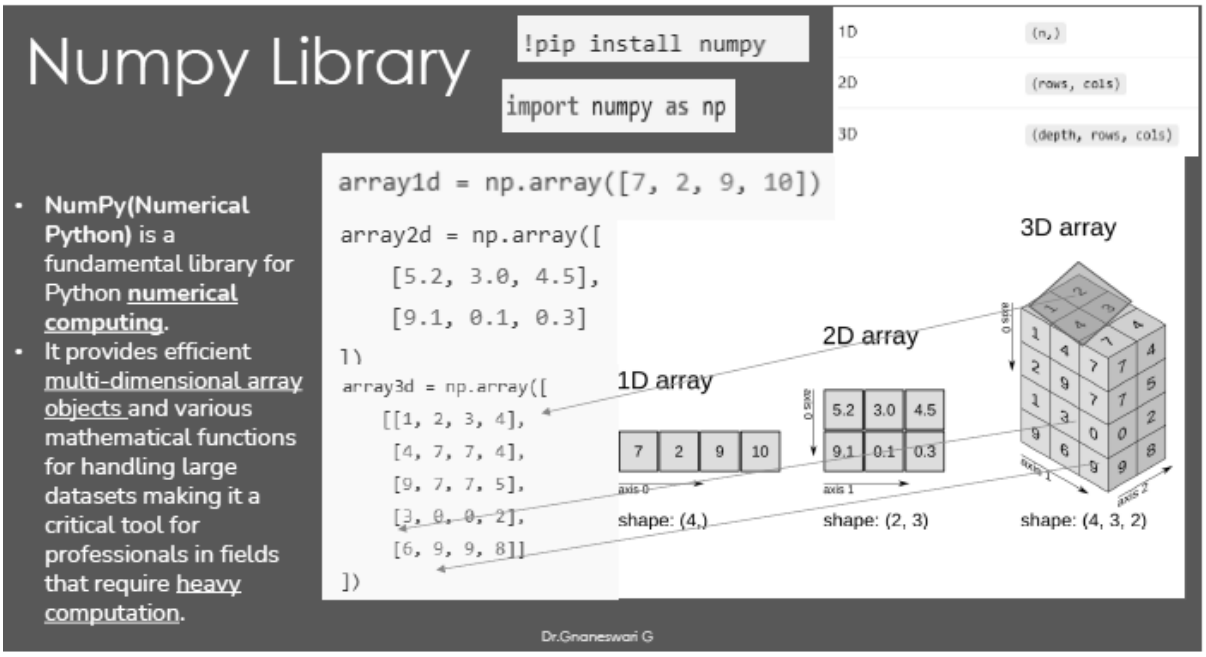
```
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])

print("x:\n{}".format(x))
```

Out[2]:

x:

[[1 2 3]

[4 5 6]]



b. Explain the steps involved in Data 'Preprocessing. Mention the techniques used for handling missing values and outliers.

Data Preprocessing Techniques:

1. Data Cleaning:

When it comes to creating a Machine Learning model, data preprocessing is the first step marking the initiation of the process. Typically, real-world data is

incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. This is where data preprocessing enters the scenario – it helps to clean, format, and organize the raw data, thereby making it ready-to-go for Machine Learning models. Let's explore various steps of data preprocessing in machine learning.

Acquiring the dataset is the first step in data preprocessing in machine learning. To build and develop Machine Learning models, you must first acquire the relevant dataset. This dataset will be comprised of data gathered from multiple and disparate sources which are then combined in a proper format to form a dataset. Dataset formats differ according to use cases. For instance, a business dataset will be entirely different from a medical dataset. While a business dataset will contain relevant industry and business data, a medical dataset will include healthcare-related data.

Import all the crucial libraries

The predefined Python libraries can perform specific data preprocessing jobs. Importing all the crucial libraries is the second step in data preprocessing in machine learning.

In [1]:

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

Import the dataset

In this step, you need to import the dataset/s that you have gathered for the ML project at hand. Importing the dataset is one of the important steps in data preprocessing in machine learning.

In [2]:

```
dataset = pd.read_csv(r"../input/preprocessingdataset/Data.csv")
```

In [3]:

```
df = pd.DataFrame(dataset)
```

In [4]:

```
df
```

Out[4]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [5]:

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [6]:

```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

In [7]:

```
print(y)
```

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

a. Identifying and handling the missing values

In data preprocessing, it is pivotal to identify and correctly handle the missing values, failing to do this, you might draw inaccurate and faulty conclusions and inferences from the data. Needless to say, this will hamper your ML project.

some typical reasons why data is missing:

A. User forgot to fill in a field.

B. Data was lost while transferring manually from a legacy database.

C. There was a programming error.

D. Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

Basically, there are two ways to handle missing data:

Deleting a particular row – In this method, you remove a specific row that has a null value for a feature or a particular column where more than 75% of the values are missing. However, this method is not 100% efficient, and it is recommended that you use it only when the dataset has adequate samples. You must ensure that after deleting the data, there remains no addition of bias. Calculating the mean – This method is useful for features having numeric data like age, salary, year, etc. Here, you can calculate the mean, median, or mode of a particular feature or column or row that contains a missing value and replace the result for the missing value. This method can add variance to the dataset, and any loss of data can be efficiently negated. Hence, it yields better results compared to the first method (omission of rows/columns). Another way of approximation is through the deviation of neighbouring values. However, this works best for linear data.

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

Solution 1 : Dropna

In [9]:

```
df1 = df.copy()
```

In [10]:

```
# summarize the shape of the raw data
print("Before:",df1.shape)
# drop rows with missing values
```

```
df1.dropna(inplace=True)

# summarize the shape of the data with missing rows removed
print("After:",df1.shape)
```

Before: (10, 4)
After: (8, 4)

Solution 2 : Fillna

In [11]:

```
df2 = df.copy()
```

In [12]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [13]:

```
# fill missing values with mean column values
df2.fillna(df2.mean(), inplace=True)
# count the number of NaN values in each column
print(df2.isnull().sum())

df2
```

Country 0
Age 0
Salary 0
Purchased 0
dtype: int64

Out[13]:

	Country	Age	Salary	Purchased
0	France	44.000000	72000.000000	No
1	Spain	27.000000	48000.000000	Yes
2	Germany	30.000000	54000.000000	No
3	Spain	38.000000	61000.000000	No
4	Germany	40.000000	63777.777778	Yes
5	France	35.000000	58000.000000	Yes
6	Spain	38.777778	52000.000000	No

	Country	Age	Salary	Purchased
7	France	48.000000	79000.000000	Yes
8	Germany	50.000000	83000.000000	No
9	France	37.000000	67000.000000	Yes

Solution 3 : Scikit-Learn

In [14]:

```
X
```

Out[14]:

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [15]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

In [16]:

```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

b. Encoding the categorical data

Categorical data refers to the information that has specific categories within the dataset. In the dataset cited above, there are two categorical variables – country and purchased.

Machine Learning models are primarily based on mathematical equations. Thus, you can intuitively understand that keeping the categorical data in the equation will cause certain issues since you would only need numbers in the equations.

Solution 1 : ColumnTransformer

In [17]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainde
r='passthrough')
X = np.array(ct.fit_transform(X))
```

In [18]:

```
df
```

Out[18]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

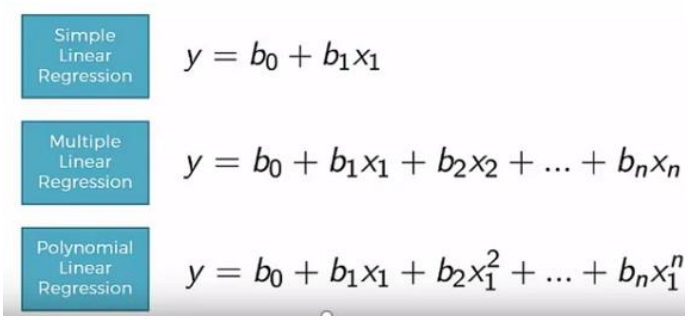
In [19]:

```
print(X)
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

Q3 a. Explain Linear and Polynomial Regression. Provide code examples using skleam.

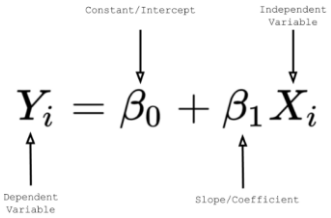
Linear and Polynomial Regression

Regression is a method of estimating a relationship from given data to depict nature of data set. This relationship can then be used for the forecasting future values or for computing if there exists a relationship among the various variables.

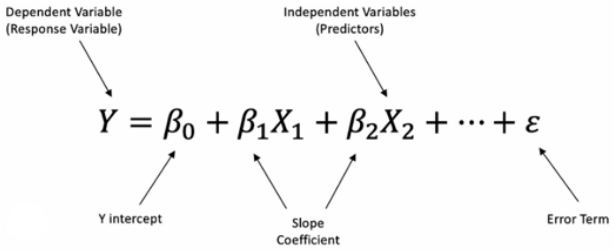


Linear Regression

In Linear Regression, there exists a linear relationship between the variables. The linear relationship amongst one response variable(dependent variable) and one regressor variable(independent variable) called as Simple linear regression and between one response variable and multiple regressor variable is called as Multivariate linear regression.



Simple Linear Regression



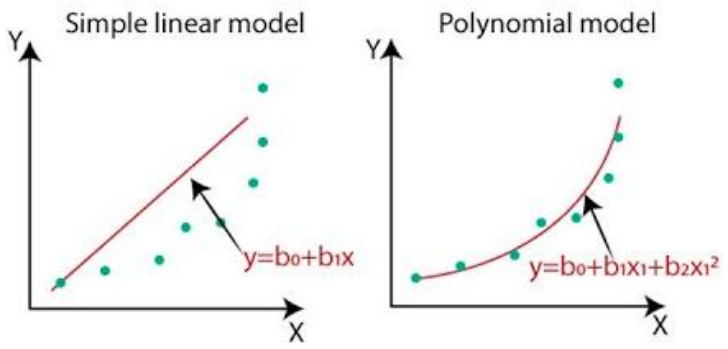
Multivariate linear regression

It is quite easy to understand as we are simply weighting the importance of each feature variable *X_n* using the coefficient weights *a_n*. We determine these weights *a_n* and the bias *b* using a Stochastic Gradient Descent (SGD).The Linear Regression curve is of the form.

Polynomial Regression

Polynomial Regression is a model which is used when the response variable is non-linear,it is rather a curve that fits into data points. General equation for Polynomial regression is of form:

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p + \epsilon$$



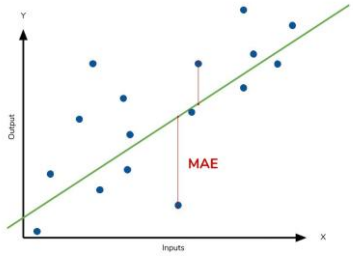
There are other Regression regularization methods such as Lasso, Ridge, etc. which work well in case of high dimensionality among the variables in the data set.

```
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score, mean_squared_error

# Fake data: y = 3x + 2 + noise

rng = np.random.RandomState(42)

X = rng.rand(200, 1) * 10

y = 3 * X.squeeze() + 2 + rng.randn(200) * 2

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

model = LinearRegression()

model.fit(X_train, y_train)

print("coef_:", model.coef_)        # slope(s)
```

```
print("intercept_:", model.intercept_)

y_pred = model.predict(X_test)

print("R^2:", r2_score(y_test, y_pred))

print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
```

b. Describe model evaluation metrics like MAE, MSE, and RMSE.

Model Evaluation Metrics:



Mean Absolute Error(MAE):

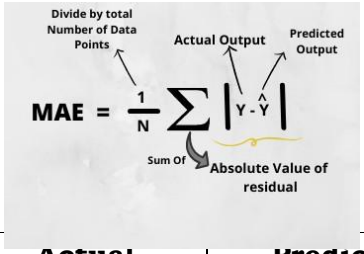
MAE is a very simple metric which calculates the absolute difference between actual and predicted values.

To better understand, let's take an example you have input data and output data and use Linear Regression, which draws a best-fit line.

Now you have to find the MAE of your model which is basically a mistake made by the model known as an error. Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset.

so, sum all the errors and divide them by a total number of observations And this is MAE. And we aim to get a minimum MAE because this is a loss.

Example:



Craft Item	Actual Price ($)	Predicted Price ($)
Necklace	25	28
Bracelet	15	14
Earrings	20	22
Ring	30	29
Brooch	40	38

Imagine you are a data scientist working for a startup that sells handmade crafts online. The company recently implemented a new pricing algorithm to predict the price of crafts based on various features like size, material, and complexity. You want to evaluate the performance of this new algorithm.

Here's a set of actual prices of crafts and the prices predicted by the algorithm:

Solution:

Here, $n = 5$,

MAE = 1/5 * (|25-28| + |15-14| + |20-22| + |30-29| + |40-38|) = 1/5 * (3+1+2+1+2) = 1/5 * (9) = 1.8

Using Scikit Learn

Scikit-learn is one of the most common and popular libraries in machine learning.

It provides a built-in function to calculate the MAE.

```
from sklearn.metrics import mean_absolute_error
actual = [25,15,20,30,40]
predicted = [28,14,22,29,38]
mae=mean_absolute_error(actual,predicted)
print(mae)
```

Output

1.8

Advantages of MAE

The MAE you get is in the same unit as the output variable. It is most Robust to outliers.

Disadvantages of MAE

The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

Mean Squared Error(MSE)

MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value.

So, above we are finding the absolute difference and here we are finding the squared difference.

What actually the MSE represents? It represents the squared distance between actual and predicted values. we perform squared

to avoid the cancellation of negative terms and it is the benefit of MSE.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}_{\text{The square of the difference between actual and predicted}}^{2}$$

Advantages of MSE
The graph of MSE is differentiable, so you can easily use it as a loss function.

Disadvantages of MSE

The value you get after calculating MSE is a squared unit of output. for example, the output variable is in meter(m) then after calculating MSE the output we get is in meter squared.

- If you have outliers in the dataset then it penalizes the outliers most and the calculated MSE is bigger. So, in short, It is not Robust to outliers which were an advantage in MAE.

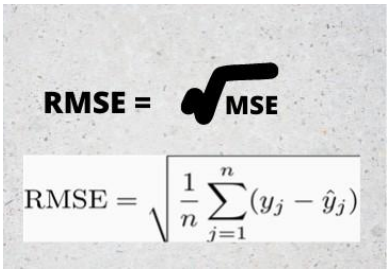
```
from sklearn.metrics import mean_squared_error
actual = [25,15,20,30,40]
predicted = [28,14,22,29,38]
mse=mean_squared_error(actual,predicted)
print(mse)
```

Output

3.8

Root Mean Squared Error(RMSE)

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.



```
from sklearn.metrics import root_mean_squared_error
actual = [25,15,20,30,40]
predicted = [28,14,22,29,38]
rmse=root_mean_squared_error(actual,predicted)
print(rmse)
```

Output:
1.9493588689617927

Advantages of RMSE

The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

Disadvantages of RMSE

It is not that robust to outliers as compared to MAE.

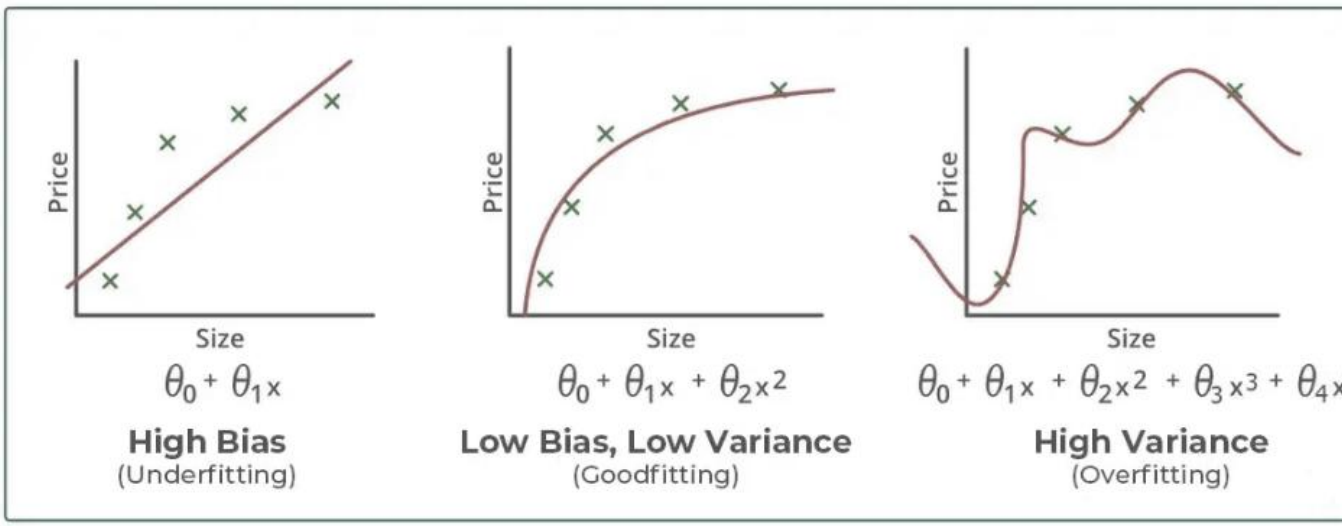
For performing RMSE we have to NumPy NumPy square root function over MSE.

Most of the time people use RMSE as an evaluation metric and mostly when you are working with deep learning techniques the most preferred metric is RMSE.

c. Write a short note on overfitting and under fitting with visual explanation.

Bias: Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.

Variance: If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.



Overfitting:

Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model.

The overfitted model has **low bias** and **high variance.**

The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model.

Overfitting is the main problem that occurs in supervised learning.

Underfitting:

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data.

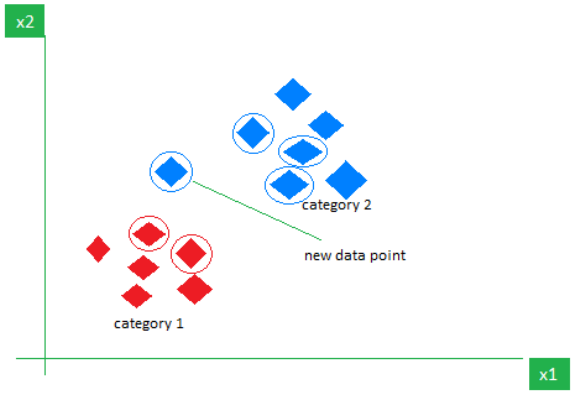
In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions.

An underfitted model has **high bias** and **low variance**.

Q.4 a. Describe the working of K-Nearest Neighbours (KNN) algorithm. Write code for classification using KNN.

The KNN algorithm is an instance-based method and is called a lazy learner. Lazy because it doesn't explicitly learn from the training data. It just memorizes the training instances which are used as "knowledge" during prediction.

As an example, consider the following table of data points containing two features:



The new point is classified as Category 2 because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points.

The image shows how KNN predicts the category of a new data point based on its closest neighbours.

- The **red diamonds** represent **Category 1** and the **blue squares** represent **Category 2**.
- The **new data point** checks its closest neighbours (circled points).

Since the majority of its closest neighbours are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

KNN works by using proximity and majority voting to make predictions.

In the **k-Nearest Neighbours (k-NN)** algorithm **k** is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

Example:

Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

If **k = 3**, the algorithm looks at the 3 closest fruits to the new one.

If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbours are apples.

Choosing the value of k for KNN Algorithm:

The value of k is critical in KNN as it determines the number of neighbors to consider when making predictions. Selecting the optimal value of k depends on the characteristics of the input data.

If the dataset has significant outliers or noise a higher k can help smooth out the predictions and reduce the influence of noisy data. However choosing very high value can lead to underfitting where the model becomes too simplistic.

Cross-Validation: A robust method for selecting the best k is to perform k-fold cross-validation. This involves splitting the data into k subsets training the model on some subsets and testing it on the remaining ones and repeating this for each subset. The value of k that results in the highest average validation accuracy is usually the best choice.

Elbow Method: In the **elbow method** we plot the model's error rate or accuracy for different values of k. As we increase k the error usually decreases initially. However after a certain point the error rate starts to decrease more slowly. This point where the curve forms an "elbow" that point is considered as best k.

Distance Metrics Used in KNN Algorithm:

KNN uses distance metrics to identify nearest neighbour, these neighbours are used for classification and regression task. To identify nearest neighbour we use below distance metrics:

1. Euclidean Distance

Euclidean distance is defined as the straight-line distance between two points in a plane or space. You can think of it like the shortest path you would walk if you were to go directly from one point to another.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{i_j})^2]}$$

2. Manhattan Distance

This is the total distance you would travel if you could only move along horizontal and vertical lines (like a grid or city streets). It's also called "taxicab distance" because a taxi can only drive along the grid-like streets of a city.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

3. Minkowski Distance

Minkowski distance is like a family of distances, which includes both **Euclidean** and **Manhattan distances** as special cases.

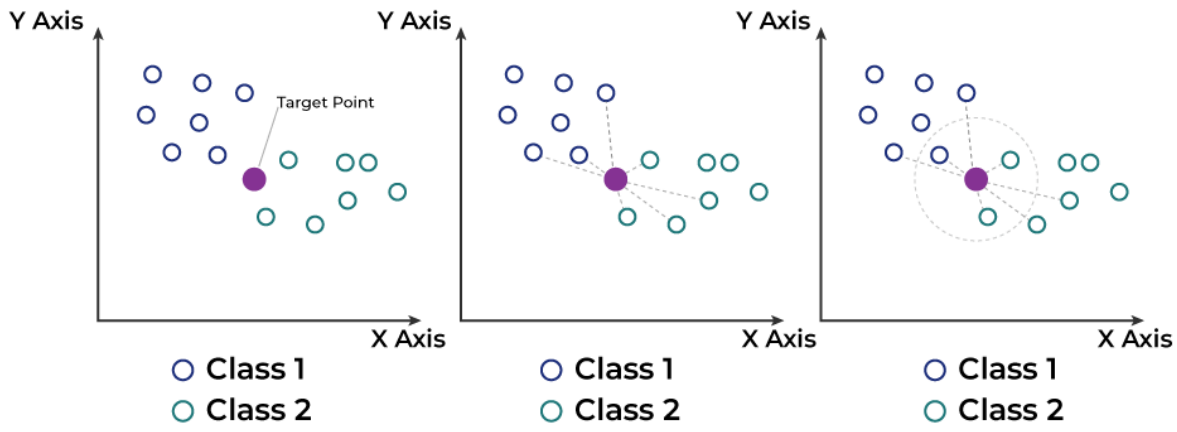
$$d(x, y) = (\sum_{i=1}^{n} (x_i - y_i)^p)^{\frac{1}{p}}$$

From the formula above we can say that when p = 2 then it is the same as the formula for the Euclidean distance and when p = 1 then we obtain the formula for the Manhattan distance.

So, you can think of Minkowski as a flexible distance formula that can look like either Manhattan or Euclidean distance depending on the value of p.

Working of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step 1: Selecting the optimal value of K

K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

When you want to classify a data point into a category (like spam or not spam), the K-NN algorithm looks at the **K closest points** in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called **majority voting**.

In regression, the algorithm still looks for the **K closest points**. But instead of voting for a class in classification, it takes the **average** of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.

b. List and explain any four classification algorithms and their use cases.

1. Logistic Regression

- **Explanation**:
  Logistic Regression is a statistical model that uses a sigmoid (logistic) function to predict the probability of a categorical outcome, usually binary (yes/no, spam/not spam). Despite its name, it is a classification algorithm, not a regression one.
- **Use Cases**:
  - Spam email detection (spam vs. not spam).
  - Medical diagnosis (disease present vs. not present).
  - Customer churn prediction (will a customer leave or stay?).

2. Decision Tree Classifier

- **Explanation**:
  A Decision Tree splits the dataset into branches based on feature values, forming a tree-like model where leaves represent class labels. It is easy to interpret and visualize.
- **Use Cases**:
  - Loan approval (approve vs. reject based on applicant details).
  - Fraud detection in transactions.
  - Predicting student performance (pass/fail).

3. Support Vector Machine (SVM)

- **Explanation**:
  SVM finds the best decision boundary (a hyperplane) that separates classes by maximizing the margin between data points of different categories. It works well in high-dimensional spaces.
- **Use Cases**:
  - Face recognition and image classification.
  - Text classification (e.g., sentiment analysis).
  - Bioinformatics (e.g., classifying proteins, cancer detection).

4. k-Nearest Neighbors (k-NN)

- **Explanation**:
  k-NN is a lazy learner that classifies new data points based on the majority

class of their 'k' closest neighbors in feature space. No explicit training model is built.

- **Use Cases**:
  - Recommender systems (finding similar users/items).
  - Handwriting recognition (digit classification).
  - Customer segmentation (grouping similar customers).

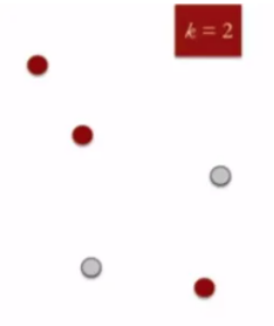
Q5. a. Discuss K-Means clustering algorithm and its implementation in Python.

K Means Clustering

K means is an iterative clustering algorithm that aims to find local maxima in each iteration. This algorithm works in these 5 steps:

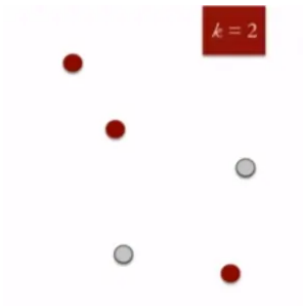
Step1:

Specify the desired number of clusters K: Let us choose k=2 for these 5 data points in 2-D space.



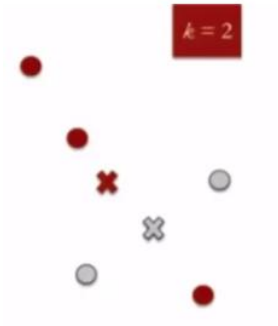
Step 2:

Randomly assign each data point to a cluster: Let's assign three points in cluster 1, shown using red color, and two points in cluster 2, shown using grey color.



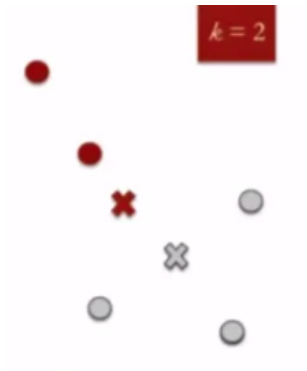
Step 3:

Compute cluster centroids: The centroid of data points in the red cluster is shown using the red cross, and those in the grey cluster using a grey cross.



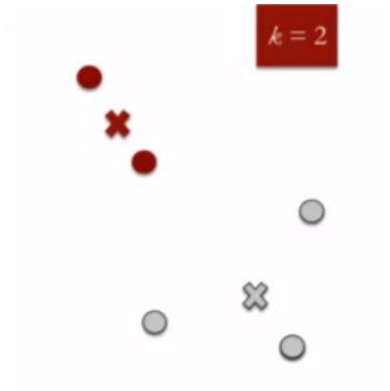
Step 4:

Re-assign each point to the closest cluster centroid: Note that only the data point at the bottom is assigned to the red cluster, even though it's closer to the centroid of the grey cluster. Thus, we assign that data point to the grey cluster.



Step 5:

Re-compute cluster centroids: Now, re-computing the centroids for both clusters.



Repeat steps 4 and 5 until no improvements are possible: Similarly, we'll repeat the 4th and 5th steps until we'll reach global optima, i.e., when there is no further switching of data points between two clusters for two successive repeats. It will mark the termination of the algorithm if not explicitly mentioned.

Example

We will cluster the houses by location and observe how house prices fluctuate across California.

```
import pandas as pd

home_data = pd.read_csv('housing.csv', usecols = ['longitude', 'latitude', 'median_house_value'])
home_data.head()
```

```
import seaborn as sns

sns.scatterplot(data = home_data, x = 'longitude', y = 'latitude', hue = 'median_house_value')
```

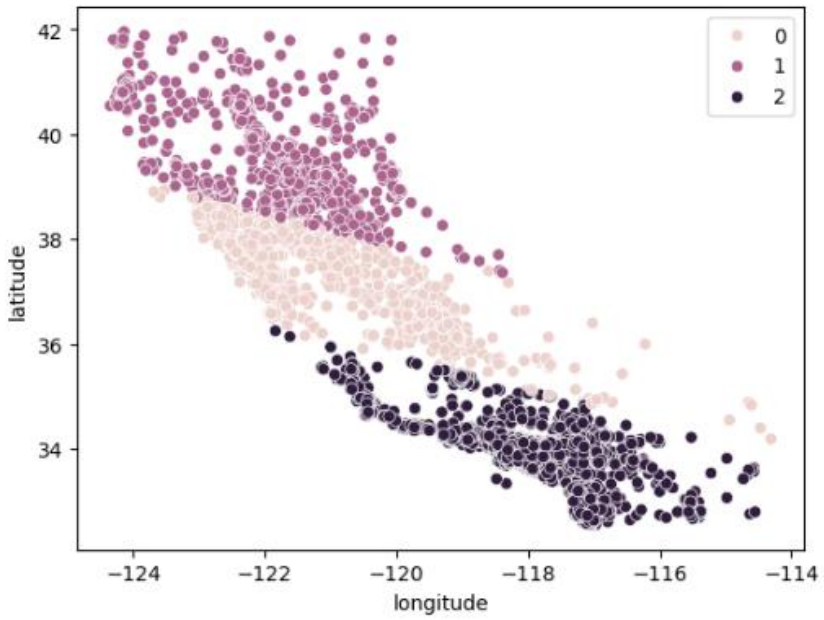
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(home_data[['latitude', 'longitude']],
                                                    home_data[['median_house_value']], test_size=0.33, random_state=0)
```

```
from sklearn import preprocessing

X_train_norm = preprocessing.normalize(X_train)
X_test_norm = preprocessing.normalize(X_test)
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, random_state = 0, n_init='auto')
kmeans.fit(X_train_norm)
```

```
KMeans
KMeans(n_clusters=3, random_state=0)
```

```
sns.scatterplot(data = X_train, x = 'longitude', y = 'latitude', hue = kmeans.labels_)
```



b. Illustrate Principal Component Analysis (PCA) and its use in dimensionality reduction.

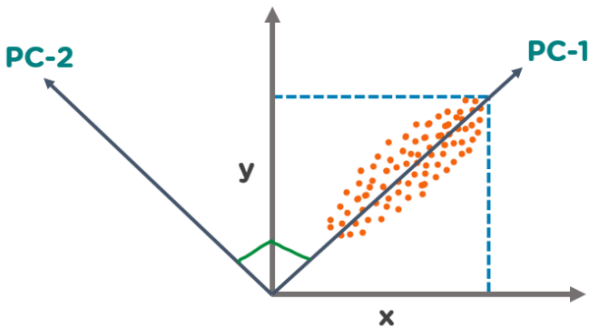
Dimensionality reduction is a technique used in machine learning and data analysis to reduce the number of features or variables under consideration. The aim is to simplify the dataset while retaining as much relevant information as possible. This is particularly useful when dealing with high-dimensional data, where the number of features is large compared to the number of samples.

There are various methods for dimensionality reduction, including:

1. **Feature selection**: Selecting a subset of the original features based on specific criteria such as relevance, importance, or correlation.
2. **Feature extraction**: Transforming the original features into a lower-dimensional space using techniques like principal component analysis (PCA), linear discriminant analysis (LDA), or t-distributed stochastic neighbor embedding (t-SNE). These methods aim to preserve the most important information while reducing the dimensionality.

By reducing dimensionality, dimensionality reduction techniques can help improve computational efficiency, mitigate the curse of dimensionality, and often lead to better performance in machine learning tasks such as classification, clustering, and visualization.

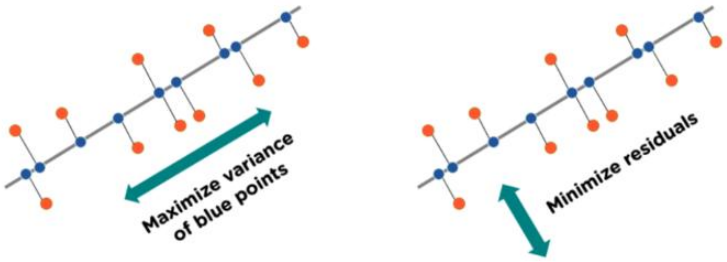
The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of large data sets. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables.



In the above figure, we have several points plotted on a 2-D plane. There are two principal components. PC1 is the primary principal component that explains the maximum variance in the data. PC2 is another principal component that is orthogonal to PC1.

The Principal Components are a straight line that captures most of the variance of the data. They have a direction and magnitude. Principal components

are orthogonal projections (perpendicular) of data onto lower-dimensional space.

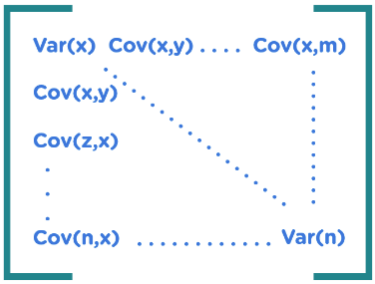


The steps involved in PCA are,

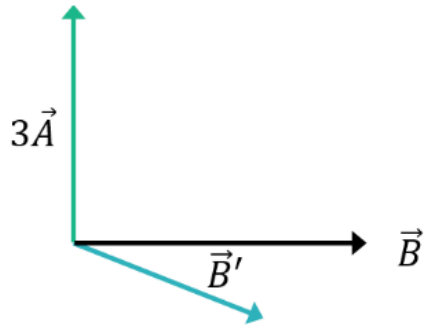
1. Standardize the data: PCA requires standardized data, so the first step is to standardize the data to ensure that all variables have a mean of 0 and a standard deviation of 1.

$$Z = \frac{x - \mu}{\sigma}$$

2. Calculate the covariance matrix: The next step is to calculate the covariance matrix of the standardized data. This matrix shows how each variable is related to every other variable in the dataset.



3. Calculate the eigenvectors and eigenvalues: The eigenvectors and eigenvalues of the covariance matrix are then calculated. The eigenvectors represent the directions in which the data varies the most, while the eigenvalues represent the amount of variation along each eigenvector.



4. Choose the principal components: The principal components are the eigenvectors with the highest eigenvalues. These components represent the directions in which the data varies the most and are used to transform the original data into a lower-dimensional space. Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.
5. Transform the data: The final step is to transform the original data into the lower-dimensional space defined by the principal components.

Advantages:

Principal component analysis (PCA) applies a relatively simple mathematical method to extract the most important dimensions with too much dimensionality so you know precisely which data to focus on.

Q.6 a. Develop a Python program to perform Hierarchical clustering and visualize using a Dendrogram.

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.preprocessing import StandardScaler

from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

def main(n_samples=80, centers=4, cluster_std=1.1, random_state=42, k=4):

    # 1) Data

    X, _ = make_blobs(n_samples=n_samples, centers=centers,
        cluster_std=cluster_std, random_state=random_state)
```

```
X = StandardScaler().fit_transform(X)
# 2) Linkage
Z = linkage(X, method="ward", metric="euclidean")
# 3) Dendrogram
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title("Hierarchical Clustering Dendrogram (Ward)")
plt.xlabel("Sample index")
plt.ylabel("Distance")
plt.tight_layout()
plt.show()
# 4) Cut to form k clusters & show simple 2D scatter
labels = fcluster(Z, t=k, criterion="maxclust")
plt.figure(figsize=(6, 5))
for lab in np.unique(labels):
    pts = X[labels == lab]
    plt.scatter(pts[:, 0], pts[:, 1], label=f"Cluster {lab}")
plt.title(f"2D View with {k} Clusters (from Dendrogram Cut)")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.legend()
plt.tight_layout()
plt.show()
```

b. Explain Apriori algorithm and its use in market basket analysis.

The association rule learning is one of the very important concepts of machine learning, and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can

understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:

Association rule learning can be divided into three types of algorithms:

1. **Apriori**
2. **Eclat**
3. **F-P Growth Algorithm**

Association rule learning works on the concept of If and Else Statement, such as if A then B.

ASSOCIATION RULE {IF} -> {THEN}.

- **IF** means **Antecedent:** An antecedent is an item found within the data
- **THEN** means **Consequent:** A consequent is an item found in combination with the antecedent.

Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also

increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Support

Support is the frequency of A or how frequently an item appears in the dataset. It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$\text{Supp(X)} = \frac{Freq(X)}{T}$$

Confidence

Confidence indicates how often the rule has been found to be true. Or how often the items X and Y occur together in the dataset when the occurrence of X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

$$\text{Confidence} = \frac{Freq(X,Y)}{Freq(X)}$$

Lift

It is the strength of any rule, which can be defined as below formula:

$$\text{Lift} = \frac{Supp(X,Y)}{Supp(X) \times Supp(Y)}$$

It is the ratio of the observed support measure and expected support if X and Y are independent of each other. It has three possible values:

- If **Lift= 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift>1**: It determines the degree to which the two itemsets are dependent to each other.

- **Lift<1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Apriori Algorithm

Apriori Algorithm is a foundational method in data mining used for discovering frequent itemsets and generating association rules. Its significance lies in its ability to identify relationships between items in large datasets which is particularly valuable in market basket analysis.
For example, if a grocery store finds that customers who buy **bread** often also buy **butter**, it can use this information to optimize product placement or marketing strategies.
The Apriori Algorithm operates through a systematic process that involves several key steps:

1. **Identifying Frequent Itemsets**: The algorithm begins by scanning the dataset to identify individual items (1-item) and their frequencies. It then establishes a **minimum support threshold**, which determines whether an itemset is considered frequent.
2. **Creating Possible item group**: Once frequent 1-itemgroup (single items) are identified, the algorithm generates candidate 2-itemgroup by combining frequent items. This process continues iteratively, forming larger itemsets (k-itemgroup) until no more frequent itemgroup can be found.
3. **Removing Infrequent Item groups**: The algorithm employs a pruning technique based on the **Apriori Property**, which states that if an itemset is infrequent, all its supersets must also be infrequent. This significantly reduces the number of combinations that need to be evaluated.
4. **Generating Association Rules**: After identifying frequent itemsets, the algorithm generates **association rules** that illustrate how items relate to one another, using metrics like **support, confidence,** and **lift** to evaluate the strength of these relationships.

Key Metrics of Apriori Algorithm

- **Support**: This metric measures how frequently an item appears in the dataset relative to the total number of transactions. A higher support indicates a more significant presence of the itemset in the dataset. Support tells us how often a

particular item or combination of items appears in all the transactions (“Bread is bought in 20% of all transactions.”)

- **Confidence**: Confidence assesses the likelihood that an item Y is purchased when item X is purchased. It provides insight into the strength of the association between two items.
- Confidence tells us how often items go together. (“If bread is bought, butter is bought 75% of the time.”)
- **Lift**: Lift evaluates how much more likely two items are to be purchased together compared to being purchased independently. A lift greater than 1 suggests a strong positive association. Lift shows how strong the connection is between items. (“Bread and butter are much more likely to be bought together than by chance.”)

Lets understand the concept of apriori Algorithm with the help of an example. Consider the following dataset and we will find frequent itemsets and generate association rules for them:

Transaction ID	Items Bought
T1	Bread, Butter, Milk
T2	Bread, Butter
T3	Bread, Milk
T4	Butter, Milk
T5	Bread, Milk

Transactions of a Grocery Shop

Step 1 : Setting the parameters

- **Minimum Support Threshold:** 50% (item must appear in at least 3/5 transactions). This threeshold is formulated from this formula:

Support(A)=Number of transactions containing itemset ATotal number of transacti onsSupport(*A*)=Total number of transactionsNumber of transactions containing it emset *A*

- **Minimum Confidence Threshold:** 70% (You can change the value of parameters as per the usecase and problem statement). This threeshold is formulated from this formula:

Confidence(X→Y)=Support(X∪Y)Support(X)Confidence(*X*→*Y*)=Support(*X*)Support(*X* ∪*Y*)

Step 2: Find Frequent 1-Itemsets

Lets count how many transactions include each item in the dataset (calculating the frequency of each item).

Item	Support Count	Support %
Bread	4	80%
Butter	3	60%
Milk	4	80%

Frequent 1-Itemsets

All items have **support% ≥ 50%**, so they qualify as **frequent 1-itemsets**. if any item has support% < 50%, It will be ommited out from the frequent 1- itemsets.

Step 3: Generate Candidate 2-Itemsets

Combine the frequent 1-itemsets into pairs and calculate their support.

For this usecase, we will get 3 item pairs (bread,butter) , (bread,ilk) and (butter,milk) and will calculate the support similiar to step 2

Item Pair	Support Count	Support %
Bread, Butter	3	60%
Bread, Milk	3	60%
Butter, Milk	2	40%

Candidate 2-Itemsets

Frequent 2-itemsets:

- {Bread, Butter}, {Bread, Milk} both meet the 50% threshold but {butter,milk} doesnt meet the threeshold, so will be ommited out.

Step 4: Generate Candidate 3-Itemsets

Combine the frequent 2-itemsets into groups of 3 and calculate their support.

for the triplet, we have only got one case i.e {bread,butter,milk} and we will calculate the support.

Item Triplet	Support Count	Support %
Bread, Butter, Milk	2	40%

Candidate 3-Itemsets

Since this **does not meet the 50% threshold**, there are no **frequent 3-itemsets**.

Step 5: Generate Association Rules

Now we generate rules from the frequent itemsets and calculate **confidence**.

Rule 1: If Bread → Butter (if customer buys bread, the customer will buy butter also)

- Support of {Bread, Butter} = 3.
- Support of {Bread} = 4.
- **Confidence = 3/4 = 75%** (Passes threshold).

Rule 2: If Butter → Bread (if customer buys butter, the customer will buy bread also)

- Support of {Bread, Butter} = 3.
- Support of {Butter} = 3.
- **Confidence = 3/3 = 100%** (Passes threshold).

Rule 3: If Bread → Milk (if customer buys bread, the customer will buy milk also)

- Support of {Bread, Milk} = 3.
- Support of {Bread} = 4.
- **Confidence = 3/4 = 75%** (Passes threshold).

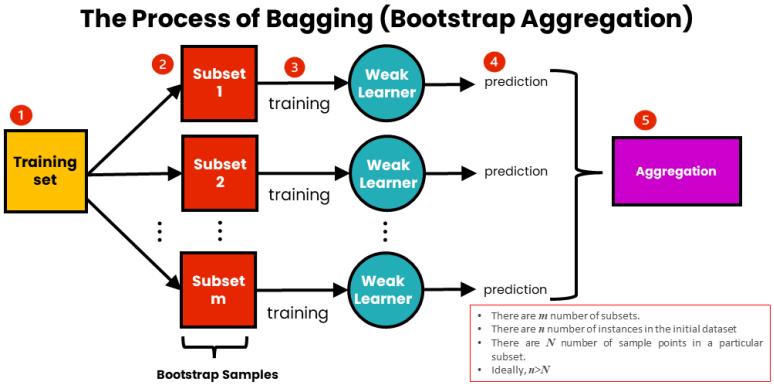
Q.7 a. Differentiate Bagging and Boosting with suitable examples.

Bagging:

Bagging (Bootstrap Aggregating) is an ensemble learning technique designed to improve the accuracy and stability of machine learning algorithms. It involves the following steps:

The steps of bagging are as follows:

1. We have an initial training dataset containing n-number of instances.
2. We create a m-number of subsets of data from the training set. We take a subset of N sample points from the initial dataset for each subset. Each subset is taken with replacement. This means that a specific data point can be sampled more than once.
3. For each subset of data, we train the corresponding weak learners independently. These models are homogeneous, meaning that they are of the same type.
4. Each model makes a prediction.
5. Aggregating the predictions into a single prediction. For this, using either max voting or averaging.



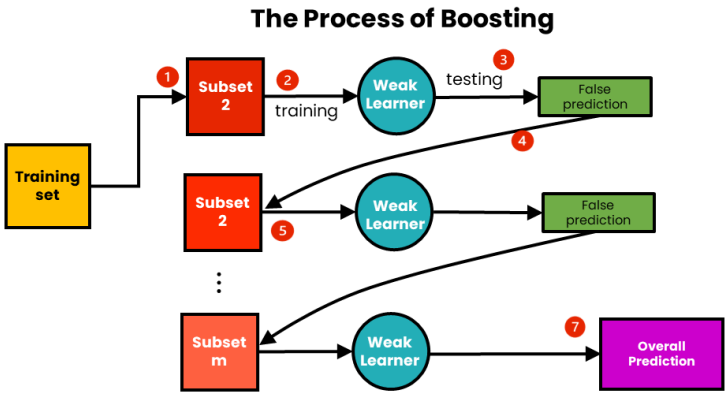
Advantages

- **Reduces Variance**: By averaging multiple predictions, bagging reduces the variance of the model and helps prevent overfitting.
- **Improves Accuracy**: Combining multiple models usually leads to better performance than individual models.
- **Eg**. **Random Forests** (an extension of bagging applied to decision trees)

Boosting:

Boosting is another ensemble learning technique that focuses on creating a strong model by combining several weak models. It involves the following steps:

- We sample m-number of subsets from an initial training dataset.
- Using the first subset, we train the first weak learner.
- We test the trained weak learner using the training data. As a result of the testing, some data points will be incorrectly predicted.
- Each data point with the wrong prediction is sent into the second subset of data, and this subset is updated.
- Using this updated subset, we train and test the second weak learner.
- We continue with the next subset until reaching the total number of subsets.
- We now have the total prediction. The overall prediction has already been aggregated at each step, so there is no need to calculate it.



Advantages:

- **Reduces Bias**: By focusing on hard-to-classify instances, boosting reduces bias and improves the overall model accuracy.
- **Produces Strong Predictors**: Combining weak learners leads to a strong **predictive model.**

b. Describe the XGBoost algorithm and provide a working Python example.

XGBOOST:

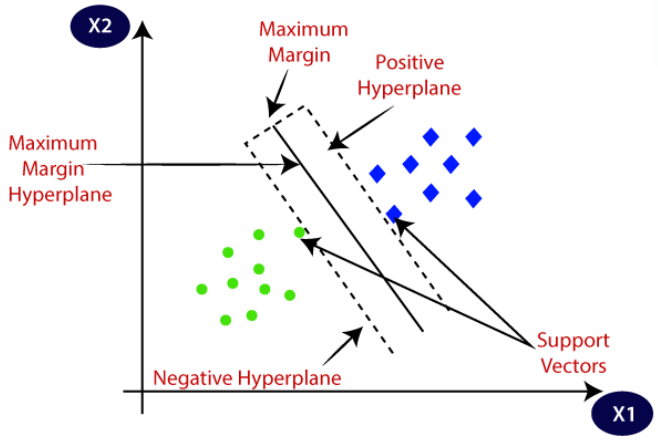
XGBoost, short for eXtreme Gradient Boosting, is an optimized and highly efficient implementation of gradient boosting. It is one of the most popular and widely used machine learning algorithms, particularly in competitions like Kaggle, due to its speed, scalability, and outstanding performance. Gradient boosting machines are generally very slow in implementation because of sequential model training. Hence, they are not very scalable. Thus, XGBoost is focused on computational speed and model performance, while introducing regularization parameters to reduce overfitting.

```python
import xgboost as xgb

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

# Load dataset

iris = load_iris()

X = iris.data

y = iris.target

# Split dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the XGBoost classifier

xgb_clf = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=42)

xgb_clf.fit(X_train, y_train)

# Make predictions

y_pred = xgb_clf.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"XGBoost Classifier Accuracy: {accuracy:.2f}")
```

c. Define Support Vector Machine. Explain kernel trick with a diagram.

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

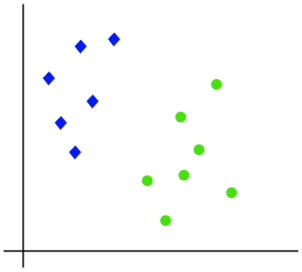
Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

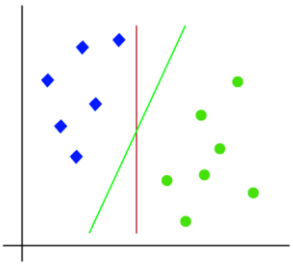
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

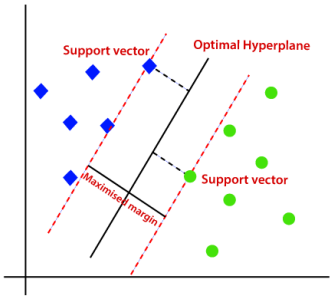
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



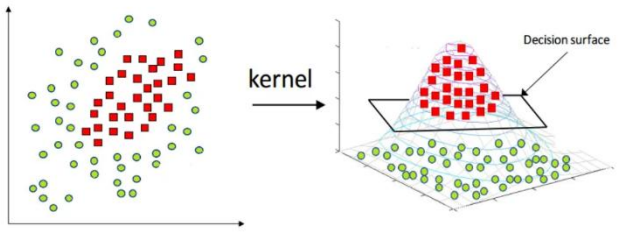
- A kernel is a function that maps data points into a higher-dimensional space **without explicitly computing the coordinates in that space.**
- This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.
- By applying a kernel function SVM **transforms the data points into a higher-dimensional space** where they become linearly separable.

The kernel trick is typically expressed as:

$$K(x, y) = \phi(x) \cdot \phi(y)$$

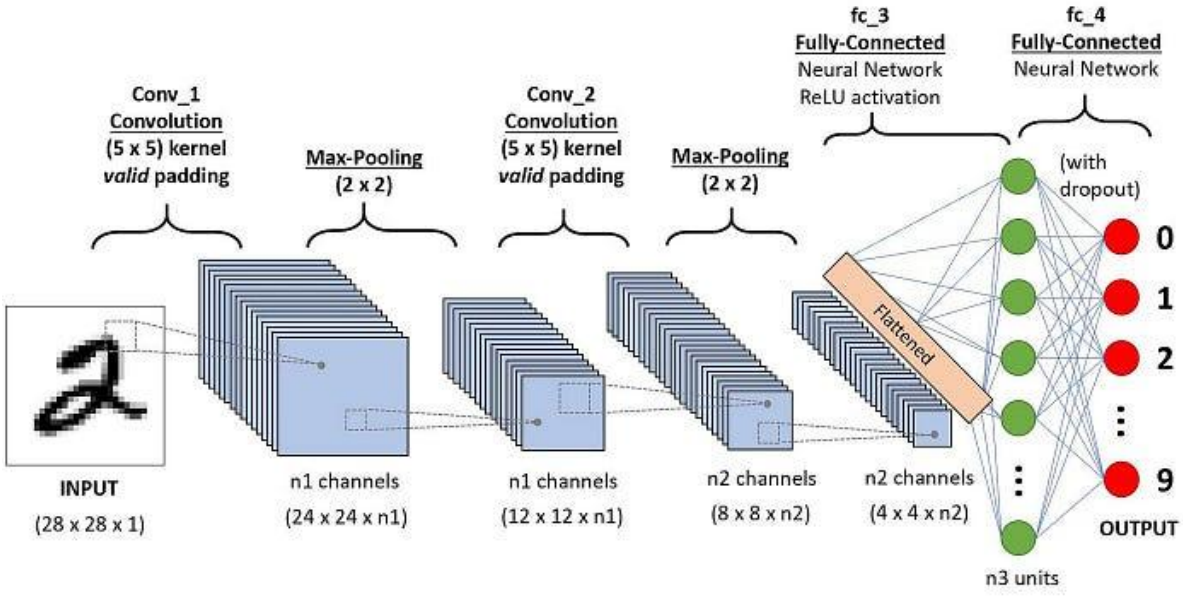
where,

- x and y are two vectors in the original input space
- ϕ is the mapping function to the higher-dimensional space.



Q.8 a. What are CNNs? Discuss their architecture and applications.

A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture that are designed for processing structured grid data like images. They use convolutional layers to detect features and are commonly used in Computer Vision.



A complete Convolution Neural Networks architecture is also known as **covnets**. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers: datasets

Let's take an example by running a covnets on of image of dimension 32 x 32 x 3.

- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.
- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: max(0, x), **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.
- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

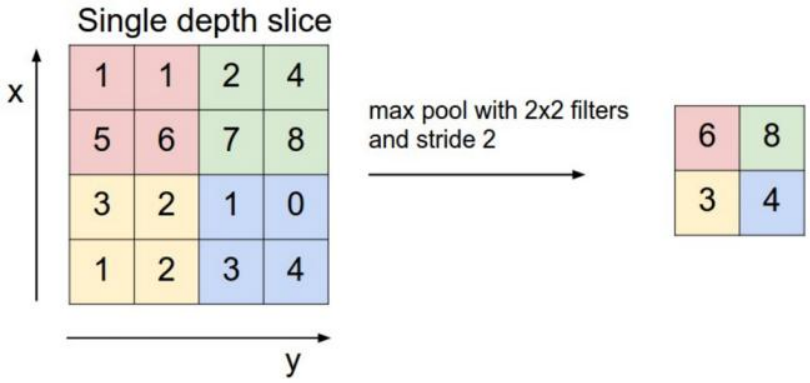


Image source: cs231n.stanford.edu

- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Applications of CNN:

Image classification: Label whole images (e.g., cat vs. dog). Common backbones: ResNet, EfficientNet.

Object detection & tracking: Locate/track multiple objects (YOLO, Faster R-CNN, SORT/DeepSORT for tracking).

Semantic & instance segmentation: Pixel-wise labeling (U-Net, DeepLab, Mask R-CNN) for tasks like road/lane segmentation.

Image restoration/enhancement: Denoising, deblurring, super-resolution (DnCNN, SRCNN, EDSR).

b. Write a Python code to build and train a simple neural network using Keras.

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

# 1) Load & preprocess data

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale pixels to [0, 1] and flatten 28x28 -> 784

x_train = (x_train.reshape(-1, 28*28) / 255.0).astype("float32")

x_test  = (x_test.reshape(-1, 28*28) / 255.0).astype("float32")

# 2) Build a simple Sequential model

model = keras.Sequential([

    layers.Input(shape=(784,)),

    layers.Dense(128, activation="relu"),

    layers.Dropout(0.2),

    layers.Dense(10, activation="softmax")  # 10 classes

])

model.compile(

    optimizer=keras.optimizers.Adam(1e-3),

    loss="sparse_categorical_crossentropy",

    metrics=["accuracy"]

)

model.summary()

# 3) Train (with a validation split)
```

```
history = model.fit(
    x_train, y_train,
    validation_split=0.1,
    epochs=5,           # increase for better accuracy
    batch_size=64,
    verbose=1
)
# 4) Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test accuracy: {test_acc:.4f} | Test loss: {test_loss:.4f}")
```

Q.9 a. Explain steps in Exploratory Data Analysis (E DA) with examples.

Exploratory Data Analysis (EDA) is like exploring a new place. You look around, observe things, and try to understand what's going on. Similarly, in EDA, you look at a dataset, check out the different parts, and try to figure out what's happening in the data.

Types of EDA

Here are five types of EDA techniques:

1. **Univariate Analysis**: In EDA Analysis, univariate analysis examines individual variables to understand their distributions and summary statistics.
2. **Bivariate Analysis**: This aspect of EDA explores the relationship between two variables, uncovering patterns through techniques like scatter plots and correlation analysis.
3. **Visualization Techniques**: EDA relies heavily on visualization methods to depict data distributions, trends, and associations using various charts and graphs.
4. **Outlier Detection**: EDA involves identifying outliers within the data, anomalies that deviate significantly from the rest, employing tools such as box plots and z-score analysis.
5. **Statistical Tests**: EDA often includes performing statistical tests to validate hypotheses or discern significant differences between

groups, adding depth to the analysis process.

Exploratory Data Analysis (EDA) is an essential step in the data analysis process. It involves analyzing and visualizing data to understand its main characteristics, uncover patterns, and identify relationships between variables. Python offers several libraries that are commonly used for EDA, including pandas, NumPy, Matplotlib, Seaborn, and Plotly. Here's a basic example of how you can perform EDA using Python:

```
# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

data = pd.read_csv('your_dataset.csv')

# Display basic information about the dataset

print("Shape of the dataset:", data.shape)

print("\nColumns:", data.columns)

print("\nData types of columns:\n", data.dtypes)

 # Display summary statistics

print("\nSummary statistics:\n", data.describe())

# Check for missing values

print("\nMissing values:\n", data.isnull().sum())

# Visualize distribution of a numerical variable

plt.figure(figsize=(10, 6))
```

```
sns.histplot(data['numerical_column'], kde=True)

plt.title('Distribution of Numerical Column')

plt.xlabel('Numerical Column')

plt.ylabel('Frequency')

plt.show()

# Visualize relationship between two numerical variables

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

sns.scatterplot(x='numerical_column_1', y='numerical_column_2',
data=data)

plt.title('Relationship between Numerical Column 1 and Numerical Column
2') plt.xlabel('Numerical Column 1')

plt.ylabel('Numerical Column 2')

plt.show() # Visualize relationship between a categorical and numerical
variable plt.figure(figsize=(10, 6))

sns.boxplot(x='categorical_column', y='numerical_column', data=data)
plt.title('Relationship between Categorical Column and Numerical
Column') plt.xlabel('Categorical Column')

plt.ylabel('Numerical Column')

plt.show()

# Visualize correlation matrix

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

sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
```

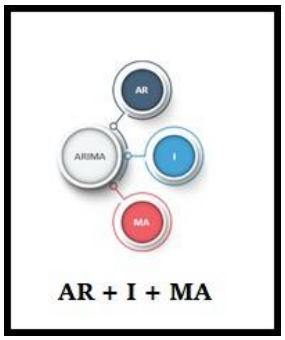
plt.show()

b. Describe the ARIMA model and its use in time series forecasting.

ARMA is a combination of the Auto-Regressive and Moving Average models for forecasting. This model provides a weakly stationary stochastic process in terms of two polynomials, one for the Auto-Regressive and the second for the Moving Average.

$$Y_t = \underbrace{\mu + \sum_{i=1}^{q} \gamma_i Y_{t-i} + \varepsilon_t}_{\text{Auto-Regressive}} + \underbrace{\sum_{i=1}^{q} \theta_i \varepsilon_{t-i}}_{\text{Moving Average}}$$

ARMA is best for predicting stationary series. **ARIMA** was thus developed to support both stationary as well as non-stationary series.



- AR ==> Uses past values to predict the future.
- MA ==> Uses past error terms in the given
- series to predict the future. I==> Uses the differencing of observation and makes the stationary data.

AR+I+MA= ARIMA

Understand the signature of ARIMA

p==> log order => No of lag observations.

d==> degree of differencing => No of times that the raw observations are differenced. q==>order of moving average => the size of the moving average window

Implementation Steps for ARIMA

Difference to make stationary on mean by removing the trend Make stationary by applying log transform.

Difference log transform to make as stationary on both statistic mean and variance Plot ACF & PACF, and identify the potential AR and MA model

Discovery of best fit ARIMA model

Forecast/Predict the value using the best fit ARIMA model

Plot ACF & PACF for residuals of the ARIMA model, and ensure no more information is left.

Implementation of ARIMA in Python

We have already discussed steps 1-5 which will remain the same; let's focus on the rest here.

Code

```
from statsmodels.tsa.arima_model import ARIMA model =
ARIMA(df_temperature, order=(0, 1, 1)) results_ARIMA = model.fit()
results_ARIMA.summary()
```

Output

ARIMA Model Results

Dep. Variable:	D.average_temperature	**No. Observations:**	240
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	-188.864
Method:	css-mle	**S.D. of innovations**	0.531
Date:	Fri, 15 Oct 2021	**AIC**	383.728
Time:	16:17:44	**BIC**	394.170
Sample:	1	**HQIC**	387.935

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0097	0.009	1.115	0.265	-0.007	0.027
ma.L1.D.average_temperature	-0.7485	0.053	-14.080	0.000	-0.853	-0.644

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.3360	+0.0000j	1.3360	0.0000

Code

```
results_ARIMA.forecast(3)[0]
```

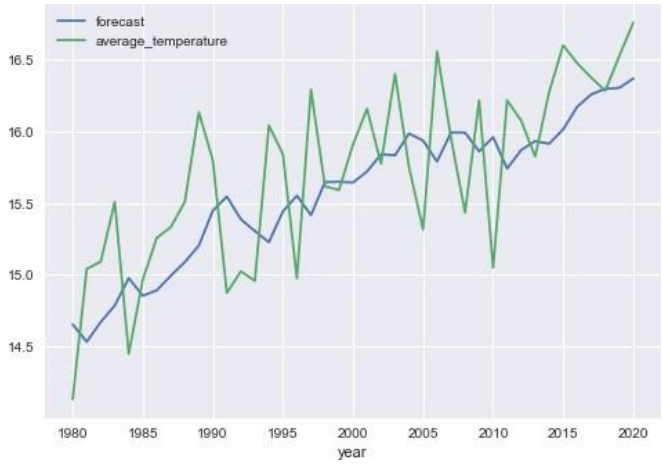
Output

```
array([16.47648941, 16.48621826, 16.49594711])
```

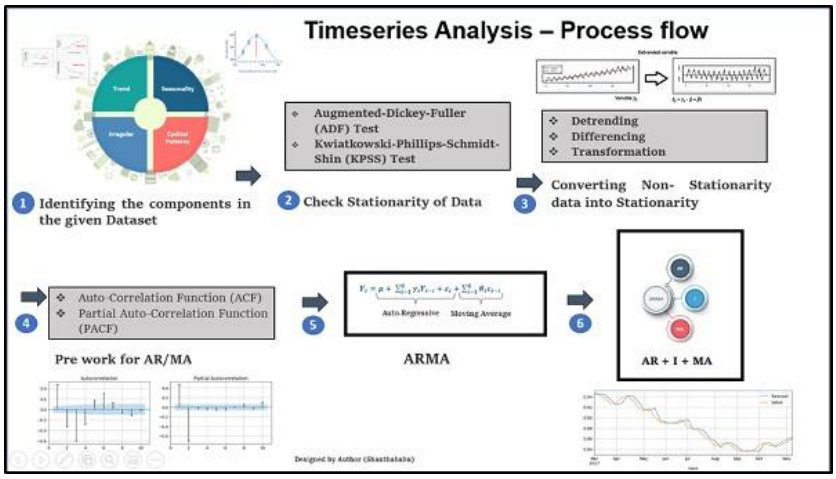
Code

```
results_ARIMA.plot_predict(start=200)   plt.show()
```

Output



Process Flow (Re-Gap)



Q.10 a. With an example, explain building a web application with Django for ML model integration.

Pre-requisites

Understanding of Machine

Learning using Python (sklearn)

Basics of Django

Basics of HTML,CSS

In this article, you will learn Machine Learning (ML) model deployment using Django. We will also discuss the ML Problem Statement which is HR Analytics.

https://dashboard.heroku.com/apps/ Let's understand what the website does. You will see three buttons with the model name. When you click on any one of the buttons it is going to download a prediction file of that particular model. To do this we are using Django. Also, make sure the headings of that .csv file are what you see in the sample submission file that you have downloaded.

1.) *ML code*

First, let's understand the ML code. We import the libraries first

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

Import the data that we downloaded and combine our train and test data. So we can do the pre-processing on the entire data directly. Copy the output column in another variable and then drop that column from the data.

Now we do pre-processing on the entire data. Look at the below code.

```
dept_counts = data['department'].value_counts()

region_count = data['region'].value_counts()

region_data = data['region'].str.replace("[a-zA-Z_]","")

data['region']=  data['region'].str.replace("[a-zA-Z_]","")

region= data['region'].astype(int) region = region.astype(int)


data = pd.get_dummies(data, columns=['gender'])

data = data.drop(['gender_f'],axis = 1)

data = pd.get_dummies(data, columns=['education'])

data = data.drop(['education_Below Secondary'],axis = 1)

data = pd.get_dummies(data,  columns=['recruitment_channel'])

data  =  data.drop(['recruitment_channel_referred'],axis= 1)


from sklearn.preprocessing import LabelBinarizer

lb_style = LabelBinarizer()
```

```
lb = lb_style.fit_transform(data["department"])

data['previous_year_rating']=
data['previous_year_rating'].fillna(data['previous_year_rating'].median())


data= data.drop(['department'],axis = 1)

d1 =data.insert(1,'Region',region)

data = data.drop(['region'],axis = 1)

d = data count_ofall_nan = data.isna().sum()

X= data.iloc[:,0:14].values

X= np.hstack((X,lb))

count_ = np.isnan(np.sum(lb))

data = data.astype(np.int64)
```

If you have worked a little on solving machine learning problems you will understand the pre-processing part easily. Basically what we are doing is converting our categorical variables into numeric values and filling our nan values with either median or mean. Here I have replaced them with a median. Pandas have a function of get_dummies that does the encoding part for us. We also have the labelbinarizer from sklearn. I have done some basic pre-processing here you need to study the dataset properly and can use better techniques to increase your accuracy.

Now that we are done with pre-processing let's divide our dataset back to our train and test data. Also, add the output column back into the training variable since we will be needing it for the model to learn. Save the test data into a .csv file. To do this...

```
#divide into train and test train = X[:length of train data,:] test = X[length of
train data:,:] test.to_csv('test_preprocessed.csv')
```

Next, we use different models and fit them into our training data. Here, I am just using 3 models, you can try different models and tune them that

will give you maximum accuracy. Now we need to save the model since we are going to predict the output using Django from our website. To save the model, I am using pickle and then with the dump function, saving the model.

```
from  sklearn.naive_bayes

import  GaussianNB

nb  =  GaussianNB()  nb.fit(X,  y)

pickle.dump(nb, open('gNB.sav','wb'))

 #random forest classifier

from sklearn.ensemble import RandomForestClassifier

random = RandomForestClassifier(n_estimators=100)

random.fit(X,y)

pickle.dump(random, open('random_forest.sav','wb'))

from sklearn.naive_bayes

import MultinomialNB

classifier_multi = MultinomialNB()

classifier_multi.fit(X, y)

pickle.dump(classifier_multi, open('classifier_multi_NB.sav','wb'))
```

Django

Now we are ready with our models saved using pickle. Let's get into Django to predict the values from the website. On the frontend, you will have three buttons in the form tag that are going to interact with Django. The form action is pointing to the link 'download', we will see that later. Below is only that part. *The bold text is quite important.*

```
<form action="download" method="POST"> {% csrf_token %}
```

<input type="submit" **name="gNB"** value="Gaussian Naive Bayes" class="btn btn-success">

<input type="submit" **name="multiNB"** value="Multinomial Naive Bayes" class="btn btn-success">

<input type="submit" **name="rf"** value="Random Forest" class="btn btn-success">

</form>

Next, go to your views.py file and first import the test data so that we can use it.

```
test_data_preprocessed = pd.read_csv('test_preprocessed.csv')

test_data_preprocessed = test_data_preprocessed.drop(['Unnamed: 0'],axis =1)

test_data_preprocessed = test_data_preprocessed.iloc[:,:].values
```

Create a function named home in the views.py file so that you can see the 3 buttons as well as all the other HTML content of your website.

```
def home(request): return render(request,"index.html")
```

In the urls.py file add the following..

```
urlpatterns = [ path('',views.home, name = 'home') ]
```

Now, we work on the functionality of the buttons. In the views.py file again, we will create a function named as models. In the HTML file above, we had named our buttons (bold text). Here, we are going to use those names to understand which one of the buttons was clicked by the user, and then it will predict values based on that model. See the below code.

```
def models(request):

if 'gNB' in request.POST:

gaussian = pickle.load(open('gNB.sav','rb'))
```

```
y_pred = gaussian.predict(test_data_preprocessed)

output = pd.DataFrame(y_pred)

output.to_csv('gaussianNB.csv')

filename = 'gaussianNB.csv'

response = HttpResponse(open(filename, 'rb').read(),
content_type='text/csv') response['Content-Length'] =
os.path.getsize(filename)

response['Content-Disposition'] = 'attachment; filename=%s' %
'gaussianNB.csv' return response if 'multiNB' in request.POST: multi =
pickle.load(open('classifier_multi_NB.sav','rb')) y_pred =
multi.predict(test_data_preprocessed) output = pd.DataFrame(y_pred)
output.to_csv('multi_NB.csv')

filename = 'multi_NB.csv'

response = HttpResponse(open(filename, 'rb').read(),
content_type='text/csv') response['Content-Length'] =
os.path.getsize(filename)

response['Content-Disposition'] = 'attachment; filename=%s' %
'multi_NB.csv' return

response

if 'rf' in request.POST:

rf = pickle.load(open('random_forest.sav','rb'))

y_pred = rf.predict(test_data_preprocessed)

output = pd.DataFrame(y_pred)

output.to_csv('rf.csv')
```

filename = 'rf.csv' response = HttpResponse(open(filename, 'rb').read(), content_type='text/csv')

response['Content-Length'] = os.path.getsize(filename) response['Content-Disposition'] = 'attachment;

filename=%s' % 'rf.csv'

return response

The If statement will check the button name then we load the test data that we imported earlier. After that, we use the predict function to predict the values. Convert it into a dataframe and then create a CSV file of it. But our main task was to download the file, so for that, we have in Django an HTTP response that will send the file to our browser for the user to download as an attachment. This is how you download the prediction files.

Lastly, we have to update our urls.py file also, since we have created a function called models. Add the following

urlpatterns = [path ('',views.home, name = 'home'), path('download', views.models)]

The download parameter is what we saw in the HTML page in the form tag. So when the user clicks on any one of the buttons this particular path is triggered which runs the function models in the views.py file.

b. Explain model deployment using Flask. Provide a deployment flow diagram.

Building the Flask Application

After saving the final model in a pickle file. We can simply

start building the flask application. Here's a step-by-step

guide to deploying your sentiment analysis model with Flask.

Create Flask Application: Organise the project directory. The project directory might look like this:

```
project/ | ├── __init__.py | ├── templates/ | | ├── index.html | └── static/ | ├──
css/- style.css | └── js/ ├── venv/ |___ sentiment.ipynb ├── requirements.txt |___

preprocess.py |___ models/ |___LRmodel.pickle └── app.py
```

Create a Folder with Suitable Project Name

The first step begins by creating a folder with some name that is suitable for your project, or you can simply name the folder as project, here we are keeping "sentiment_analysis". As you can see from the directory diagram above, there are multiple files and folders.

Directory Structure for a Flask Machine Learning Model Project

The explanation of the directory structure for a Flask machine learning model project:

Folders in the directory:

templates/: This folder contains HTML files that the Flask app will render and serve to the client. Flask uses Jinja2 templating engine for rendering templates.

css/: Contains CSS files that define the styling of the web application.

- style.css: This specific stylesheet file contains custom styles to make the web app visually appealing.

venv/: A directory for the virtual environment where Flask and other Python dependencies are installed. Keeping a virtual environment is best practice for managing project-specific dependencies separately from the global Python environment.

static/: This directory stores static files like CSS, JavaScript, and images. Flask serves these files to be used by the HTML templates.

models/: A directory for storing machine learning model files.

Files in the Directory:

init.py : This file initializes the Flask application and defines the Flask app instance.The __init__.py file can be empty or can contain initialization code if needed. In this case, since we are not creating a package and there are no specific initialization requirements, the __init__.py file can be left empty. Its presence simply indicates that the app directory is a Python package.

index.html: This HTML file is the main page of the web application. It contains the user interface where users input data for sentiment analysis, and where results are displayed.

sentiment.ipynb: This file is a Jupyter Notebook named sentiment.ipynb.

It contains the code for training and evaluating the sentiment analysis model using machine learning. Jupyter Notebooks are often used for exploratory data analysis and prototyping machine learning models. It's useful for development and documentation but not directly involved in the Flask application.

preprocess.py: This Python script contains functions for preprocessing input data before it's fed into the logistic regression model for sentiment analysis. This include cleaning text, removing stopwords, vectorization, etc.

LRmodel.pickle: A pickled file containing the trained logistic regression model. Pickling is a way to serialize and save a Python object to disk, allowing you to load the model in your Flask application for making predictions.

app.py: The main Python script for the Flask application. It initializes the Flask app and defines routes for handling web requests. It likely includes routes for rendering the HTML template, receiving input from users, preprocessing that input with preprocess.py, loading the logistic regression model from LRmodel.pickle, making predictions, and then sending the results back to the client.

GET AND POST REQUESTS

In web development, HTTP (Hypertext Transfer Protocol) defines a set of request methods that indicate the desired action to be performed for a given resource. Two common request methods are GET and POST, which serve different purposes:

GET

The GET method is used to request data from a specified resource.

Parameters are sent in the URL's query string.

GET requests can be bookmarked, cached, and shared, as they are visible in the browser's address bar.

GET requests are idempotent, meaning making the same request multiple times will produce the same result.

POST

The POST method is used to submit data to be processed to a specified resource. Parameters are sent in the request body and are not visible in the URL.

POST requests are not bookmarked or cached by default, making them more secure for sending sensitive data.

POST requests are not idempotent, meaning making the same request multiple times may produce different results, especially if the request results in changes on the server side.