

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Machine Learning (23CS6PCMAL)

Submitted by

ADITYA RAM S H (1BM22CS019)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)

BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **Aditya Ram S H (1BM22CS019)**, who is bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab faculty Incharge Name: M Lakshmi Neelima
Assistant Professor
Department of CSE, BMSCE

Dr. Kavitha Sooda
Professor & HOD
Department of CSE, BMSCE

Index

Sl. No .	Date	Experiment Title	Page No.
1	4-3-2025	Write a python program to import and export data using Pandas library functions	4
2	11-3-2025	Demonstrate various data pre-processing techniques for a given dataset	10
3	18-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	24
4	1-4-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	35
5	8-4-2025	Build Logistic Regression Model for a given dataset	47
6	15-4-2025	Build KNN Classification model for a given dataset.	65
7	15-4-2025	Build Support vector machine model for a given dataset	76
8	22-4-2025	Implement Random forest ensemble method on a given dataset.	88
9	22-4-2025	Implement Boosting ensemble method on a given dataset.	94
10	29-4-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	100
11	29-4-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	109

Github Link: <https://github.com/Ram-Aditya-7/6A-ML-LAB>

Program 1

Write a python program to import and export data using Pandas library functions

Algorithm:

Lab 1:

- ② Dataset considered: housing.csv
- To load .csv file into dataframe
`import pandas as pd`
`df = pd.read_csv('housing.csv')`
 - To display information of all columns
 $\Rightarrow \text{print(df.columns)}$
 $\Rightarrow \text{Index}([{'longitude': 'longitude', 'latitude': 'latitude', 'housing_median_age': 'housing_median_age', 'total_rooms': 'total_rooms', 'total_bedrooms': 'total_bedrooms', 'population': 'population', 'households': 'households', 'median_income': 'median_income', 'median_house_value': 'med_house_value', 'ocean_proximity': 'ocean_proximity'}], \text{dtype}=\text{object})$
 - To display statistical information of all numerical columns
 $\Rightarrow \text{df.describe()}$

	longitude	latitude	housing_median_age	total_rooms	population
count	20640	20640	20640	20640	20640
total_bedrooms	20433	20640	20640	20640	20640
households					
median_income					
med_house_value					

- To display the count of unique labels for "ocean_proximity" column.
~~`L = df.ocean_proximity.unique()`~~
~~`print(L)`~~
~~`print(len(L))`~~
~~`[['NEAR BAY', 'IN OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']]`~~

5

- To display which columns in a dataset having missing values count greater than 0.
`print(df.columns[df.isnull().any()])`
`Index(['total_bedrooms'], dtype='object')`

(1)

datasets considered: diabetes.csv, income.csv

import pandas as pd

df1 = pd.read_csv('diabetes.csv')

df2 = pd.read_csv('income.csv')

(2)

Which columns in the dataset had missing values?

How did you handle them?

print(df.columns[df.isnull().any()])

Index([], dtype='object')

(3)

Which categorical columns did you identify in dataset?

How did you encode them?

categoricals = cols = df.select_dtypes(exclude=['number']).columns

diabetes:

Gender, class

adult:

workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

	0	1	2	3	4	5	6	7	8	9
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

diabetes: gender, class

adult: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

diabetes: gender, class

adult: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

diabetes: gender, class

adult: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

diabetes: gender, class

adult: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

diabetes: gender, class

adult: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

diabetes: gender, class

adult: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

income: workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

Code:

```
#1  
import pandas as pd
```

In [32]:

```
#2  
df= pd.read_csv('housing.csv')
```

In [33]:

```
#3 df.columns
```

Out[33]:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',  
       'total_bedrooms', 'population', 'households', 'median_income',  
       'median_house_value', 'ocean_proximity'],  
      dtype='object') In [34]:
```

```
#4  
df.describe()
```

Out[34]:

	longit ude	latitud e	housing_ ge	total_ rooms	total_b edroom	popul ation	house holds	median _incom e	median_h ouse_valu e
co	20640.	20640.		20640.		20640.	20640.		
				20640.000	20433.0	20640.0	20640.000	un	00000 00000
00000		00000	00000	00000	00000	t 0	0	0	0
			000	00000	00000	000 t 0	0	0	0
m		-							
		35.631		2635.7	537.870	1425.4	499.53	3.87067	206855.81
ea	119.56		28.639486						
		861	63081 553	76744 9680 1		6909 n	9704 st2.0035 2.1359		
2181.6		421.385	1132.4 382.32 1.89982			115395.61			
			12.585558 d	32 52		15252 070	62122 9753 2		
5874									

mi	-	32.540	2.0000	1.00000	3.0000	1.0000	0.49990	14999.000
n	124.35	000	1.000000	00	0	00	00	000
25	-	33.930	1447.7	296.000	787.00	280.00	2.56340	119600.00
%	121.80	000	50000	000	0000	0000	0	0000
longit ude	latitud e	housing_ median_a ge	total_ rooms	total_b edroom s	popul ation	house holds	median _incom e	median_h ouse_valu e
50	-	34.260	2127.0	435.000	1166.0	409.00	3.53480	179700.00
%	118.49	000	00000	000	00000	0000	0	0000
75	-	37.710	3148.0	647.000	1725.0	605.00	4.74325	264725.00
%	118.01	000	00000	000	00000	0000	0	0000
m	114.31	41.950	39320.	6445.00	00000	6082.0	15.0001	500001.00
ax	0000	000	0	0000	0	00000	00	0000
In [36]:								

```
#5
l=df.ocean_proximity.unique()
print(l) print(len(l))
```

[NEAR BAY' '<1H OCEAN' 'INLAND' 'NEAR OCEAN' 'ISLAND']

5

In [39]:

```
#6 df.isna().sum()
print(df.columns[df.isnull().any()])
```

Index(['total_bedrooms'], dtype='object')

In [42]:

```
df=pd.read_csv('/content/Dataset of Diabetes .csv')
print("Sample Data") print(df.head())
print(df.isna().sum())
print(df.columns[df.isnull().any()])
```

```
categorical_cols = df.select_dtypes(exclude=['number']).columns categorical_cols
```

Sample Data

```
ID No_Pation Gender AGE Urea Cr HbA1c Chol TG HDL LDL VLDL \
0 502 17975 F 50 4.7 46 4.9 4.2 0.9 2.4 1.4 0.5
1 735 34221 M 26 4.5 62 4.9 3.7 1.4 1.1 2.1 0.6
2 420 47975 F 50 4.7 46 4.9 4.2 0.9 2.4 1.4 0.5
3 680 87656 F 50 4.7 46 4.9 4.2 0.9 2.4 1.4 0.5
4 504 34223 M 33 7.1 46 4.9 4.9 1.0 0.8 2.0 0.4
```

```
BMI CLASS
```

```
0 24.0 N
1 23.0 N
2 24.0 N
3 24.0 N
4 21.0 N
ID 0
No_Pation 0
Gender 0
AGE 0
Urea 0
Cr 0
HbA1c 0
Chol 0
TG 0
HDL 0
LDL 0
VLDL 0
BMI 0 CLASS
0 dtype: int64
```

```
Index([], dtype='object') Out[42]:
```

```
Index(['Gender', 'CLASS'], dtype='object')
```

```
In [44]:
```

```
df=pd.read_csv('/content/adult.csv')
print("Sample Data") print(df.head())
print(df.isna().sum())
print(df.columns[df.isnull().any()])
categorical_col = df.select_dtypes(exclude=['number']).columns
categorical_col
```

Sample Data

```
age workclass fnlwgt education educational-num marital-status \
0
25 Private 226802 11th 7 Never-married
```

```

1 38 Private 89814 HS-grad         9 Married-civ-spouse
2 28 Local-gov 336951 Assoc-acdm      12 Married-civ-spouse
3 44 Private 160323 Some-college     10 Married-civ-spouse
4 18 ? 103497 Some-college        10 Never-married

          occupation relationship race gender capital-gain capital-loss \
0           Machine-op-inspect Own-child Black Male       0       0
1           Farming-fishing   Husband White Male       0       0
2           Protective-serv   Husband White Male       0       0
3           Machine-op-inspect Husband Black Male    7688       0
4           ? Own-child White Female      0       0

```

```

hours-per-week native-country income 0
40 United-States <=50K
1      50 United-States <=50K
2      40 United-States >50K
3      40 United-States >50K 4      30 United-States <=50K age      0 workclass
      0 fnlwtg      0
education      0
educational-num 0
marital-status 0
occupation      0
relationship    0 race
0 gender      0
capital-gain    0
capital-loss    0
hours-per-week  0
native-country  0
income      0 dtype:
int64 Index([], dtype='object') Out[44]:
Index(['workclass', 'education', 'marital-status', 'occupation',
'relationship', 'race', 'gender', 'native-country', 'income'],
dtype='object') In [ ]:
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset Algorithm:

LAB-2

PAGE NO:
DATE: 04.03.25

```

→ tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-31",
group_by='ticker')

import yfinance as yf import pandas as pd import matplotlib.pyplot as plt
hdfc_data = data['HDFC BANK.NS']

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
print("Daily Returns for HDFC Industries:")
print(hdfc_data['Daily Return'].head())
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="HDFC Industries - Daily Return",
color="orange")
plt.tight_layout()
plt.show()

```

⇒ Daily Returns for HDFC Industries:

Date:

2024-01-01	NaN
2024-01-02	0.000589
2024-01-03	-0.015420
2024-01-04	0.016730
2024-01-05	-0.005116

```
import pandas as pd
1. df = { 'USN': [1, 2, 3, 4, 5], 'Name': ['A', 'B', 'C', 'D', 'E'],
         'Marks': [10, 20, 30, 40, 50] }
```

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

USN Name Marks

0	1	A	10
1	2	B	20
2	3	C	30
3	4	D	40
4	5	E	50

2. from sklearn.datasets import load_diabetes

diabetes = load_diabetes()

```
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
```

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

```
df.head()
```

3. file_path = 'sample_sales_data.csv'

```
df = pd.read_csv(file_path)
```

```
print(sample_data)
```

	Product	Quantity	Price	Sales	Region	Date
0	Laptop	5	1000	5000	US North	2023-10-10
1	Mouse	15	20	300	EU West	2023-10-09

df.to_csv('output.csv', index=False)

```
print("Data saved to output.csv")
```

Code:

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
data = pd.DataFrame(data)
print(data)
```

```
Name  Age      City
0   Alice  25  New York
1   Bob   30  Los Angeles
2  Charlie  35  Chicago
3   David  40  Houston
```

```
In [ ]:
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df.head()
```

```
sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) \
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
...             ...
145              6.7            3.0            5.2            2.3
146              6.3            2.5            5.0            1.9
147              6.5            3.0            5.2            2.0
148              6.2            3.4            5.4            2.3
148              1.8          149             5.9            3.0            5.1
```

```
target  0
0
1   0
2   0
3   0
4   0
...
145  2
146  2
147  2
148  2
```

149 2

[150 rows x 5 columns]

In []:

```
file_path='data.csv'  
df=pd.read_csv(file_path)  
print("Sample Data")  
print(df.head())
```

Sample Data

	ID	Name	Age	City
0	1	Alice	25	New York
1	2	Bob	30	Los Angeles
2	3	Charlie	35	Chicago
3	4	David	40	Houston
4	5	Eva	28	Phoenix

In [20]:

```
df=pd.read_csv('mobiles.csv',encoding='unicode_escape')  
print("Sample Data")  
print(df.head())
```

Sample Data

	Company	Name	Model	Name	Mobile	Weight	RAM	Front Camera	\
0	Apple	iPhone 16	128GB		174g	6GB	12MP		
1	Apple	iPhone 16	256GB		174g	6GB	12MP		
2	Apple	iPhone 16	512GB		174g	6GB	12MP		
3	Apple	iPhone 16 Plus	128GB		203g	6GB	12MP		
4	Apple	iPhone 16 Plus	256GB		203g	6GB	12MP		

	Back Camera	Processor	Battery Capacity	Screen Size	\
0	48MP	A17 Bionic	3,600mAh	6.1 inches	
1	48MP	A17 Bionic	3,600mAh	6.1 inches	
2	48MP	A17 Bionic	3,600mAh	6.1 inches	
3	48MP	A17 Bionic	4,200mAh	6.7 inches	
4	48MP	A17 Bionic	4,200mAh	6.7 inches	

	Launched Price (Pakistan)	Launched Price (India)	Launched Price (China)	\
0	PKR 224,999	INR 79,999	CNY 5,799	
1	PKR 234,999	INR 84,999	CNY 6,099	
2	PKR 244,999	INR 89,999	CNY 6,499	
3	PKR 249,999	INR 89,999	CNY 6,199	
4	PKR 259,999	INR 94,999	CNY 6,499	

	Launched Price (USA)	Launched Price (Dubai)	Launched Year
0	USD 799	AED 2,799	2024
1	USD 849	AED 2,999	2024
2	USD 899	AED 3,199	2024
3	USD 899	AED 3,199	2024
4	USD 949	AED 3,399	2024 In [21]:

```
#1
df={
    'USN':[1,2,3,4,5],
    'Name':['A','B','C','D','E'],
    'Marks':[10,20,30,40,50]
}
```

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

	USN	Name	Marks
0	1	A	10
1	2	B	20
2	3	C	30
3	4	D	40
	4	E	50 In [24]:

```
#2
from sklearn.datasets import load_diabetes diabetes=
load_diabetes()
df=pd.DataFrame(diabetes.data,columns=diabetes.feature_names)
df['target']=diabetes.target df.head() print(df)
```

	age	sex	bmi	bp	s1	s2	s3	\
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	
..	
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	439
	0.041708	0.050680	-0.015906	0.017293	-0.037344	-0.013840	-0.024993	440
	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	
441	-0.045472	-0.044642	-0.073030	-0.081413	0.083740	0.027809	0.173816	
								s4 s5
s6	target							
0	-0.002592	0.019907	-0.017646	151.0				
1	-0.039493	-0.068332	-0.092204	75.0				

```

2 -0.002592 0.002861 -0.025930 141.0
3 0.034309 0.022688 -0.009362 206.0 4 -0.002592 -0.031988 -0.046641 135.0
...
437 ... ...
437 -0.002592 0.031193 0.007207 178.0
438 0.034309 -0.018114 0.044485 104.0
439 -0.011080 -0.046883 0.015491 132.0
440 0.026560 0.044529 -0.025930 220.0
441 -0.039493 -0.004222 0.003064 57.0

```

[442 rows x 11 columns] In

[29]:

```
#3 file_path='sample_sales_data.csv'
df=pd.read_csv(file_path)
print("Sample Data")
print(df.head())
```

Sample Data

	Product	Quantity	Price	Sales	Region
0	Laptop	5	1000	5000	North
1	Mouse	15	20	300	West
2	Keyboard	10	50	500	East
3	Monitor	8	200	1600	South

4 Laptop 12 950 11400 North In [30]:

```
#4 df=pd.read_csv('/content/Dataset of Diabetes
.csv') print("Sample Data")
print(df.head())
```

Sample Data

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	\
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	

BMI CLASS

0	24.0	N
1	23.0	N
2	24.0	N
3	24.0	N
4	21.0	N

In [28]:

```
df.to_csv('output.csv', index=False)  
print("Data saved to output.csv")
```

Data saved to output.csv

In [34]:

```
sales_df = pd.read_csv('sample_sales_data.csv')  
sales_by_region =  
sales_df.groupby('Region')['Sales'].sum() print("\nTotal  
sales by region:") print(sales_by_region)  
best_selling_products = sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)  
print("\nBest-selling products by quantity:") print(best_selling_products)  
# Saving the sales by region data to a CSV file  
sales_by_region.to_csv('sales_by_region.csv') # Saving  
the best-selling products data to a CSV file  
best_selling_products.to_csv('best_selling_products.csv'  
) print("\nAnalysis results saved to CSV files.")
```

Total sales by region:

Region

East	770
North	16400
South	3070
West	650

Name: Sales, dtype: int64

Best-selling products by quantity:

Product

Mouse	29
Laptop	17
Keyboard	16
Monitor	15

Name: Quantity, dtype: int64

Analysis results saved to CSV files.

In [35]:

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

In [36]:

```
# Step 2: Downloading Stock Market Data
# Define the ticker symbols for Indian companies
# Example: Reliance Industries (RELIANCE.NS), TCS (TCS.NS), Infosys (INFY.NS) tickers
= ["RELIANCE.NS", "TCS.NS", "INFY.NS"]

# Fetch historical data for the last 1 year
data = yf.download(tickers, start="2022-10-01", end="2023-10-01", group_by='ticker')

# Display the first 5 rows of the dataset
print("First
5 rows of the dataset:")
print(data.head())
```

YF.download() has changed argument auto_adjust default to True
[*****100%*****] 3 of 3 completed

First 5 rows of the dataset:

Ticker	RELIANCE.NS	\\				
Price	Open	High	Low	Close	Volume	
Date						
2022-10-03	1088.493134	1100.076669	1075.521371	1078.479858	11852723	
2022-10-04	1091.360592	1100.554607	1087.878645	1098.369873	8948850	
2022-10-06	1105.561293	1115.119370	1100.622856	1102.420654	13352162	
2022-10-07	1099.029980	1112.343173	1099.029980	1107.086182	7714340	2022-10-10
	1094.637648	1100.372591	1086.900123	1095.001831	6329527	

Ticker	TCS.NS	\\				
Price	Open	High	Low	Close	Volume	
Date						
2022-10-03	2799.043864	2823.062325	2779.417847	2789.651367	1763331	
2022-10-04	2831.707536	2895.305232	2825.212304	2888.903320	2145875	
2022-10-06	2907.454507	2919.603947	2890.118145	2898.996582	1790816	
2022-10-07	2894.744206	2901.847047	2858.015696	2864.370605	1939879	2022-10-10
	2813.062865	2922.407833	2808.390003	2914.510742	3064063	

Ticker	INFY.NS	\\				
Price	Open	High	Low	Close	Volume	
Date						
2022-10-03	1337.743240	1337.743240	1313.110574	1320.453003	4943169	
2022-10-04	1345.038201	1356.928245	1339.638009	1354.228149	6631341	
2022-10-06	1369.007786	1383.029504	1368.155094	1378.624023	6180672	2022-10-07
	1370.286676	1381.181893	1364.412779	1374.881592	3994466	
2022-10-10	1351.338576	1387.956005	1351.338576	1385.729614	5274677	In
[38]:						

```

# Step 3: Basic Data Exploration
# Check the shape of the dataset
print("\nShape of the dataset:")
print(data.shape)

# Check column names
print("\nColumn names:")
print(data.columns)

# Summary statistics for a specific stock (e.g., Reliance)
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())

```

Shape of the dataset:

(247, 15)

Column names:

```

MultiIndex([('RELIANCE.NS', 'Open'),
            ('RELIANCE.NS', 'High'),
            ('RELIANCE.NS', 'Low'),
            ('RELIANCE.NS', 'Close'),
            ('RELIANCE.NS', 'Volume'),
            ('TCS.NS', 'Open'),
            ('TCS.NS', 'High'),
            ('TCS.NS', 'Low'),
            ('TCS.NS', 'Close'),
            ('TCS.NS', 'Volume'),
            ('INFY.NS', 'Open'),
            ('INFY.NS', 'High'),
            ('INFY.NS', 'Low'),
            ('INFY.NS', 'Close'),
            ('INFY.NS', 'Volume')],
            names=['Ticker', 'Price'])

```

Summary statistics for Reliance Industries:

Price	Open	High	Low	Close	Volume	count
247.000000	247.000000	247.000000	247.000000	247.000000	2.470000e+02	mean
1147.548565	1156.216094	1137.195377	1146.522438	1.316652e+07	std	
65.890232	66.850140	65.752477	66.687266	6.754099e+06	min	

```
1008.159038 1010.434788 992.228728 1001.900696 3.370033e+06 25%
1098.881943 1103.399363 1084.795104 1097.357178 8.717141e+06
50% 1147.435126 1155.036230 1138.787343 1147.253052 1.158959e+07
75% 1196.261444 1203.625226 1184.985050 1195.064575 1.530302e+07
max 1288.076823 1299.910771 1273.056848 1293.470337 5.708188e+07
```

In [39]:

```
# Calculate daily returns
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\nDaily returns for Reliance Industries:") print(reliance_data['Daily
Return'].head())

#
```

Daily returns for Reliance Industries:

Date

```
2022-10-03      NaN
2022-10-04  0.018443
2022-10-06  0.003688
2022-10-07  0.004232
2022-10-10 -0.010915
```

Name: Daily Return, dtype: float64

```
<ipython-input-39-2bd40fbe735c>:2: SettingWithCopyWarning:
```

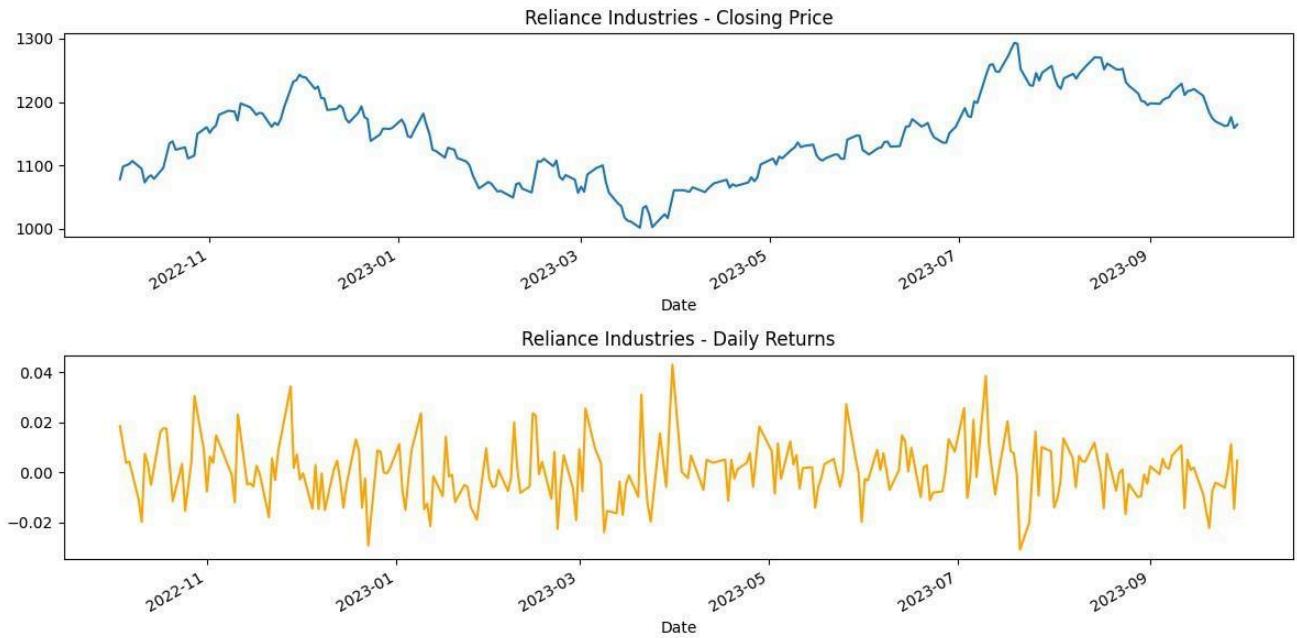
A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:

```
https://pandas.pydata.org/pandasdocs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy reliance_data['Daily Return'] =
reliance_data['Close'].pct_change() In [40]:
```

```
# Plot the closing price and daily returns plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1) reliance_data['Close'].plot(title="Reliance
Industries - Closing Price") plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```



In [41]:

```
# Step 4: Saving the Processed Data to a New CSV File
# Save the Reliance data to a CSV file reliance.to_csv('reliance_stock_data.csv')
print("\nReliance stock data saved to 'reliance_stock_data.csv'.")
```

Reliance stock data saved to 'reliance_stock_data.csv'.

In [43]:

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')
```

[*****100%*****] 3 of 3 completed

In [44]:

```
hdfc_data = data['HDFCBANK.NS']
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
print("\nDaily returns for HDFC Industries:")
print(hdfc_data['Daily Return'].head())
plt.figure(figsize=(12, 6)) plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="HDFC Industries - Closing Price") plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="HDFC Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

```
<ipython-input-44-f8f2798a87f5>:2: 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:

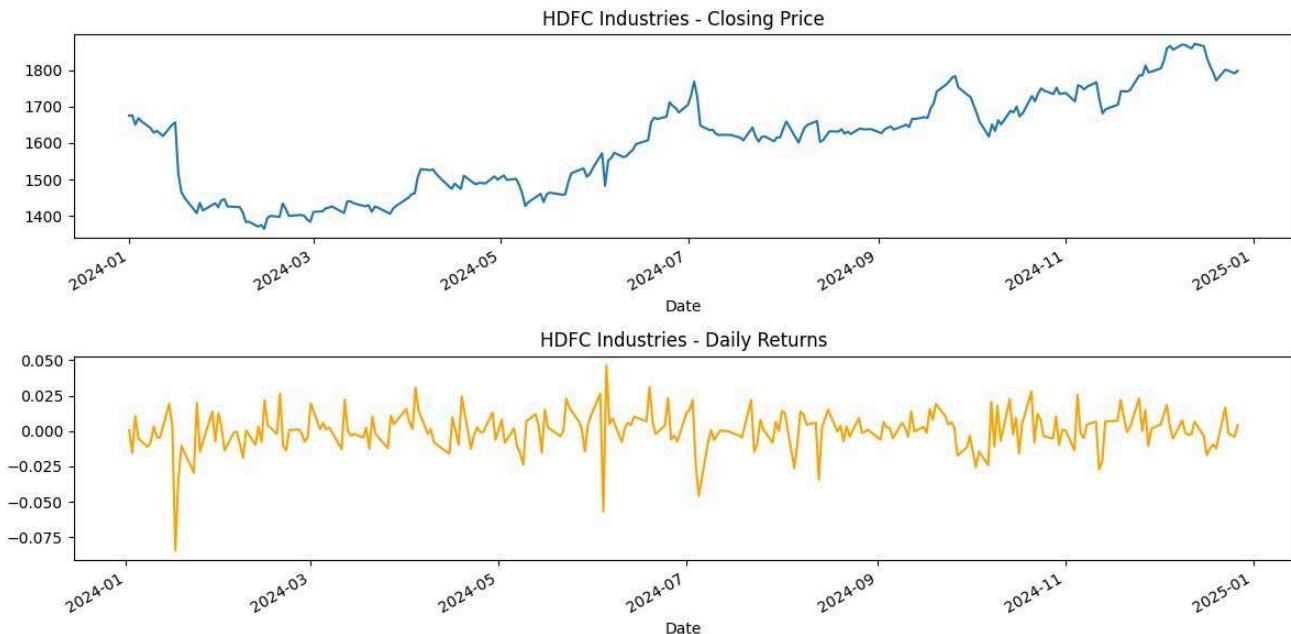
https://pandas.pydata.org/pandasdocs/stable/user_guide/indexing.html#

```
returning-a-view-versus-a-copy  hdfc_data['Daily Return'] =  
hdfc_data['Close'].pct_change() Daily returns for HDFC Industries:
```

Date

```
2024-01-01      NaN  
2024-01-02   0.000589  
2024-01-03  -0.015420  
2024-01-04   0.010730  
2024-01-05  -0.005116
```

Name: Daily Return, dtype: float64



In [45]:

```
icici_data = data['ICICIBANK.NS']  
icici_data['Daily Return'] = icici_data['Close'].pct_change()  
print("\nDaily returns for ICICI Industries:")  
print(icici_data['Daily Return'].head())  
plt.figure(figsize=(12, 6)) plt.subplot(2, 1, 1)  
icici_data['Close'].plot(title="ICICI Industries - Closing Price") plt.subplot(2,  
1, 2)  
icici_data['Daily Return'].plot(title="ICICI Industries - Daily Returns", color='orange')  
plt.tight_layout()
```

```
plt.show()
```

```
<ipython-input-45-6f81270ab07d>:2: 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/pandasdocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

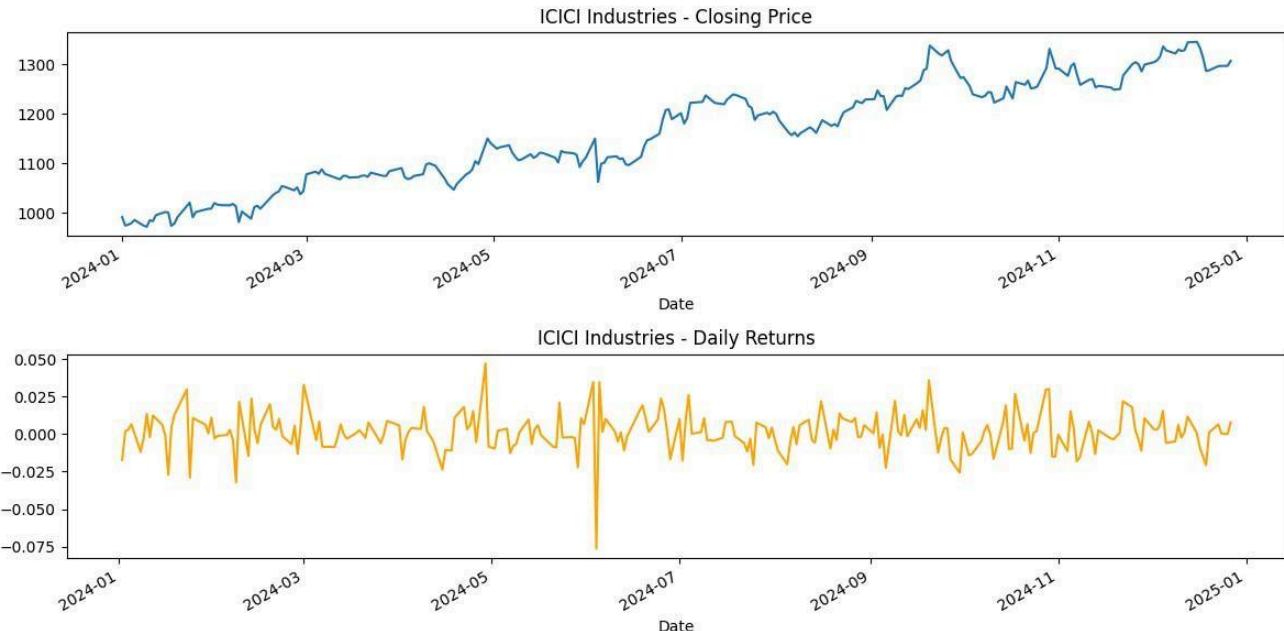
```
icici_data['Daily Return'] = icici_data['Close'].pct_change()
```

Daily returns for ICICI Industries:

Date

```
2024-01-01      NaN  
2024-01-02   -0.017160  
2024-01-03    0.001833  
2024-01-04    0.003150  
2024-01-05    0.006635
```

Name: Daily Return, dtype: float64



In [46]:

```
kotak_data = data['KOTAKBANK.NS']  
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()  
print("\nDaily returns for Kotak Industries:")  
print(kotak_data['Daily Return'].head())  
plt.figure(figsize=(12, 6)) plt.subplot(2, 1, 1)  
kotak_data['Close'].plot(title="Kotak Industries - Closing Price") plt.subplot(2,  
1, 2)
```

```
kotak_data['Daily Return'].plot(title="Kotak Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

```
<ipython-input-46-098de9be386a>:2: 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:

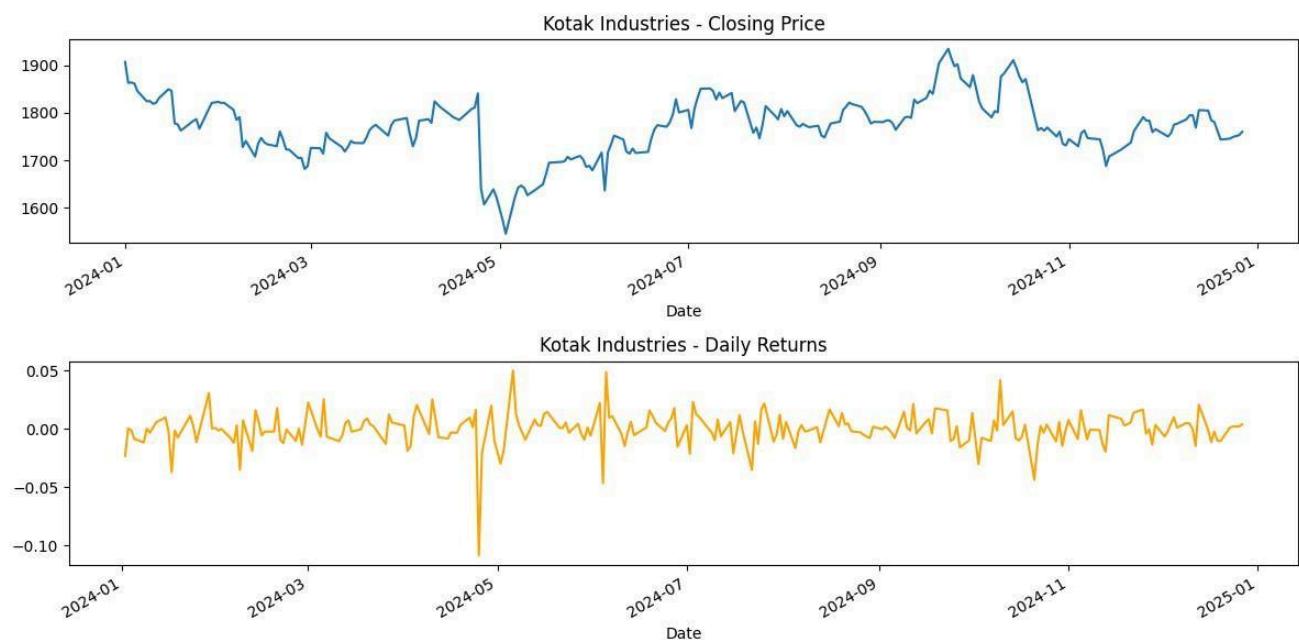
https://pandas.pydata.org/pandasdocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
kotak_data['Daily Return'] =
kotak_data['Close'].pct_change() Daily returns for Kotak Industries:
```

Date

```
2024-01-01      NaN
2024-01-02 -0.023099
2024-01-03  0.000456
2024-01-04 -0.001233
2024-01-05 -0.008586
```

Name: Daily Return, dtype: float64



Program 3

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Algorithm:

Lab 5

PAGE NO :
DATE : 25-03-25

attribute : a1

values (a1) = True, False

$S = [6+, 4-]$ Entropy(S) = $-\frac{6}{10} \log_2 \frac{6}{10} - \frac{4}{10} \log_2 \frac{4}{10} = 0.9709$

$S_{True} = [1+, 4-]$ Entropy(S_{True}) = $\frac{1}{5} \log_2 \frac{1}{5} - \frac{4}{5} \log_2 \frac{4}{5} = 0.7219$

$S_{False} = [5+, 0-]$ Entropy(S_{False}) = 0.10

Gain(S, a1) = Entropy(S) - $\sum_{v \in \{\text{True, False}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$

Gain(S, a1) = Entropy(S) - $\frac{5}{10} \text{Entropy}(S_{True}) - \frac{5}{10} \text{Entropy}(S_{False})$

Gain(S, a1) = $0.9709 - \frac{5}{10} \cdot 0.7219 - \frac{5}{10} \cdot 0.10 = 0.6090$

attribute : c2

values (a2) = Not, Cool

$S = [6+, 4-]$ Entropy(S) = $-\frac{6}{10} \log_2 \frac{6}{10} - \frac{4}{10} \log_2 \frac{4}{10} = 0.9709$

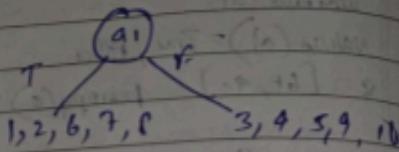
$S_{Not} = [2+, 3-]$ Entropy(S_{Not}) = $\frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_2 \frac{3}{5} = 0.9209$

$S_{Cool} = [4+, 1-]$ Entropy(S_{Cool}) = $-\frac{4}{5} \log_2 \frac{4}{5} - \frac{1}{5} \log_2 \frac{1}{5} = 0.2219$

Gain(S, a2) = Entropy(S) - $\sum_{v \in \{\text{Not, Cool}\}} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$

Gain(S, a2) = Entropy(S) - $\frac{6}{10} \text{Entropy}(S_{Not}) - \frac{4}{10} \text{Entropy}(S_{Cool})$

Gain(S, a2) = $0.9709 - \frac{6}{10} \cdot 0.9209 - \frac{4}{10} \cdot 0.2219 = 0.499$

$\text{Gain}(S, a_1)$ $\text{Gain}(S, a_2)$ $\text{Gain}(S, a_3) = 0.4199$ Instance a_2, a_3 classification

		No.
1	Not High	No
2	Not High	No
6	High	No
7	Not High	No
8	Not Normal	Yes

attribute: a_2 value $a_2 = \text{Not, cost}$ value $(a_2) = \text{Not, cost}$

$$S_{a_1} = [1+, 4-] \quad \text{Entropy}(S_{a_1}) = -\frac{1}{5} \log_2 \frac{1}{5} - \frac{4}{5} \log_2 \frac{4}{5} = 0.7219$$

$$S_{\text{Not}} = [1+, 3-] \quad \text{Entropy}(S_{\text{Not}}) = -\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4} = 0.8112$$

$$S_{\text{cost}} = [0+, 1-] \quad \text{Entropy}(S_{\text{cost}}) = 0.0$$

~~$$\text{Gain}(S, a_2) = \text{Entropy}(S) - \sum_{v \in C(a_2)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$~~

~~$$\text{Gain}(S, a_2) = 0.9709 - \frac{1}{5} * 0.8112 - \frac{4}{5} * 0.0 = 0.3219$$~~

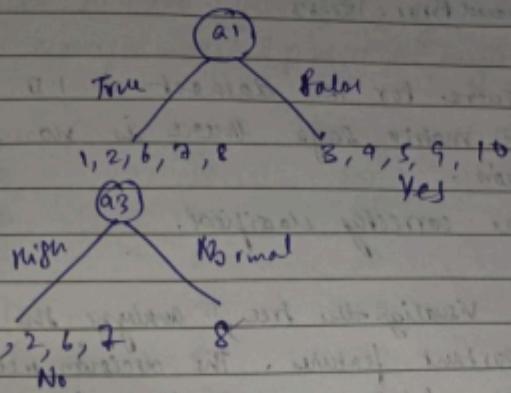
Attribute: a_3 Value $(a_3) = \text{High, Normal}$

$$S_{a_1} = [1+, 4-] \quad \text{Entropy}(S_{a_1}) = -\frac{1}{5} \log_2 \frac{1}{5} - \frac{4}{5} \log_2 \frac{4}{5} = 0.7219$$

$$S_{\text{High}} = [0+, 1-] = 0$$

$$S_{\text{Normal}} = [1+, 0-] = 0$$

$$\text{Gain } (S, a_2) = \frac{0.7219}{0.9709} - \frac{9 \times 0.0}{5} - \frac{1 \times 0.1}{5} = 0.7219$$



```

code:
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
df = pd.read_csv('petrol-consumption.csv')
X = df.drop(['Petrol-Consumption'], axis=1)
y = df['Petrol-Consumption']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
  
```

~~regressor = DecisionTreeRegressor()~~

~~regressor.fit(X_train, y_train)~~

~~y_pred = regressor.predict(X_test)~~

~~mae = mean_absolute_error(y_test, y_pred)~~

~~mse = mean_squared_error(y_test, y_pred)~~

~~rmse = mse**0.5~~

~~print(f"Mean Absolute Error: {mae}")~~

~~print(f"Mean Squared Error: {mse}")~~

~~print(f"Root Mean Squared Error: {rmse}")~~

~~Output:~~

Mean Absolute Error: 96.2

Mean Squared Error: 10558.2

Root Mean Squared Error: 103.13

- The accuracy score for IRIS dataset is 1.0. The confusion matrix says there is no misclassification. All classes are correctly classified.

- We need to visualize the tree, analyze the tree & identify important features. The decision tree classifier discrete class labels are predicted for each node. The regression tree has continuous values for each instance. It aims to create splits that minimize the variance of target variable.

→ Consider
(X, y)

KNN c

Perfor

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Code:

In []:

```
# -*- coding: utf-8 -*- """Decision_Tree.ipynb
```

Automatically generated by Colab.

Original file is located at

```
https://colab.research.google.com/drive/1RXDK8CR1doVCMHgkaXpJsNLAvzOlaXdd  
"""
```

```
import pandas as pd from sklearn.preprocessing  
import LabelEncoder from sklearn.tree import  
DecisionTreeClassifier from  
sklearn.model_selection import train_test_split  
from sklearn.metrics import accuracy_score, classification_report  
  
# Create the dataset data  
= {  
    'a1': [True, True, False, False, False, True, True, True, False, False],  
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],  
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],  
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes'] }  
  
data  
  
# Convert to DataFrame df  
= pd.DataFrame(data)  
  
# Convert categorical data to numerical  
data_label_encoders = {} for column in  
df.columns: le = LabelEncoder()  
    df[column] = le.fit_transform(df[column])  
label_encoders[column] = le  
  
# Split the dataset into features and target  
X = df.drop('Classification', axis=1) y =  
df['Classification']  
  
# Split the data into training and testing sets  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)  
  
# Initialize the Decision Tree Classifier with entropy as the criterion
```

```

clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier accuracy =
accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

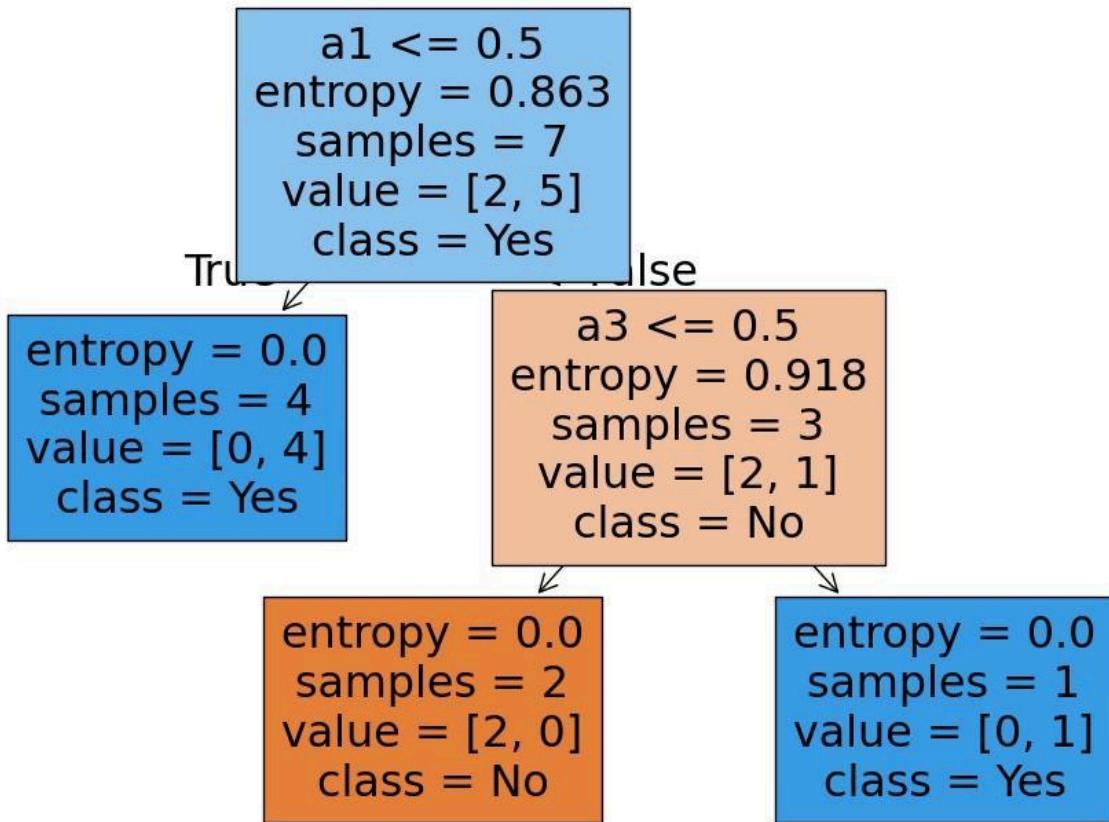
# Optionally, visualize the decision tree from
sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes']) plt.show()

```

Accuracy: 1.00

	precision	recall	f1-score	support
No	1.00	1.00	1.00	2
Yes	1.00	1.00	1.00	1
accuracy		1.00	1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3



In []:

```

from sklearn.datasets import load_iris from sklearn.tree import
DecisionTreeClassifier from sklearn.model_selection import
train_test_split from sklearn.metrics import accuracy_score,
confusion_matrix import seaborn as sns import
matplotlib.pyplot as plt

# Load the iris dataset
iris = load_iris() X =
iris.data
y = iris.target

# Split data 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)

# Initialize and train the DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

```

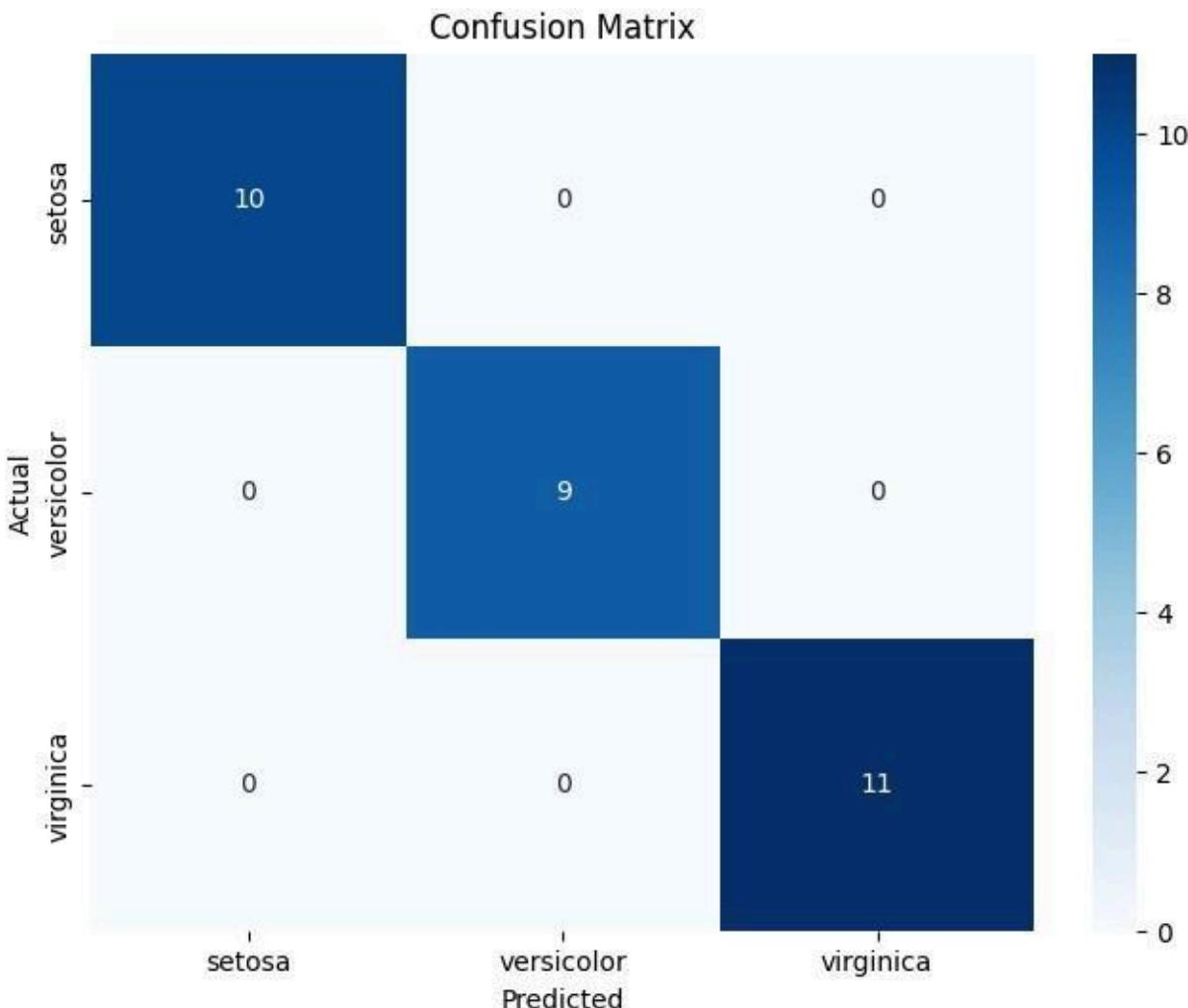
```

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Create and display the confusion matrix cm
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted") plt.ylabel("Actual") plt.title("Confusion Matrix")
plt.show()

```

Accuracy: 1.0



In []:

```
import pandas as pd
```

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

try:
    df = pd.read_csv('drug.csv')
except FileNotFoundError:
    print("Error: 'drugdataset.csv' not found. Please upload the file or provide the correct path.")
    exit()

# Identify categorical columns (replace with your actual column names)
categorical_cols = ['Sex', 'BP', 'Cholesterol', 'Na_to_K']

# Use one-hot encoding for categorical features
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Split data into features (X) and target (y)
X = df.drop('Drug', axis=1) # Assuming 'Drug' is the target variable
y = df['Drug']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Train the classifier
clf.fit(X_train, y_train)

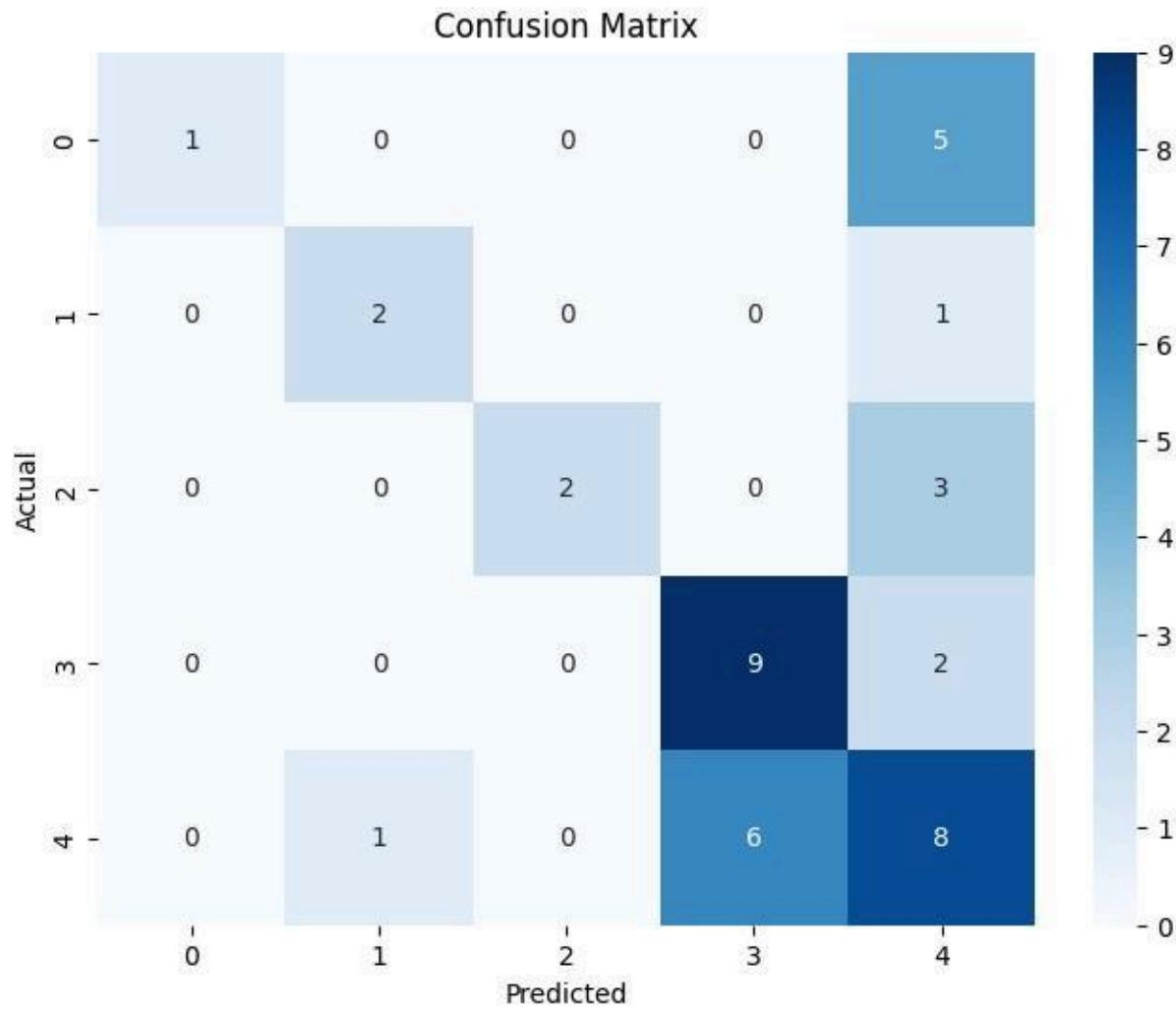
# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Create and display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

```

Accuracy: 0.55



In []:

```
import pandas as pd from sklearn.model_selection  
import train_test_split from sklearn.tree import  
DecisionTreeRegressor  
from sklearn.metrics import mean_absolute_error, mean_squared_error  
  
df = pd.read_csv('petrol_consumption.csv')  
  
# Split data into features (X) and target (y)  
X = df.drop('Petrol_Consumption', axis=1)  
y = df['Petrol_Consumption']  
  
# Split data into training and testing sets (80% train, 20% test)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```

# Initialize the DecisionTreeRegressor regressor
regressor = DecisionTreeRegressor()

# Train the regressor
regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = regressor.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")

```

Mean Absolute Error: 96.2

Mean Squared Error: 17858.2

Root Mean Squared Error: 133.63457636405332 **Program 4**

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Algorithm

Linear Regression and Multiple Linear Regression

Find linear regression of data of week and product
 x_i (week) y_i (Sales in thousands)

1 2
 2 4
 3 5

4 12.0 225.0 9 228.0 235.0 3 3

182.0 250.0 240.0 251.0

$$\beta = ((X^T X)^{-1} X^T) Y$$

$$\begin{array}{l} \begin{array}{|c|} \hline y_1 \\ \hline y_2 \\ \hline y_3 \\ \vdots \\ \hline y_n \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \ 2 \ x_1 \\ \hline 1 \ 2 \ x_2 \\ \hline 1 \ 2 \ x_3 \\ \vdots \\ \hline 1 \ 2 \ x_n \\ \hline \end{array} - \begin{array}{|c|} \hline e_1 \\ \hline e_2 \\ \hline e_3 \\ \vdots \\ \hline e_n \\ \hline \end{array} \end{array}$$

$$X^T = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \quad X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

$$Y^T = \begin{bmatrix} 2 & 4 & 5 & 9 \end{bmatrix}$$

$$X^T X$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

~~$$= \begin{bmatrix} 1+1+1+1 & 2+2+3+4 \\ 2+2+3+4 & 4+16+25+16 \end{bmatrix}$$~~

$$= \begin{bmatrix} 4 & 20 \\ 20 & 40 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 1.2115 & -0.192 \\ -0.192 & 0.0384 \end{bmatrix} \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

$$(X^T X)^{-1} X^T = \begin{bmatrix} 1.5 & -0.5 \\ -2.115 & -0.192 \\ -0.5 & 0.2 \\ -0.192 & 0.0389 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 3 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1.5 & -0.5 & -2.115 \\ -2.115 & -0.192 & 1.2115 - 4 \times 0.192 \\ -0.5 & 0.2 & 1.2115 + 5 \times 0.192 \end{bmatrix}$$

$$Z = \begin{bmatrix} 0.8275 & 0.4435 & 0.2515 & -0.5 \times 0.5 \\ -0.1192 & -0.0364 & 0.0025 & 0.1581 \end{bmatrix}$$

$$Z = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.2 \end{bmatrix}$$

$$ZY = Z \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$$= \begin{bmatrix} 0.038 \\ 1.0614 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

$$Y = \beta_0 + \beta_1 X$$

$$= -0.5 + 2.2 \times 5 \quad \text{for } n=5$$

$$= 10.5$$

Linear Regression

Sample code :

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df = pd.read_csv('canada_per-capita-income.csv')
plt.xlabel('year')
plt.ylabel('per capita income (US$)')
plt.scatter(df['year'], df['per-capita income (US$)'],
            color='red', marker='+')
new_df = df.drop('per capita income (US$)', axis='columns')
price = df['per capita income (US$)']
reg = linear_model.LinearRegression()
reg.fit(new_df, price)
print(reg.predict([[2020]]))
print(reg.coef_)
print(reg.intercept_)
```

Outputs

(I) Predicting per capita income for year 2020
 prediction: [91288.69409442]
 coeff: [828.46507522]
 intercept: -1632210.7578554575

(II) Predicting Salary of the employee
 prediction: [9398.64060184]
 intercept: 27197.2020175
 coeff: 139980.88923969

```

Multiple Linear Regression
import pandas as pd
import numpy as np
from sklearn import linear_model
df = pd.read_csv('1000-Companies.csv')
from sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
df['state'] = ord_enc.fit_transform(df[['state']])
reg = linear_model.LinearRegression()
reg.fit(df.drop('Profit', axis='columns'), df['Profit'])
print(reg.coef_)
print(reg.intercept_)
print(reg.predict([[71694.48, 515841.3, 11931.29]]))

```

Output

① Predicting salary
 $\text{coeff} = [0.55389123 \ 1.02672443 \ 0.0805 \ 46.623]$

Intercept = -70214.49

$\text{predict} = [511204.20332292]$

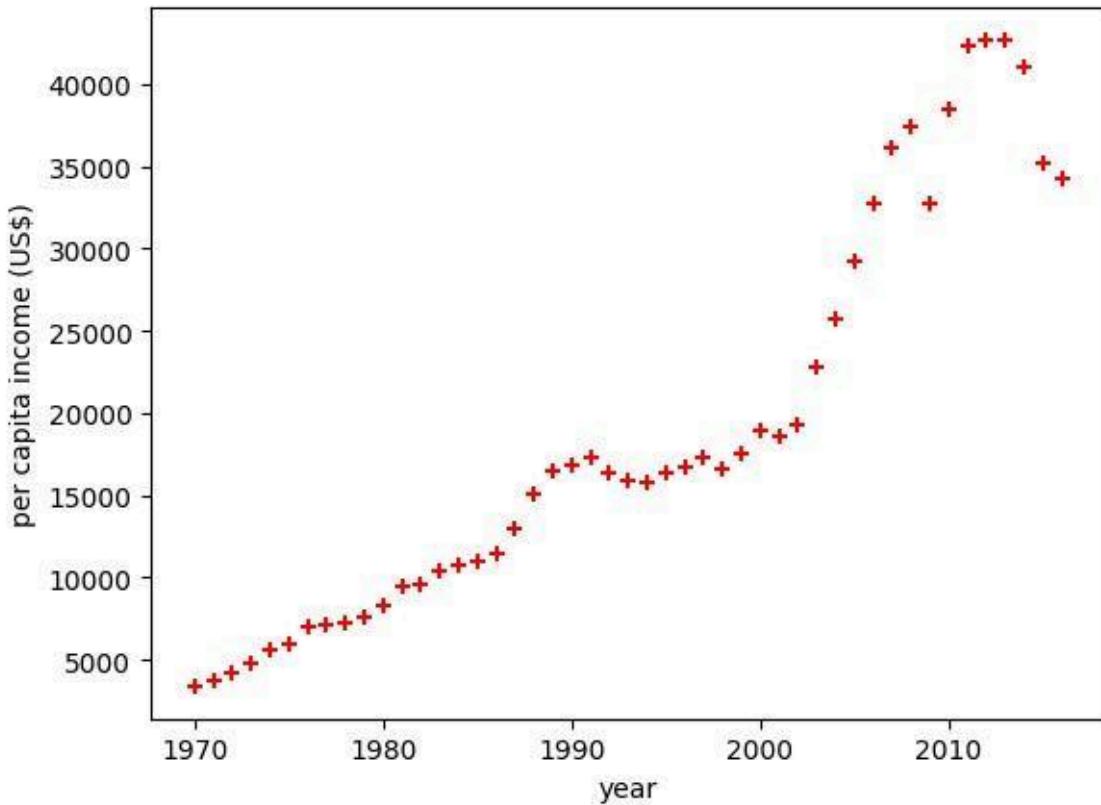
② Predicting profit

$\text{prediction} = [45968.51077949]$

$\text{prediction} = [92912.4732889]$

Code:

```
# -*- coding: utf-8 -*- #I.  
LINEAR_LR import pandas as pd  
import numpy as np from sklearn  
import linear_model  
import matplotlib.pyplot as plt  
  
df = pd.read_csv('canada_per_capita_income.csv') df  
  
# Commented out IPython magic to ensure Python compatibility.  
# %matplotlib inline plt.xlabel('year')  
plt.ylabel('per capita income (US$)')  
plt.scatter(df['year'],df['per capita income (US$)'],color='red',marker='+')  
  
new_df = df.drop('per capita income (US$)',axis='columns')  
new_df  
  
price = df['per capita income (US$)']  
  
# Create linear regression object reg =  
linear_model.LinearRegression()  
reg.fit(new_df,price)  
  
print(reg.predict([[2020]]))  
  
print(reg.coef_)  
  
print(reg.intercept_)  
  
"""Y = m * X + b (m is coefficient and b is intercept)"""  
  
[41288.69409442]  
[828.46507522]  
-1632210.7578554575  
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not  
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn(  
Out[31]:  
'Y = m * X + b (m is coefficient and b is intercept)'
```



In [15]:

```
# -*- coding: utf-8 -*-
LINEAR_LR import pandas as pd
import numpy as np from sklearn
import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('salary.csv')
df

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.scatter(df['YearsExperience'], df['Salary'], color='red', marker='+')

df['YearsExperience']=df['YearsExperience'].fillna(df['YearsExperience'].mean())
new_df = df.drop('Salary', axis='columns')
new_df

salary = df['Salary']
```

```

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,salary)

reg.predict([[12]])

print(reg.coef_)

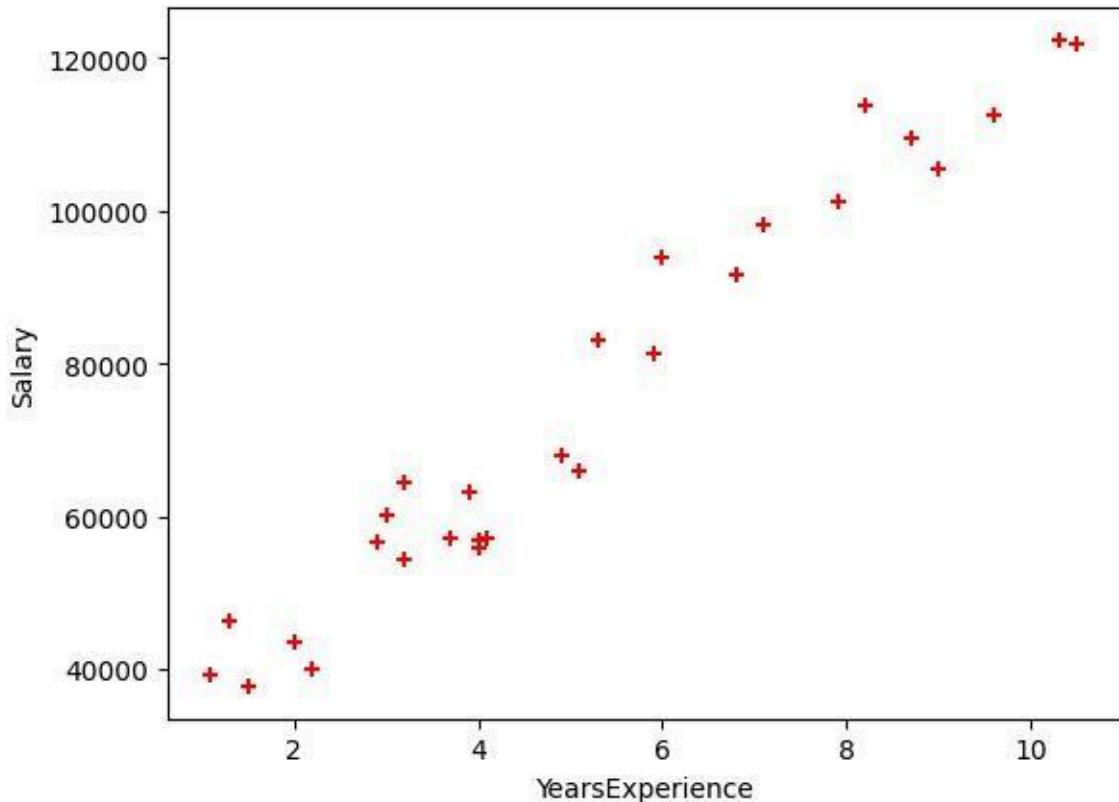
print(reg.intercept_)

"""Y = m * X + b (m is coefficient and b is intercept)"""

print(reg.predict([[12]]))

[9398.64060184]
27197.2020175696
[139980.88923969]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn(

```



In [17]:

```
!pip install word2number
```

Collecting word2number

 Downloading word2number-1.1.zip (9.7 kB)

 Preparing metadata (setup.py) ... done

Building wheels for collected packages: word2number

 Building wheel for word2number (setup.py) ... done

 Created wheel for word2number: filename=word2number-1.1-py3-none-any.whl size=5568
sha256=88e1bcbd080599c701a9fbaf1d8af7e1c4b020a078bd8566ebf1efa7804baf89

 Stored in directory:

/root/.cache/pip/wheels/cd/ef/ae/073b491b14d25e2efafcffca9e16b2ee6d114ec5c643ba4f06

Successfully built word2number

Installing collected packages:

word2number Successfully installed

word2number-1.1 In [20]:

```
#I MULTIPLE_LR
# -*- coding: utf-8 -*-
from word2number import w2n
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
df = pd.read_csv('hiring.csv')
df
```

"""Data Preprocessing: Fill NA values with median value of a column"""

```
df.experience = df.experience.bfill() #
df.experience = df.experience.fillna("zero")
# df
```

```
df.experience = df.experience.apply(w2n.word_to_num)
# df
df['test_score(out of 10)'] = df['test_score(out of 10)'].fillna(df['test_score(out of 10)'].mean())
# df
```

```
reg = linear_model.LinearRegression()
reg.fit(df.drop('salary($)',axis='columns'),df['salary($)'])
```

```
reg.coef_
```

```

reg.intercept_

print(reg.predict([[2, 9, 6]]))
print(reg.predict([[12, 10, 10]]))
[45968.51077949] [92712.4732389]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn( In
[32]:


#2. MULTIPLE_LR import
pandas as pd import
numpy as np
from sklearn import linear_model

df = pd.read_csv('1000_Companies.csv')

from sklearn.preprocessing import OrdinalEncoder ord_enc
= OrdinalEncoder()
df["State"] = ord_enc.fit_transform(df[["State"]])

reg = linear_model.LinearRegression()
reg.fit(df.drop('Profit',axis='columns'),df['Profit'])

print(reg.coef_)

print(reg.intercept_)

print(reg.predict([[91694.48, 515841.3, 11931.24, 1.0]]))

[ 0.55389023  1.02672443  0.08058525 46.62387015]
-70214.44175560221
[511209.20332292]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn( In
[28]:


pd.get_dummies(df, columns=["State"]).head() from
sklearn.preprocessing import OrdinalEncoder
ord_enc = OrdinalEncoder()
df["State"] = ord_enc.fit_transform(df[["State"]]) df.head()

```

Out[28]:

	R&D Spend	Administration	Marketing Spend	State Profit	state_code
0	165349.20	136897.80	471784.10	2.0	192261.83 2.0
1	162597.70	151377.59	443898.53	0.0	191792.06 0.0
2	153441.51	101145.55	407934.54	1.0	191050.39 1.0
3	144372.41	118671.85	383199.62	2.0	182901.99 2.0
4	142107.34	91391.77	366168.42	1.0	166187.94 1.0

In [3]:

```
# -*- coding: utf-8 -*-
"""Multiple_LR_HomePrice.ipynb
```

Automatically generated by Colab.

Original file is located at

```
https://colab.research.google.com/drive/1fK78C8TPV44HdvT6lsMhaau2wMtKXquQ
"""
```

```
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
df = pd.read_csv('homeprices_Multiple_LR.csv') df
```

```
"""Data Preprocessing: Fill NA values with median value of a column"""

```

```
df.bedrooms.median()
```

```
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df
```

```
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)
```

```
reg.coef_
```

```
reg.intercept_
```

```
"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

```

```
reg.predict([[3000, 3, 40]])  
112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not  
have valid feature names, but LinearRegression was fitted with feature names  warnings.warn(  
Out[3]:
```

```
498408.25157402386
```

```
In [4]:
```

```
# -*- coding: utf-8 -*  
"""/Linear-Regression-Housing_Area_Price.ipynb
```

Automatically generated by Colab.

Original file is located at

<https://colab.research.google.com/drive/1CALZml-P6V2V1RrodgMff8L3Ux4V9FT> """"

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

```
df = pd.read_csv('housing_area_price.csv') df
```

```
# Commented out IPython magic to ensure Python compatibility.  
# %matplotlib inline  
plt.xlabel('area') plt.ylabel('price')  
plt.scatter(df.area, df.price, color='red', marker='+')
```

```
new_df = df.drop('price', axis='columns')  
new_df
```

```
price = df.price price
```

```
# Create linear regression object reg =  
linear_model.LinearRegression()  
reg.fit(new_df, price)
```

```
""""(1) Predict price of a home with area = 3300 sqr ft""""
```

```
reg.predict([[3300]])
```

```
reg.coef_
```

```
reg.intercept_
```

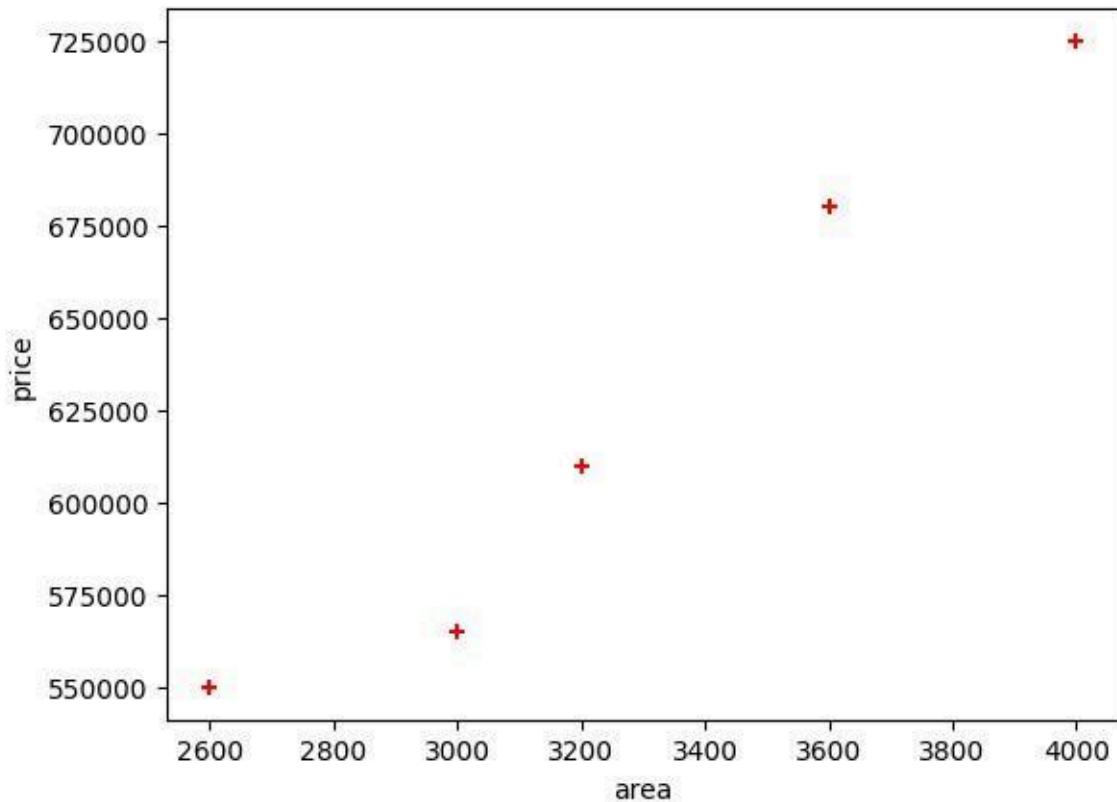
"""Y = m * X + b (m is coefficient and b is intercept)"""

3300*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

```
reg.predict([[5000]])
```

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
Out[4]: array([859554.79452055])



In []:

Program 5

Build Logistic Regression Model for a given dataset

Algorithm

LAB - 3

① $y = \begin{cases} 1 & \text{if } p(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

$\Rightarrow a_0 = 1, a_1 = 8, x = 60$

$$y = \frac{1}{1 + e^{-(1+60 \times 8)}} = \frac{1}{1 + e^{-481}} = 1$$

\therefore Pass

$\Rightarrow a = -5, a_1 = 0.8, x = 7$

$$y = \frac{1}{1 + e^{(-5 + 0.8 \times 7)}} = \frac{1}{1 + e^{-0.64}} = 0.645 > 0.5$$

pass

logistic Regression (using softmax)

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

suppose $z = [z_1, z_2]$

$$\text{softmax}(z_1) = \frac{e^2}{e^2 + e^1 + e^0} = 0.665$$

$$\text{softmax}(z_2) = \frac{e^1}{e^2 + e^1 + e^0} = 0.244$$

$$\text{softmax}(z_3) = \frac{e^0}{e^2 + e^1 + e^0} = 0.091$$

② $n = [5.1, 3.5, 1.4, 0.2]$

weights $w_0 = [0.5, -0.2, 0.1, 0.3]$, bias $b_0 = -1.0$

weights $w_1 = [-0.3, 0.4, -0.5, 0.2]$, $b_1 = 0.5$

~~$w_2 = [0.2, -0.1, 0.6, -0.4]$, $b_2 = 0.0$~~

~~$z_{02} = 1.05$~~

$$z_{1,2} (5.1 \times 0.5 + 3.5 \times 0.4 + 1.4 \times -0.5 + 0.2 \times 0.2) + 0.5$$

~~$z_2 = 1.43$~~

$$z_0 = w_0^T x + b_0$$

Setosa

$$z_1 = w_1^T x + b_1$$

Versicolor

$$z_2 = w_2^T x + b_2$$

Virginica

$$P(y=0|x) = \frac{e^{z_0}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.3671$$

$$P(y=1|x) = \frac{e^{z_1}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.0961$$

$$P(y=2|x) = \frac{e^{z_2}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.5368$$

Predict class.

$$\hat{y} = \arg\max \{ P(y=0|x), P(y=1|x), P(y=2|x) \}$$

$$= 0.5368 \Rightarrow \text{Virginica}$$

Logistic Regression - Binary

```

import pandas as pd
from matplotlib import pyplot as plt
df = pd.read_csv("insurance_data.csv")
df.head()
plt.scatter(df['age'], df['bought_insurance'], marker = '+', color = 'red')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['age'], df['bought_insurance'], train_size = 0.9, random_state = 10)
X_train.shape
X_test
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_pred = model.predict(X_test)
y_predicted
model.score(X_test, y_test)
model.predict_proba(X_test)
y_predicted_d = model.predict([[60]])
y_predict
model.coef_
model.intercept_
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
def prediction_function(age):
    z = 0.123 * age - 4.973
    y = sigmoid(z)
    return y
age = 3
prediction_function(age)

```

0.37 is less than 0.5 which means person with 35 will not buy the insurance

Logistic Regression: Multiclass

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn import metrics

import matplotlib.pyplot as plt

iris = pd.read_csv("iris.csv")

iris.head()

X = iris.drop('species', axis=1)

y = iris['species']

X-train, X-test, y-train, y-test = train_test_split(X, y, test_size=0.2,

random_state=42)

model = LogisticRegression(multi_class='multinomial')

model.fit(X-train, y-train)

y-pred = model.predict(X-test)

accuracy = accuracy_score(y-test, y-pred)

print(f"Accuracy of the Multinomial logistic Regression

model on the test set: {accuracy:.2f}%")

confusion_matrix = confusion_matrix(y-test, y-pred)

sns. heatmap(conf_matrix, annot=True, fmt=".1f", cmap="Blues")

clrs = Pallete)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

Accuracy of logistic Regression = 84.5%

HR-comm-a-sep.PY

- (i) Salary and work-satisfaction have a direct impact on employee retention because as per the dataset, employees who have low salary & low satisfaction level have low retention.
 - (ii) The accuracy of the logistic regression model is 79.29%.

zoo-data.csv

- 300-data.csv

 - (i) The most frequently misclassified label is mammal according to the confusion matrix
 - (ii) accuracy of the model = 95.24%.
 - (iii) The confusion matrix tells that it mostly accurate

~~100~~ ~~12~~ x 25

Code:

In []:

```
# -*- coding: utf-8 -*-
"""LogisticRegression_Binary.ipynb
```

Automatically generated by Colab.

Original file is located at

```
https://colab.research.google.com/drive/1M8PXdcmPsrQtqyVXpET3sgghAMr_McG5
"""
```

```
# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
from matplotlib import pyplot as plt
# %matplotlib inline
# "%matplotlib inline" will make your plot outputs appear and be stored within the notebook.

df = pd.read_csv("insurance_data.csv") df.head()

plt.scatter(df.age, df.bought_insurance, marker='+', color='red')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df[['age']], df.bought_insurance, train_size=0.9, random_state=10) X_train.shape

X_test

from sklearn.linear_model import LogisticRegression model
= LogisticRegression()

model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test, y_test)
```

```

model.predict_proba(X_test)

y_predicted = model.predict([[60]]) y_predicted

#model.coef_ indicates value of m in y=m*x + b equation model.coef_

#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with
hand import math def sigmoid(x): return 1 / (1 +
math.exp(-x))

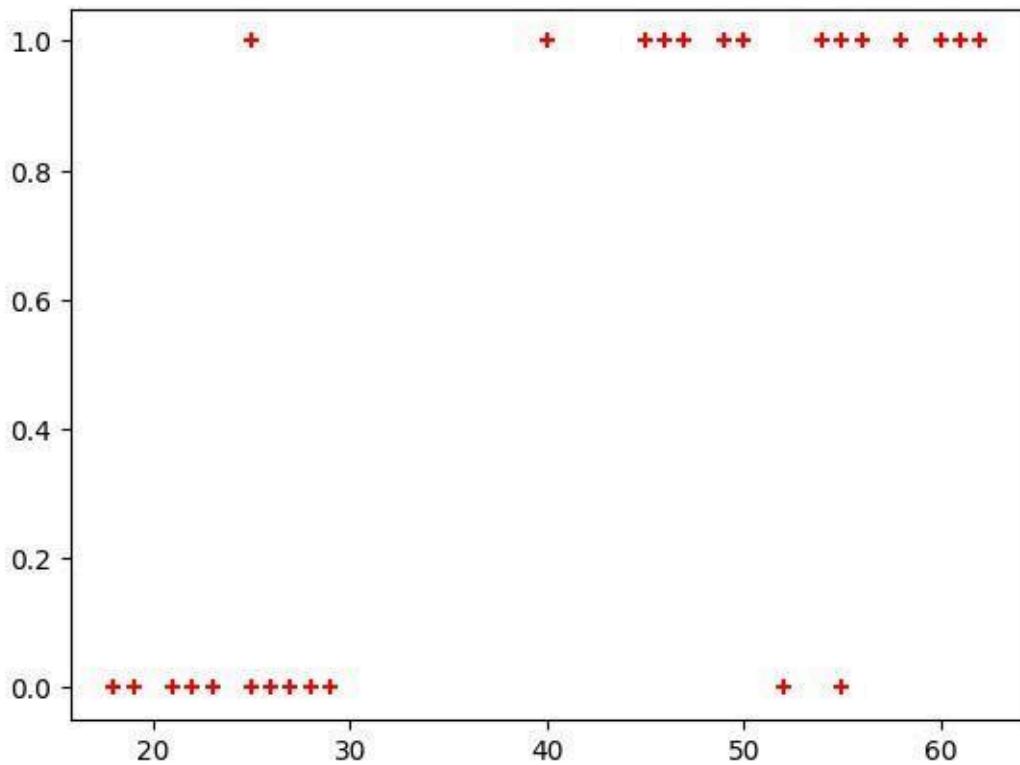
def prediction_function(age):
    z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
    y = sigmoid(z)
    return y

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but LogisticRegression was fitted with feature names
warnings.warn(
Out[ ]:
'0.37 is less than 0.5 which means person with 35 will not buy the insurance'

```



In []:

```
#-*- coding: utf-8 -*-
"""/LogisticRegression_Multiclass.ipynb
```

Automatically generated by Colab.

Original file is located at

```
https://colab.research.google.com/drive/1anBybVXILenh0a_R4aM_ZemLrEqYWnJl """
```

```
# Import necessary libraries import
pandas as pd
from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression from
sklearn.metrics import accuracy_score from sklearn
import metrics
import matplotlib.pyplot as plt
```

```
# Load the Iris dataset iris =
pd.read_csv("iris.csv")
iris.head()
```

```

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width) y
= iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver model
= LogisticRegression(multi_class='multinomial')

# Train the model on the training data model.fit(X_train,
y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

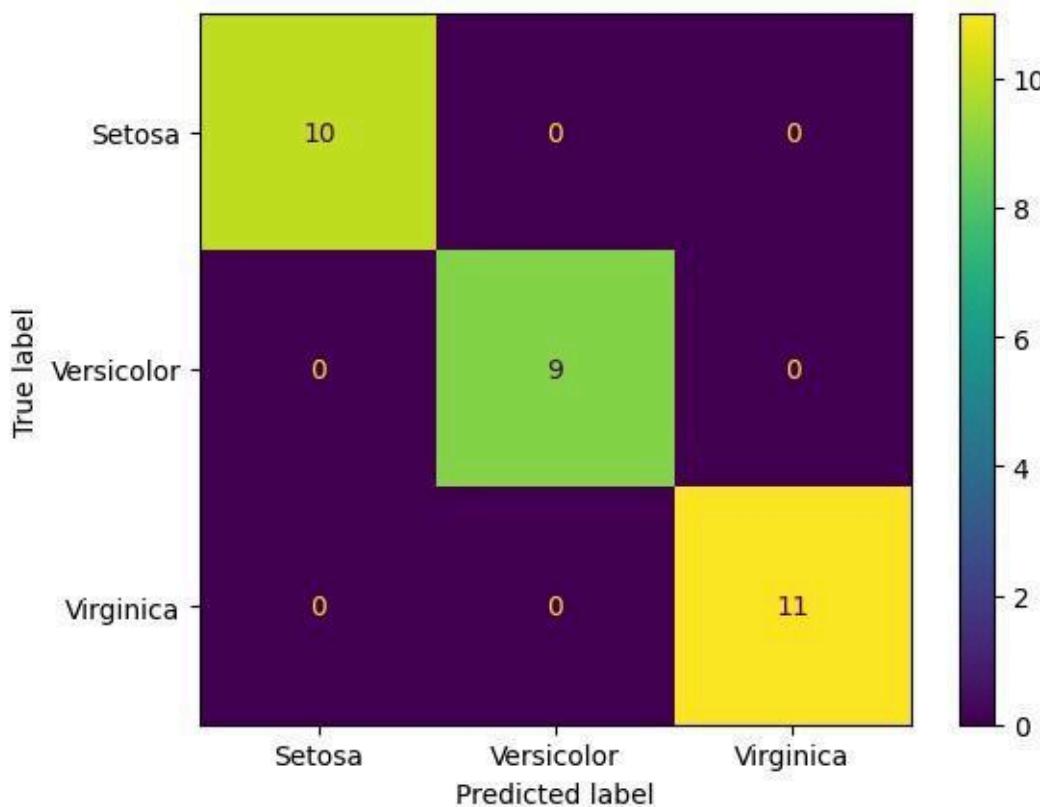
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels
= ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()

Accuracy of the Multinomial Logistic Regression model on the test set: 1.00
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning:
'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this warning.
warnings.warn(

```



In []:

```
# Importing necessary libraries import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset df=
df=pd.read_csv('HR_comma_sep.csv')

# 1. Exploratory Data Analysis (EDA)

# Check the first few rows of the dataset print(df.head())

# Check for missing values
print(df.isnull().sum())

# Describe the dataset (summary statistics) print(df.describe())

# Check the distribution of 'left' column (Employee Retention: 1 means left, 0 means stayed)
sns.countplot(x='left', data=df)
plt.title("Employee Retention (Stayed vs Left)") plt.show()
```

```

# 2. Impact of Employee Salaries on Retention
# Plotting bar chart showing impact of employee salaries on retention
sns.countplot(x='salary', hue='left', data=df)
plt.title("Impact of Employee Salaries on Retention")
plt.show()

# 3. Correlation between Department and Employee Retention
# Plotting bar chart showing correlation between department and retention
sns.countplot(x='Department', hue='left', data=df) plt.title("Correlation
Between Department and Employee Retention") plt.xticks(rotation=45) #
Rotate x-axis labels for better readability plt.show()

# 4. Build Logistic Regression Model

# Convert categorical columns to numeric using one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)

# Split the dataset into features (X) and target (y)
X = df_encoded.drop('left', axis=1) # Features (drop the 'left' column as it's the target) y
= df_encoded['left'] # Target: whether the employee left (1) or stayed (0)

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
logreg_model = LogisticRegression()

# Fit the model to the training data
logreg_model.fit(X_train, y_train)

# 5. Measure Model Accuracy

# Predict on the test set
y_pred = logreg_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression Model: {accuracy * 100:.2f}%")

# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False) plt.title("Confusion
Matrix") plt.xlabel("Predicted") plt.ylabel("Actual")

```

```
plt.show()
```

```
satisfaction_level last_evaluation number_project average_montly_hours \
0      0.38        0.53        2        157
1      0.80        0.86        5        262
2      0.11        0.88        7        272
3      0.72        0.87        5      223  4        0.37        0.52        2
159

time_spend_company Work_accident left promotion_last_5years Department \
0      3        0  1        0    sales
1      6        0  1        0    sales
2      4        0  1        0    sales
3      5        0  1        0    sales  4        3        0  1        0
sales

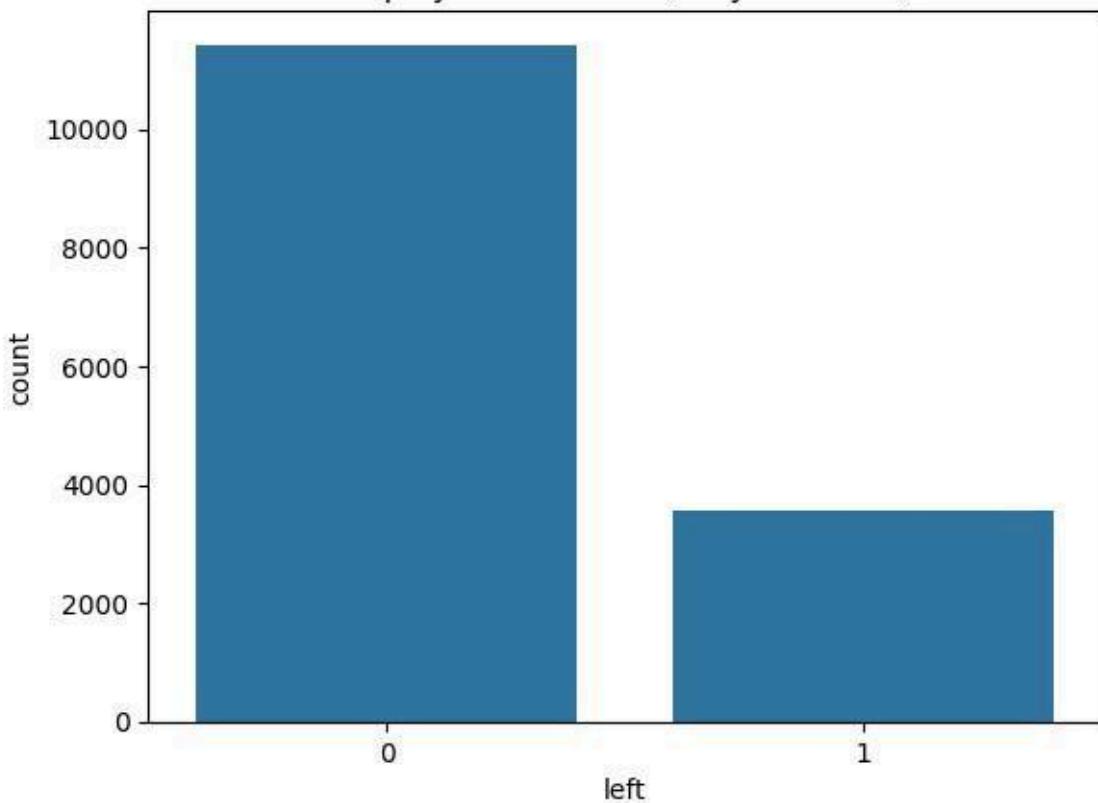
salary 0
low
1 medium
2 medium
3 low 4 low satisfaction_level 0 last_evaluation 0 number_project 0
average_montly_hours 0 time_spend_company 0 Work_accident 0 left 0
promotion_last_5years 0 Department
0
salary      0 dtype: int64      satisfaction_level
last_evaluation number_project \ count
14999.000000 14999.000000 14999.000000  mean
0.612834  0.716102  3.803054  std
0.248631  0.171169  1.232592  min
0.090000  0.360000  2.000000  25%
0.440000  0.560000  3.000000
50%       0.640000  0.720000  4.000000
75%       0.820000  0.870000  5.000000
max       1.000000  1.000000  7.000000
average_montly_hours time_spend_company
Work_accident left \ count 14999.000000
14999.000000 14999.000000 14999.000000  mean
201.050337  3.498233  0.144610  0.238083
std       49.943099  1.460136  0.351719
0.425924  min       96.000000  2.000000
0.000000  0.000000  25%       156.000000
3.000000  0.000000  0.000000
```

50%	200.000000	3.000000	0.000000	0.000000
75%	245.000000	4.000000	0.000000	0.000000
max	310.000000	10.000000	1.000000	1.000000

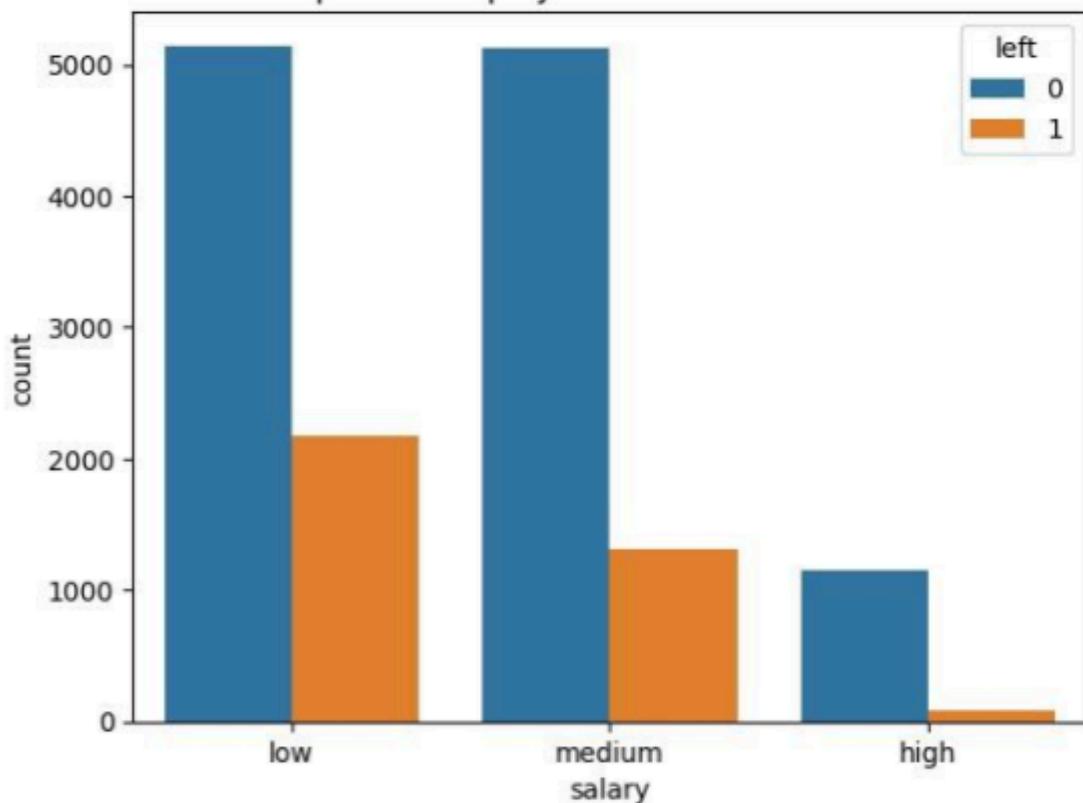
promotion_last_5years

count	14999.000000
mean	0.021268
std	0.144281
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

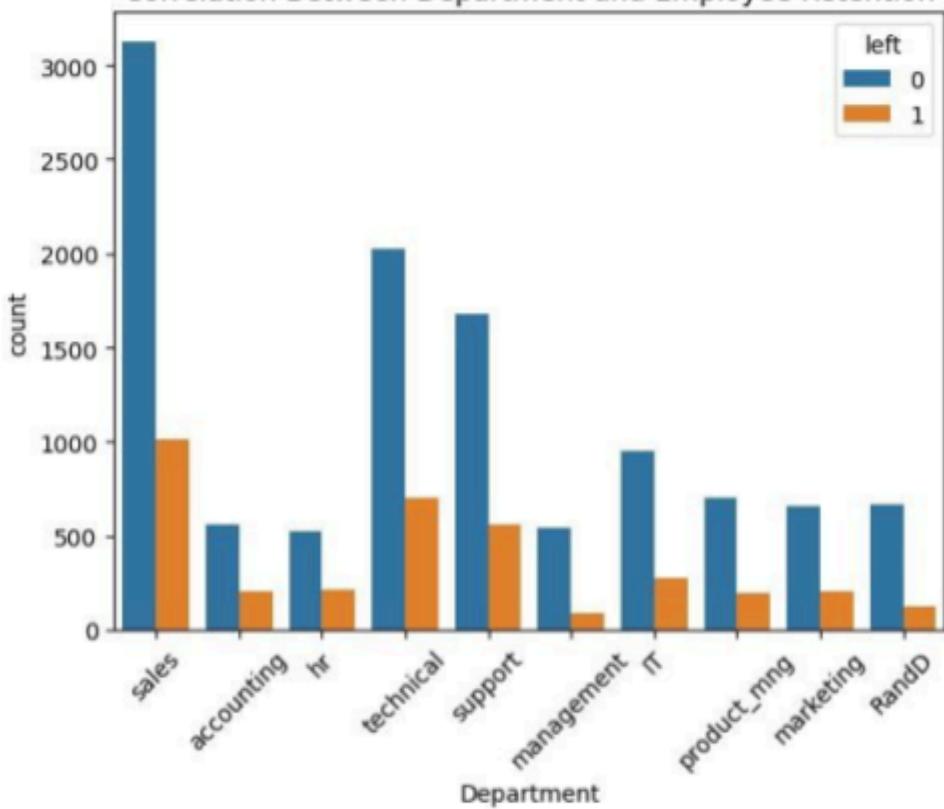
Employee Retention (Stayed vs Left)



Impact of Employee Salaries on Retention

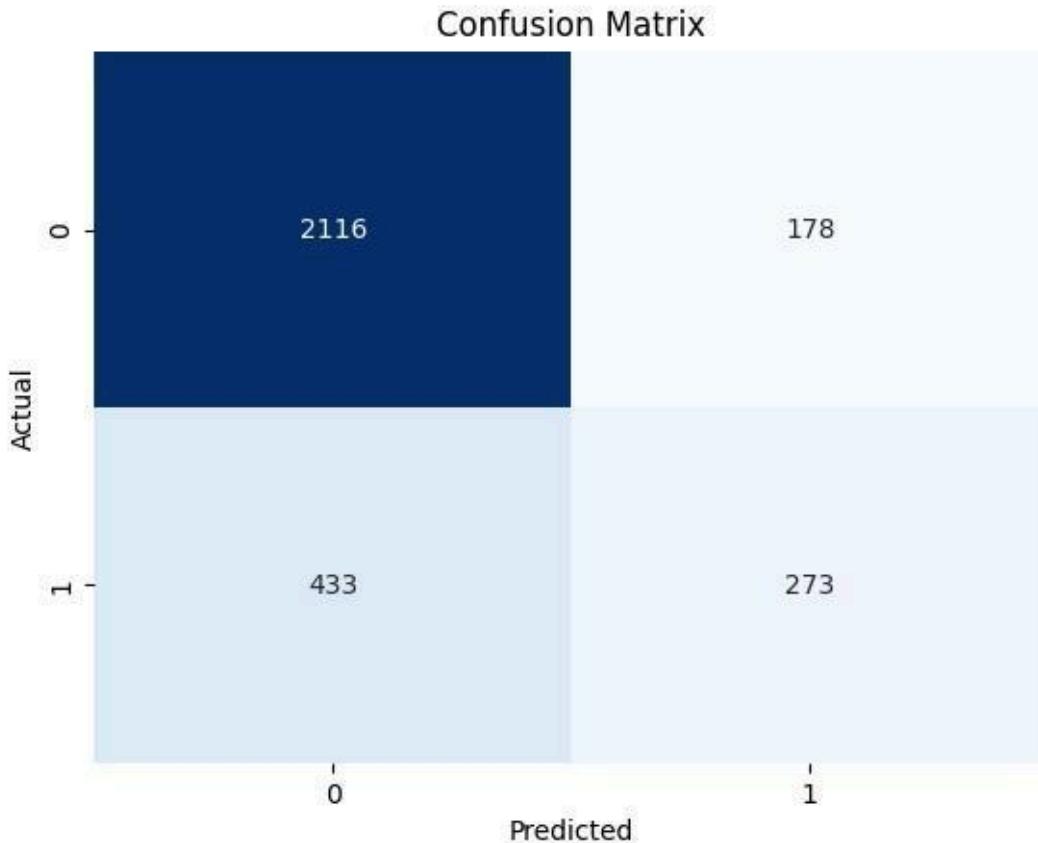


Correlation Between Department and Employee Retention



Accuracy of Logistic Regression Model: 79.63%
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html> Please also refer to
the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result



In []:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
```

```
# Load the datasets
zoo_data =
pd.read_csv('zoo-data.csv')
```

```

class_type = pd.read_csv('zoo-class-type.csv')

# 1. Data Preprocessing: Merge datasets to get class_type
zoo_data = zoo_data.merge(class_type[['Class_Number', 'Class_Type']], how='left',
left_on='class_type', right_on='Class_Number')

# Drop the unnecessary 'Class_Number' column and keep 'Class_Type' as the target variable
zoo_data.drop(columns=['Class_Number'], inplace=True)

# Check for missing values
print(zoo_data.isnull().sum())

# Since we don't have missing values, we can proceed to encode categorical variables and model building.

# 2. Feature Engineering: Encode categorical variables
# Here, we'll encode 'class_type' which is now 'Class_Type' into numeric labels using LabelEncoder
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Convert 'Class_Type' (target) into numerical labels
zoo_data['Class_Type'] = label_encoder.fit_transform(zoo_data['Class_Type'])

# 3. Split the data into features (X) and target (y)
X = zoo_data.drop(['animal_name', 'class_type', 'Class_Type'], axis=1) # Features
y = zoo_data['Class_Type'] # Target

# 4. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Build Logistic Regression Model
logreg_model = LogisticRegression(max_iter=1000, multi_class='ovr') # Using 'ovr' for multiclass classification
logreg_model.fit(X_train, y_train)

# 6. Predictions and Accuracy
y_pred = logreg_model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Logistic Regression Model: {accuracy * 100:.2f}%')

# 7. Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

```

```
# Plotting Confusion Matrix sns.heatmap(conf_matrix, annot=True,
fmt="d", cmap="Blues", cbar=False, xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_) plt.title("Confusion Matrix for Logistic
Regression Model") plt.xlabel("Predicted") plt.ylabel("Actual") plt.show()
```

```
animal_name    0
hair          0
feathers      0
eggs          0
milk          0
airborne      0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
fins           0
legs          0 tail
0
domestic      0 catsize
0 class_type   0
Class_Type     0
dtype: int64
```

Accuracy of Logistic Regression Model: 95.24%

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning:
'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use
OneVsRestClassifier(LogisticRegression(..)) instead. Leave it to its default value to avoid this
warning.  warnings.warn(
```

Confusion Matrix for Logistic Regression Model

	Amphibian	Bird	Bug	Fish	Invertebrate	Mammal	Reptile
Actual	2	0	0	0	0	0	
Reptile							-
Amphibian -	2	0	0	0	0	0	
Bird -	0	3	0	0	0	0	
Bug -	0	0	2	0	0	0	
Fish -	0	0	0	1	0	0	
Invertebrate -	0	0	0	0	12	0	
Mammal -	0	0	1	0	0	0	

In []:

Program 6

Build KNN Classification model for a given dataset.

Algorithm

→ Consider the following dataset, for $k=3$ and test data $(X, 35, 100)$ as $(\text{Person } \& \text{ Pos}, \text{Age}, \text{Salary } k)$ solve using KNN classifier model & predict the target.

Person	Age	Salary k	Target dist	Rank
A	18	50	N	52.81
B	23	55	N	76.57
C	24	70	N	81.95
D	41	60	Y	40.49
E	43	70	Y	31.09
F	38	90	Y	60.07
X	35	100	?	

K = 3

Rank 1 = Y

Rank 2 = N

Rank 3 = Y

The target predicted is Y

Code:

import pandas as pd

from sklearn.model_selection import train-test-split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report

confusion_matrix, accuracy_score

from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data

y = iris.target

$X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}} = \text{train-test-split}(X, y)$
 test_size = 0.2, random_state = 42
 knn = KNeighborsClassifier(n_neighbours=3)
 knn.fit(X_train, y_train)
 y_pred = knn.predict(X_test)
 accuracy = accuracy_score(y_true, y_pred)
 conf_matrix = confusion_matrix(y_true, y_pred)
 class_report = classification_report(y_true, y_pred)
 print(f"Accuracy: {accuracy}")
 print(f"Confusion Matrix: \n{conf_matrix}")
 print(f"Classification Report: \n{class_report}")

Output:

Accuracy: 1.0

Confusion Matrix:

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

Classification Report

	precision	recall	f1-score	support
0	1.0	1.00	1.00	10
1	1.0	1.00	1.00	9
2	1.0	1.00	1.00	11

accuracy

Macro avg

Weighted avg

1.0 1.0 1.0 30

1.0 1.0 1.0 30

1.0 1.0 1.0 30

Q1) Looping from 1 to 20 & choose the value which has max accuracy

Q2) To normalize or standardize the range of values for different features to ensure that no single feature dominates the learning process due to its scale, which can improve model performance

Code

IRIS DATASET

In [3]:

```
import pandas as pd
```

```
# Assuming the uploaded file is named 'iris.csv' file_path  
= 'iris.csv' # The path to your uploaded file
```

```
# Load the dataset into a pandas DataFrame  
df = pd.read_csv(file_path)
```

```
# Display the first few rows to ensure the dataset loaded correctly  
print(df.head())
```

```
from sklearn.model_selection import train_test_split from  
sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report from  
sklearn.preprocessing import LabelEncoder
```

```
# Assuming the dataset has columns: sepal_length, sepal_width, petal_length, petal_width, species  
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] # Features y  
= df['species'] # Target label
```

```
# Encode the target labels (species) if needed le  
= LabelEncoder()  
y = le.fit_transform(y)
```

```
# Split the data into 80% training and 20% testing  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create and train the KNN model knn =  
KNeighborsClassifier(n_neighbors=5)  
knn.fit(X_train, y_train)
```

```
# Make predictions on the test set  
y_pred = knn.predict(X_test)
```

```
# Evaluate the model  
accuracy = accuracy_score(y_test, y_pred)  
conf_matrix = confusion_matrix(y_test, y_pred)  
class_report = classification_report(y_test, y_pred)
```

```
# Print results
print("Accuracy:", accuracy) print("Confusion
Matrix:\n", conf_matrix) print("Classification
Report:\n", class_report)
```

```
sepal_length sepal_width petal_length petal_width species
0      5.1       3.5       1.4       0.2 setosa
1      4.9       3.0       1.4       0.2 setosa
2      4.7       3.2       1.3       0.2 setosa
3      4.6       3.1       1.5       0.2 setosa
4      5.0       3.6       1.4       0.2 setosa
```

Accuracy: 1.0 Confusion

Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy		1.00		30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

DIABETES PREDICTION

In [4]:

```
import pandas as pd import
numpy as np
```

In [5]:

```
data = pd.read_csv("diabetes.csv") data.head()
```

Out[5]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
-------------	---------	---------------	---------------	---------	-----	--------------------------	-----	---------

0	6	148	72	35	0	33. 6	0.627	50	1
1	1	85	66	29	0	26. 6	0.351	31	0
Pregnan	Gluc	BloodPres	SkinThic	Insu	B	DiabetesPedigree	A	Outco	
cies	ose	sure	kness	lin	MI	Function	ge	me	
2	8	183	64	0	0	0.672	32	1	23. 3
3	1	89	66	23	94	0.167	21	0	28. 1
4	0	137	40	35	168	2.288	33	1	43. 1

In [20]:

```
from sklearn.preprocessing import StandardScaler sc_X
= StandardScaler()
X = pd.DataFrame(sc_X.fit_transform(data_copy.drop(["Outcome"], axis
=1,)),columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI',
'DiabetesPedigreeFunction', 'Age'])
```

In [21]:

```
X.head()
```

Out[21]:

	Pregnant BMI	Glucose Age	BloodPressure Cies	SkinThickness Se	Insulin Sure	DiabetesPedigreeFunction Thickness	
0	0.63994	0.865		-0.033518	0.670643	0.181	0.166
	7		108			0.468492	1.425
					541	619	995
	-	-	-	-	-	-	-

1		0.84488	1.206	-0.529859	-0.012301	0.181	0.852	-0.365061
		0.190						
5	162			541	200			672
				-	-			-
	1.23388	2.015						
2		-0.695306	-0.012301	0.181	1.332	0.604397	0.105 0	813
				541	500			584
	-	-		-	-		-	-
3		0.84488	1.074	-0.529859	-0.695245	0.540	0.633	-0.920763
		1.041						
5	652			642	881			549
	-	0.503		0.316	1.549			-
4		1.14185	-2.680669	0.670643	5.484909	0.020		
	458			566	303			
	2							496

In [22]:

```
y = data_copy.Outcome
```

In [23]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 42, stratify=y)
```

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier
```

```
train_scores = []
test_scores = []
```

```
for i in range(1,15):    knn =
    KNeighborsClassifier(i)    knn.fit(X_train,
    y_train)    train_scores.append(knn.score(X_train,
    y_train))    test_scores.append(knn.score(X_test,
    y_test))
```

In [25]:

```
max_test_score = max(test_scores)
```

In [26]:

```
test_score_index = [i for i, v in enumerate(test_scores) if v == max_test_score]
```

```
print('Max test score {} % and k = {}'.format(max_test_score * 100, list(map(lambda x: x+1, test_score_index))))
```

Max test score 76.5625 % and k = [11] In
[28]:

```
# K=11
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

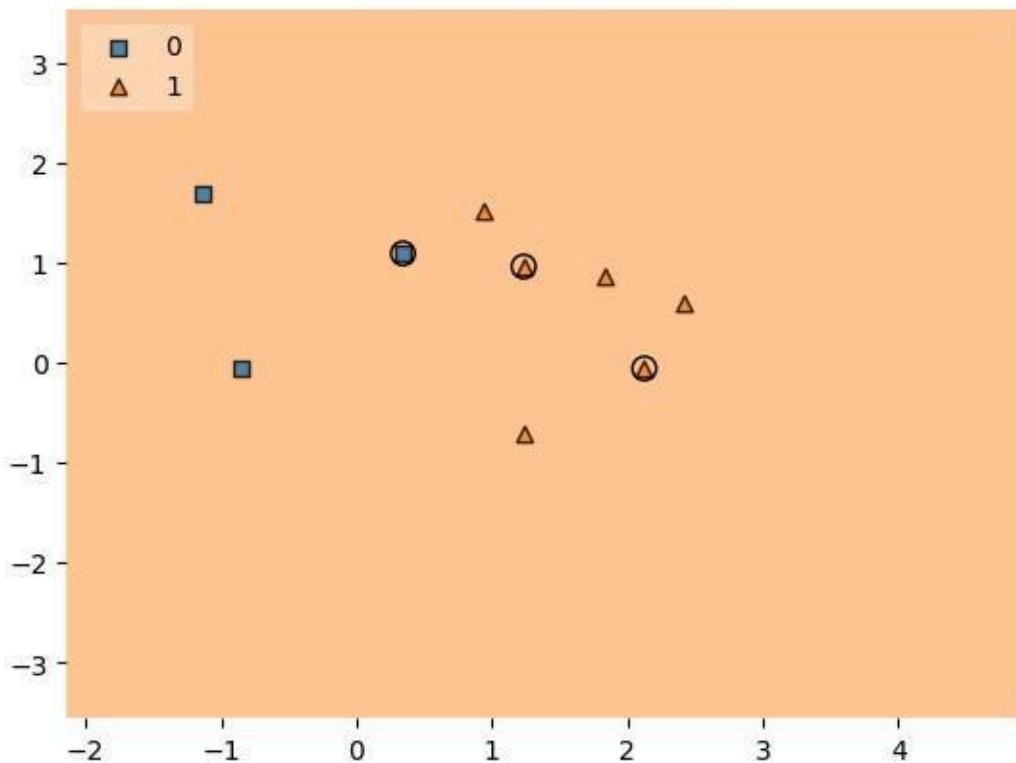
Out[28]:

```
0.765625 In
[29]: from
mlxtend.plo
tting import
plot_decisi
on_regions
value =
20000
width =20000
```

```
plot_decision_regions(X.values, y.values, clf=knn, legend=2,filler_feature_values={2: value, 3:
value, 4: value, 5: value, 6: value, 7: value},
filler_feature_ranges={2: width, 3: width, 4: width, 5: width, 6: width, 7: width},
X_highlight=X_test.values)
plt.title("KNN with diabetes data") plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature names
warnings.warn(
```

KNN with diabetes data



In [30]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, fbeta_score
y_pred = knn.predict(X_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
```

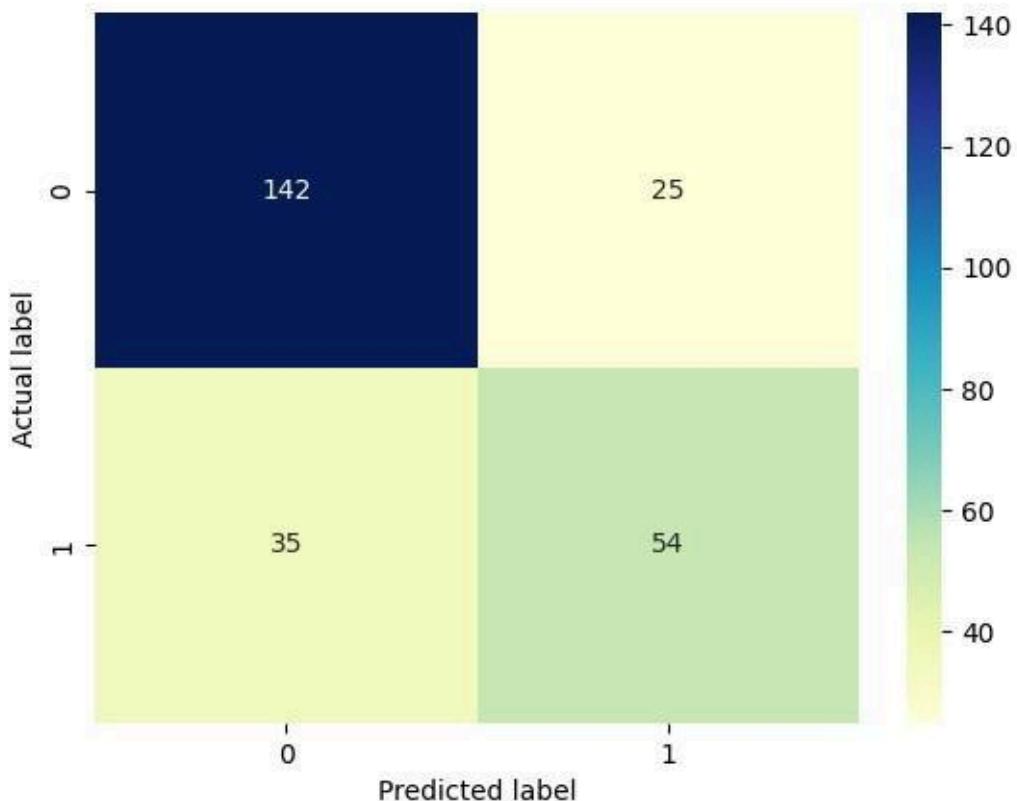
In [31]:

```
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[31]:

```
Text(0.5, 23.52222222222222, 'Predicted label')
```

Confusion matrix



In [32]:

```
def model_evaluation(y_test, y_pred, model_name):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta = 2.0)

    results = pd.DataFrame([[model_name, acc, prec, rec, f1, f2]],
    columns = ["Model", "Accuracy", "Precision", "Recall",
               "F1 Score", "F2 Score"])
    results = results.sort_values(["Precision", "Recall", "F2 Score"], ascending = False)
    return results
```

```
model_evaluation(y_test, y_pred, "KNN")
```

Out[32]:

Model	Accuracy	Precision	Recall	F1 Score	F2 Score
0 KNN	0.765625	0.683544	0.606742	0.642857	0.62069

HEART DISEASE PREDICTION

In [62]:

```

import pandas as pd import matplotlib.pyplot as plt
import seaborn as sns from sklearn.model_selection
import train_test_split from sklearn.neighbors
import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score from
sklearn.preprocessing import StandardScaler

# Load the dataset df =
pd.read_csv('heart.csv')

# Define features (X) and target (y) X
= df.drop('target', axis=1)
y = df['target']

# Split data 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)

# Feature scaling scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the best k value
best_k = 1 best_accuracy
= 0
for k in range(1, 31): # Test k values from 1 to 30
knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train) y_pred =
    knn.predict(X_test) accuracy =
    accuracy_score(y_test, y_pred) if
    accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

print(f'Best k: {best_k}')

# Train the KNN classifier with the best k value knn
= KNeighborsClassifier(n_neighbors=best_k)

```

```

knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model accuracy =
accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display results print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")

# Plot confusion matrix plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disease', 'Disease'],
            yticklabels=['No Disease', 'Disease'])
plt.xlabel('Predicted') plt.ylabel('Actual')
plt.title('Confusion Matrix') plt.show()

# Plot classification report (text-based, can't be plotted directly)
print("Classification Report:\n", class_report)

```

Best k: 7

Accuracy: 0.9180327868852459 Confusion

Matrix:

`[[27 2]`

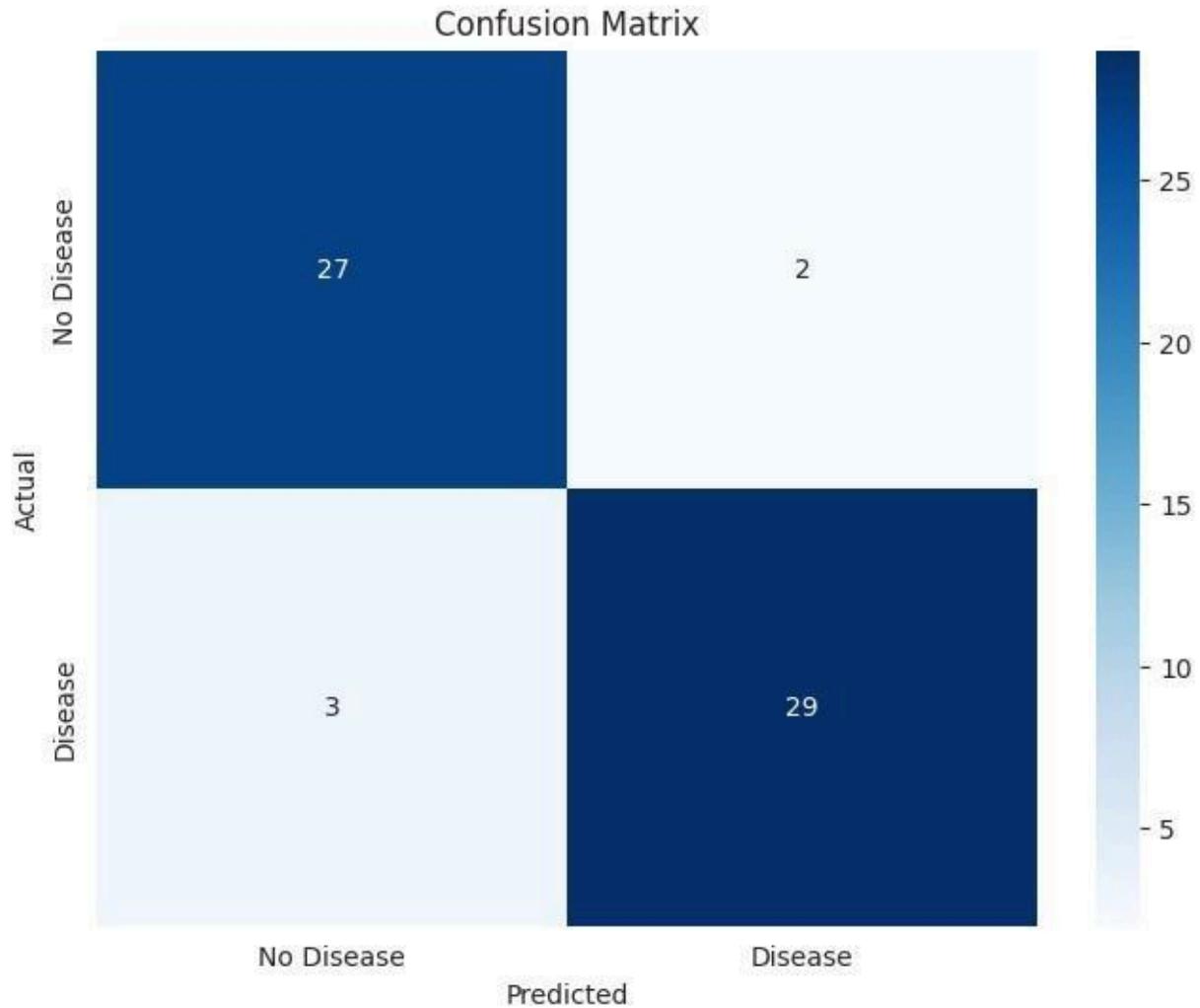
`[3 29]]`

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.90	0.93	0.92	29
1	0.94	0.91	0.92	32

	accuracy		0.92	61
macro avg	0.92	0.92	0.92	61
weighted avg	0.92	0.92	0.92	61



Classification Report:

precision recall f1-score support

0	0.90	0.93	0.92	29
1	0.94	0.91	0.92	32

accuracy		0.92	61	
macro avg	0.92	0.92	0.92	61
weighted avg	0.92	0.92	0.92	61

In []:

Program 7

Build Support vector machine model for a given dataset

Algorithm

LAB-7

PAGE NO : 08 / 4 / 25
DATE :

Q. Draw an optimal hyperplane using linear SVM to classify the following points.

Positive labels: $(1, 1), (1, -1), (-1, -1)$
Negative labels: $(4, 0), (5, 1), (5, -1), (6, 0)$

$S_1 = 1, 1$
 $S_2 = 2, -1$
 $S_3 = 4, 0$

$\tilde{S}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}, \tilde{S}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}, \tilde{S}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$\tilde{S}_4 = (\tilde{x}_1 \tilde{s}_1 + \tilde{x}_2 \tilde{s}_2 + \tilde{x}_3 \tilde{s}_3) = 1$
 $\tilde{S}_5 = (\tilde{x}_1 \tilde{s}_1 + \tilde{x}_2 \tilde{s}_2 + \tilde{x}_3 \tilde{s}_3) = 1$
 $\tilde{S}_6 = (\tilde{x}_1 \tilde{s}_1 + \tilde{x}_2 \tilde{s}_2 + \tilde{x}_3 \tilde{s}_3) = 1$
 $\tilde{S}_7 = (\tilde{x}_1 \tilde{s}_1 + \tilde{x}_2 \tilde{s}_2 + \tilde{x}_3 \tilde{s}_3) = 1$

~~$\tilde{x}_1 + 4\tilde{x}_2 + 9\tilde{x}_3 = 1$~~ $\alpha_1 = 13/4$
 ~~$4\tilde{x}_1 + 6\tilde{x}_2 + 9\tilde{x}_3 = 1$~~ $\alpha_2 = 13/4$
 ~~$9\tilde{x}_1 + 9\tilde{x}_2 + 17\tilde{x}_3 = 1$~~ $\alpha_3 = -3/2$

$W = \alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3$
 $= \frac{13}{4} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} + \frac{13}{4} \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} - \frac{3}{2} \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$W = \begin{bmatrix} -1 \\ 0 \\ 13 \end{bmatrix} \Rightarrow W = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$b = -3$
 $b + 3 = 0$

```

code
import pandas as pd
from sklearn.ensemble import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
columns = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
           'species']
iris = pd.read_csv('iris.csv', header=None, names=columns)
iris['species'] = iris['species'].map({0: 'Iris-setosa', 1: 'Iris-versicolor',
                                       2: 'Iris-virginica'})
X = iris.drop(['species'], axis=1)
y = iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
conf_matrix_linear = confusion_matrix(y_test, y_pred_linear)
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
conf_matrix_rbf = confusion_matrix(y_test, y_pred_rbf)
print(f"Accuracy of SVM with Linear kernel: {accuracy_linear}")
print(f"Accuracy of SVM with RBF kernel: {accuracy_rbf}")
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
sns.heatmap(conf_matrix_linear, annot=True, fmt="d", ax=axes[0])
sns.heatmap(conf_matrix_rbf, annot=True, fmt="d", ax=axes[1])

```

plt-show

Output:

Accuracy of SVM with Linear Kernel: 1.00

Accuracy of SVM with RBF Kernel: 1.00

Confusion Matrix - Linear

0	1	2
0	10	0
1	0	9
2	0	0

① Linear Kernel accuracy: 1.00

RBF Kernel accuracy: 1.00

Since the dataset is quite simple the both datasets performed equally well.

② Overall accuracy is 85.45%.

Confusion matrix:

C, Q, O or I, J, L, T show more confusion among each other. These pairs are visually similar, which likely leads the model to classify them.

Average AUC score: 0.987

It's a good performance across all ^{letters in} datasets

Code:

In [23]:

```
# Import necessary libraries
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# URL of the Iris dataset (adjust this URL if needed)
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Column names as the dataset contains headers now
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

# Load the dataset from the URL into a pandas DataFrame
iris = pd.read_csv(url, header=None, names=columns)

# Map the species names to numeric values (for classification)
iris['species'] = iris['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

# Split features and target X =
iris.drop('species', axis=1)
y = iris['species']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train SVM with a linear kernel
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)

# Predict and calculate accuracy for the linear kernel model
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
conf_matrix_linear = confusion_matrix(y_test, y_pred_linear)

# Initialize and train SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)
```

```

# Predict and calculate accuracy for the RBF kernel model
y_pred_rbf = svm_rbf.predict(X_test) accuracy_rbf =
accuracy_score(y_test, y_pred_rbf) conf_matrix_rbf =
confusion_matrix(y_test, y_pred_rbf)

# Display accuracy scores
print(f"Accuracy of SVM with Linear Kernel: {accuracy_linear:.2f}")
print(f"Accuracy of SVM with RBF Kernel: {accuracy_rbf:.2f}")

# Plot confusion matrices for both models
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

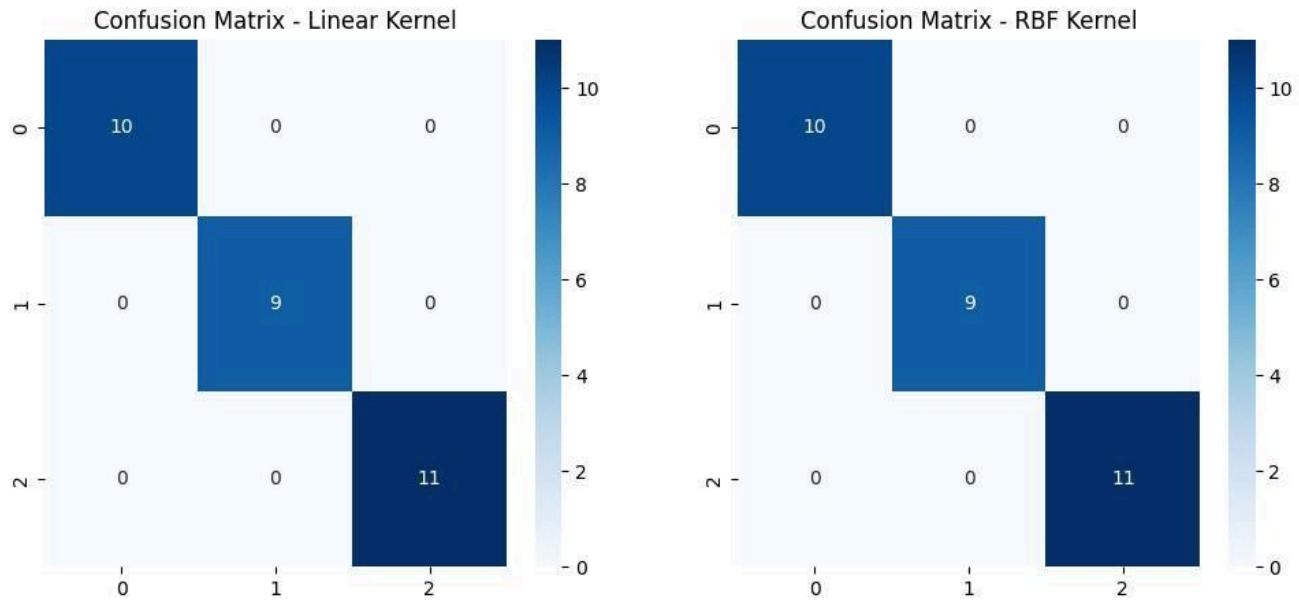
# Linear Kernel Confusion Matrix
sns.heatmap(conf_matrix_linear, annot=True, fmt="d", cmap="Blues",
xticklabels=iris['species'].unique(),
yticklabels=iris['species'].unique(), ax=axes[0])
axes[0].set_title('Confusion Matrix - Linear Kernel')

# RBF Kernel Confusion Matrix
sns.heatmap(conf_matrix_rbf, annot=True, fmt="d", cmap="Blues",
xticklabels=iris['species'].unique(),
yticklabels=iris['species'].unique(), ax=axes[1])
axes[1].set_title('Confusion Matrix - RBF Kernel')

plt.show()

```

Accuracy of SVM with Linear Kernel: 1.00
Accuracy of SVM with RBF Kernel: 1.00



In [21]:

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,
auc
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import
label_binarize

# Load the Letter-recognition dataset
letter_data = pd.read_csv('letter-recognition.csv')

# Prepare the dataset
X_letter = letter_data.drop('letter', axis=1)
y_letter = letter_data['letter']

# Encode the letter labels to numerical values
le = LabelEncoder()
y_letter_encoded = le.fit_transform(y_letter)

# Split the data into training and testing sets (80% training, 20% testing)
X_train_letter, X_test_letter, y_train_letter, y_test_letter =
train_test_split(X_letter, y_letter_encoded, test_size=0.2, random_state=42)

```

```

# Train the SVM model with a linear kernel svm_linear
= SVC(kernel='linear') svm_linear.fit(X_train_letter,
y_train_letter)

# Make predictions
y_pred_letter = svm_linear.predict(X_test_letter)

# Calculate accuracy score accuracy =
accuracy_score(y_test_letter, y_pred_letter)
print(f'Accuracy Score with Linear Kernel: {accuracy:.4f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_letter, y_pred_letter)

# Plot Confusion Matrix (without printing the elements)
plt.figure(figsize=(10, 8)) # Make the plot larger
ConfusionMatrixDisplay(conf_matrix,
display_labels=le.classes_).plot(cmap='Blues') plt.title('Confusion Matrix for
Letter-recognition Dataset') plt.show()

# Binarize the labels for multi-class classification
y_test_letter_binarized = label_binarize(y_test_letter, classes=np.arange(26)) # 26 classes
y_pred_prob_letter = svm_linear.decision_function(X_test_letter)

# Compute ROC curve and AUC score for each class
fpr = {} tpr = {} roc_auc = {}
for i in range(26): # 26 classes
    fpr[i], tpr[i], _ = roc_curve(y_test_letter_binarized[:, i], y_pred_prob_letter[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure(figsize=(10, 8)) for i
in range(26):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

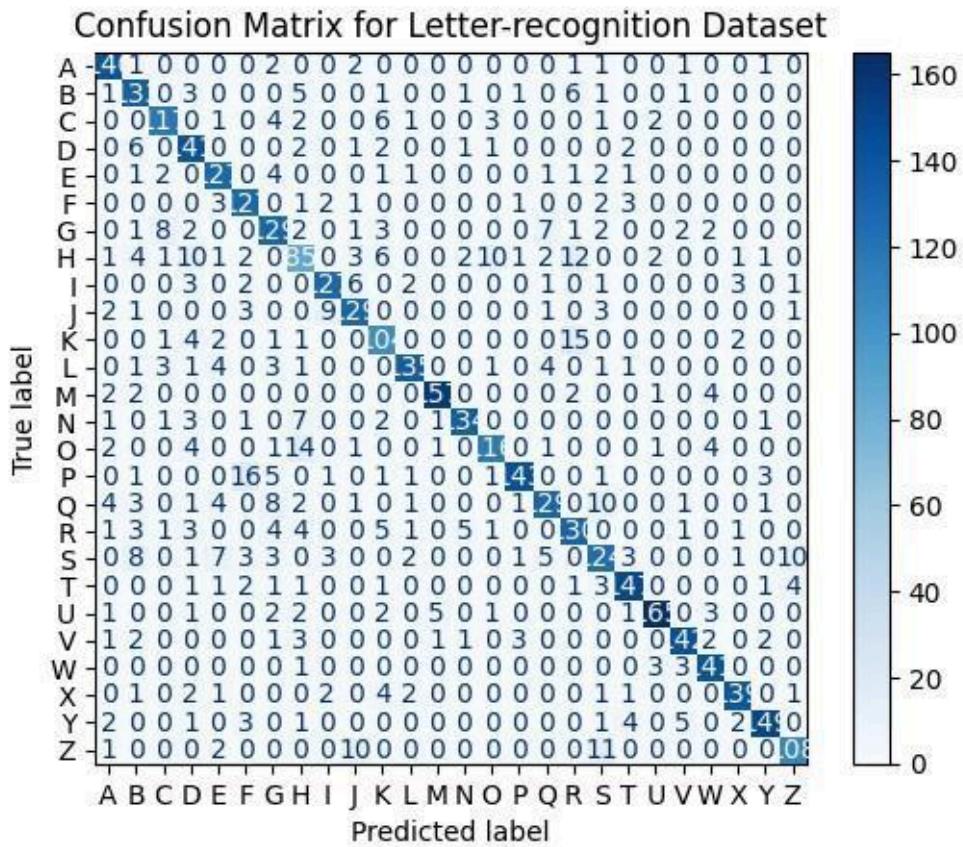
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate') plt.ylabel('True
Positive Rate')
plt.title('Receiver Operating Characteristic for Letter-recognition Dataset (Multi-class)')
plt.legend(loc="lower right") plt.show()

# Display AUC scores for each class for i in
range(26): print(f'AUC score for Class {i}:
{roc_auc[i]:.4f}')

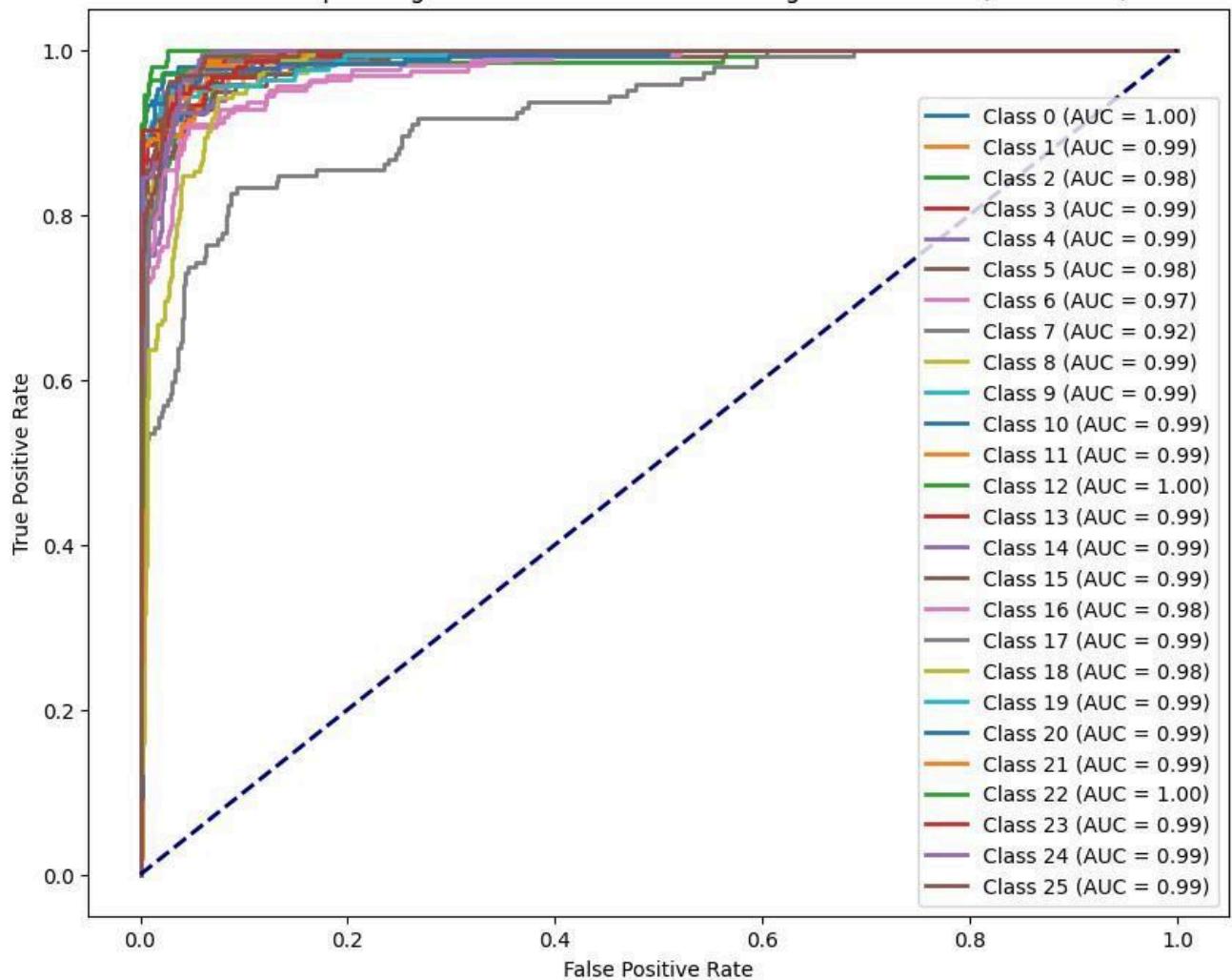
```

Accuracy Score with Linear Kernel: 0.8545

<Figure size 1000x800 with 0 Axes>



Receiver Operating Characteristic for Letter-recognition Dataset (Multi-class)



AUC score for Class 0: 0.9955

AUC score for Class 1: 0.9896

AUC score for Class 2: 0.9825

AUC score for Class 3: 0.9938

AUC score for Class 4: 0.9915

AUC score for Class 5: 0.9825

AUC score for Class 6: 0.9742

AUC score for Class 7: 0.9238

AUC score for Class 8: 0.9888

AUC score for Class 9: 0.9925

AUC score for Class 10: 0.9864

AUC score for Class 11: 0.9908

AUC score for Class 12: 0.9973

AUC score for Class 13: 0.9918

AUC score for Class 14: 0.9855

AUC score for Class 15: 0.9909

```
AUC score for Class 16: 0.9766  
AUC score for Class 17: 0.9882  
AUC score for Class 18: 0.9779  
AUC score for Class 19: 0.9893  
AUC score for Class 20: 0.9906  
AUC score for Class 21: 0.9934  
AUC score for Class 22: 0.9979  
AUC score for Class 23: 0.9931  
AUC score for Class 24: 0.9936  
AUC score for Class 25: 0.9934
```

```
In [13]:
```

```
import pandas as pd
```

```
# Data  
data = {  
    "Height": [44, 52.1, 57.1, 33, 27.8, 27.2, 32, 45.1, 56.7, 56.9, 122.1],  
    "Weight": [126.3, 136.9, 109.2, 148.3, 110.4, 107.8, 128.4, 120.2, 140.2, 139.2, 154.1] }
```

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

```
# Save to CSV  
df.to_csv('height_weight_data.csv', index=False)  
import pandas as pd import matplotlib.pyplot as  
plt
```

```
# Load the data from CSV  
df = pd.read_csv('height_weight_data.csv')
```

```
# Plotting the data  
plt.scatter(df['Height'], df['Weight'], color='blue', label='Data points')  
plt.xlabel('Height') plt.ylabel('Weight') plt.title('Height vs Weight')  
plt.grid(True) plt.show()  
from sklearn.svm import SVR  
from sklearn.model_selection import train_test_split from  
sklearn.metrics import mean_squared_error
```

```
# Define features and target X =  
df[['Height']] # Feature: Height  
y = df['Weight'] # Target: Weight
```

```
# Split data 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)

# Create the model
svr = SVR(kernel='linear')

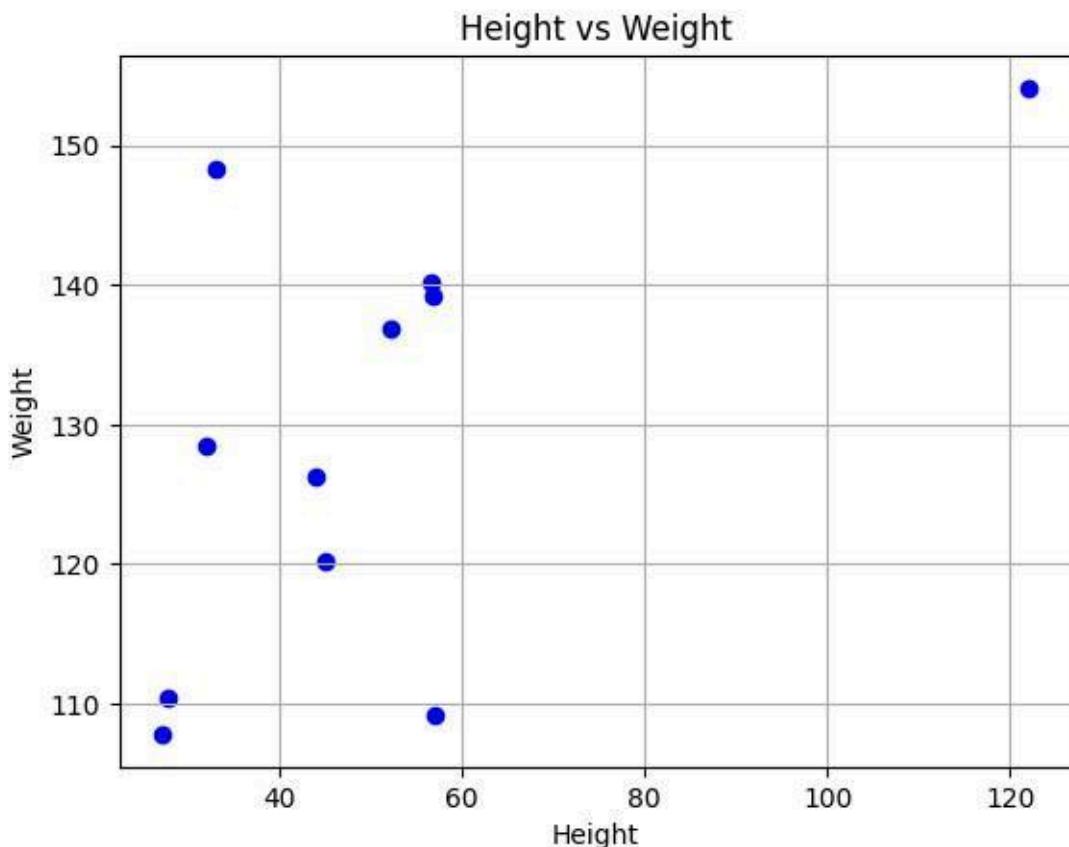
# Fit the model
svr.fit(X_train, y_train)

# Predict on the test set
y_pred = svr.predict(X_test)

# Calculate accuracy using Mean Squared Error (since it's regression) mse
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5 # Root Mean Squared Error

# Print the support vectors and accuracy
print(f"Support Vectors: {svr.support_}")
print(f"Root Mean Squared Error: {rmse}")

```



Support Vectors: [0 1 2 3 4 5 6 7]
Root Mean Squared Error: 11.680171349283054

Program 8

Implement Random forest ensemble method on a given dataset.

Algorithm

Lab 8 - Random Forest	
Decision Tree	Random Forest
→ A decision tree is a tree-like model of decisions along with possible outcomes in a diagram.	a classification algorithm consisting of many decision trees combined to get a more accurate result compared to a single tree.
→ There is always a scope for overfitting due to the presence of various leaf nodes.	Random forest algorithm handles overfitting by using multiple trees.
→ The leafs are not accurate.	This gives accurate & precise results.
→ Decision trees require low computation.	Requires more computation.
→ Easy to visualize.	Complex visualization.
Parameters of random forest	
→ max_features : maximum number of features Random forest is allowed to try in a individual tree. Default is Auto/None, sqrt, 0.2	Random forest
→ n_estimators : number of trees you want to build before taking maximum voting or averaging of predictions.	n_estimators
→ min_samples_leaf :	min_samples_leaf
→ n_jobs : no. of processors allowed to use.	n_jobs
→ random_state :	random_state
→ oob_score : cross validation method	oob_score
Algorithm:	
Step 1:	Select random samples from a given data or training set
Step 2:	The algorithm will construct a decision tree for every training data
Step 3:	Voting will take place by averaging the decision tree

Step 9: Finally, Select the most voted prediction result as the final prediction result.

accuracy with 10 n-estimators = $10 \div 10$

accuracy with 50 " $= 50 \div 50$

accuracy with 100 " $= 100 \div 100$

Code:

In [11]:

```
import pandas as pd from sklearn.model_selection import  
train_test_split from sklearn.ensemble import  
RandomForestClassifier from sklearn.metrics import  
accuracy_score, confusion_matrix from sklearn.preprocessing  
import LabelEncoder  
  
# Load the dataset df =  
pd.read_csv('train.csv')  
  
# Preprocess the dataset  
# Convert categorical columns into numeric  
# For simplicity, we'll drop columns that are not necessary for modeling (e.g., 'Name', 'Ticket',  
'Cabin')  
df = df.drop(columns=['Name', 'Ticket', 'Cabin'])  
  
# Convert categorical features into numeric values (Sex, Embarked)  
label_encoder = LabelEncoder()  
df['Sex'] = label_encoder.fit_transform(df['Sex'])  
df['Embarked'] = df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2}) # 'S', 'C', 'Q' to numeric  
  
# Handle missing values (impute with the median for simplicity)  
df['Age'].fillna(df['Age'].median(), inplace=True)  
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) # Mode is used for Embarked  
  
# Features and target variable  
X = df.drop(columns=['Survived', 'PassengerId'])  
y = df['Survived']  
  
# Split the data into training (80%) and testing (20%) sets  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
# Initialize the Random Forest Classifier  
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)  
  
# Train the model rf_classifier.fit(X_train,  
y_train)  
  
# Predict on the test set  
y_pred = rf_classifier.predict(X_test)
```

```

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred) print(f'Accuracy: {accuracy:.4f}')
```



```

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix: ')
print(conf_matrix)
```

<ipython-input-11-ca7102c51cf1>:21: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```

df['Age'].fillna(df['Age'].median(), inplace=True)
<ipython-input-11-ca7102c51cf1>:22: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) # Mode is used for Embarked
Accuracy: 0.8212 Confusion Matrix:
[[91 14]
 [18 56]] In
[12]:
```

```

# Import necessary libraries from sklearn.datasets
import load_iris from sklearn.model_selection import
train_test_split from sklearn.ensemble import
RandomForestClassifier from sklearn.metrics import
accuracy_score
import matplotlib.pyplot as plt
```

```

# Load the Iris
dataset iris =
load_iris() X =
iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize RandomForestClassifier with default n_estimators=10
clf = RandomForestClassifier(n_estimators=10, random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the performance
score_default = accuracy_score(y_test, y_pred)
print(f"Accuracy with n_estimators=10: {score_default}")

# Fine-tune the model by changing the value of
n_estimators n_estimators_values = [10, 50, 100, 200, 500]
scores = []

for n in n_estimators_values:    clf =
RandomForestClassifier(n_estimators=n, random_state=42)
clf.fit(X_train, y_train)    y_pred = clf.predict(X_test)    score =
accuracy_score(y_test, y_pred)    scores.append(score)
print(f"Accuracy with n_estimators={n}: {score}")

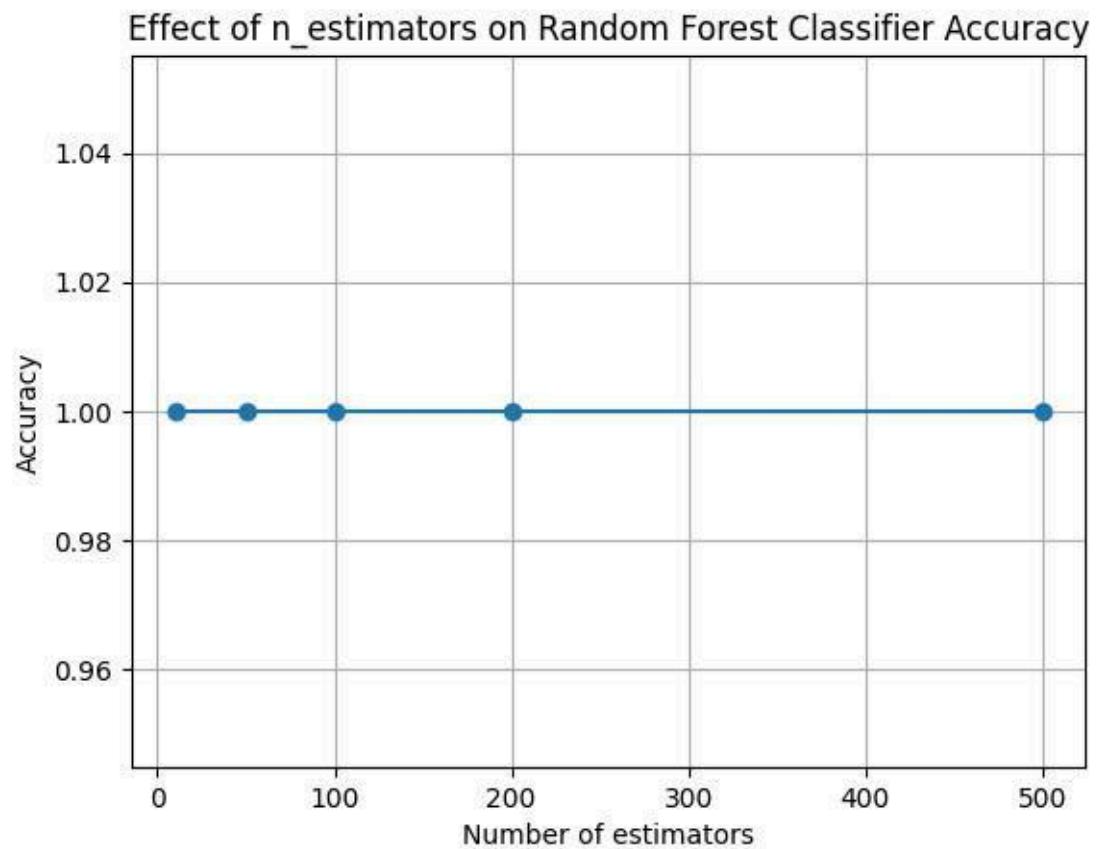
# Plot the results
plt.plot(n_estimators_values, scores, marker='o')
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.title('Effect of n_estimators on Random Forest Classifier Accuracy') plt.grid(True)
plt.show()

```

Accuracy with n_estimators=10: 1.0
 Accuracy with n_estimators=10: 1.0
 Accuracy with n_estimators=50: 1.0
 Accuracy with n_estimators=100: 1.0

Accuracy with n_estimators=200: 1.0

Accuracy with n_estimators=500: 1.0



In []:

Program 9

Implement Boosting ensemble method on a given dataset.

Algorithm

Lab 9 - AdaBoost

(a) Boosting is an ensemble learning technique that combines multiple weak decision trees to create a strong learner. It works by sequentially training models, where each model focuses on mistakes made by previous ones. The models' predictions are weighted based on their performance, and the final prediction is made by combining the weighted predictions of these models.

Q2 Parameters of AdaBoost Classifiers

- * best estimator (object, default = None) - base model to be used for boosting, uses decision trees by default.
- * n_estimators (int) - the number of boosting rounds, it makes learners to train
- * learning_rate (float) - scaling factor for the contribution of each estimator; lower values make models robust.
- * algorithm ('SAMME', 'SAMME-R') default = 'SAMME'
boosting algorithm to use - SAMME-R uses probabilistic approaches
- * random_state (int) - controls randomness of the estimator
- * loss ('linear', 'square', 'exponential') - loss function to be used for boosting

Q3 Algorithm

- (1) initialize: Set the weights of all training samples equally
- (2) for each initialization (n-estimators)
 - Train a base learner on weighted training set
 - Calculate error of the model on training set
 - Compute model's weight based on error
 - Update sample weight - increase weight of misclassified samples

③ combine : The final model is the weighted sum of the base learners , where learners with lower errors have higher weights

code:

```
data = pd.read_csv('income.csv')
```

```
X = data.drop(columns = ['income-level'])
```

```
y = data['income-level']
```

```
X-train, X-test, y-train, y-test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
ada-model = AdaBoostClassifier(n_estimators=50, learning_rate=1, random_state=42)
```

```
ada-model.fit(X-train, y-train)
```

```
y-pred = ada-model.predict(X-test)
```

```
accuracy = accuracy_score(y-test, y-pred)
```

```
conf-matrix = confusion_matrix(y-test, y-pred)
```

```
print("Accuracy: " + str(accuracy * 100))
```

```
print("Confusion Matrix: ")
```

```
print(conf-matrix)
```

O/P

Accuracy = 83.2%

Confusion Matrix

$\begin{bmatrix} 1003 & 411 \end{bmatrix}$

$\begin{bmatrix} 1223 & 1132 \end{bmatrix}$

Code

In [32]:

```
import pandas as pd from sklearn.model_selection import  
train_test_split from sklearn.preprocessing import  
LabelEncoder from sklearn.ensemble import  
AdaBoostClassifier from sklearn.tree import  
DecisionTreeClassifier from sklearn.linear_model import  
LogisticRegression from sklearn.metrics import  
accuracy_score, confusion_matrix from sklearn.ensemble  
import RandomForestClassifier # Load the data (assume  
"income.csv" is already downloaded) df =  
pd.read_csv('income.csv')  
  
# Let's check the columns to see what we're dealing with print(df.head())  
  
# Preprocessing: Encode categorical data if necessary (in this case 'income_level' is categorical)  
encoder = LabelEncoder()  
df['income_level'] = encoder.fit_transform(df['income_level'])  
  
# Splitting the data into features (X) and target (y)  
X = df.drop('income_level', axis=1)  
y = df['income_level']  
  
# Split the data into 80% training and 20% testing  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	income_level
0	39	77516	13	2174	0	40	0
1	50	83311	13	0	0	13	0
2	38	215646	9	0	0	40	0
3	53	234721	7	0	0	40	0
4	28	338409	13	0	0	40	0

In [33]:

```
# Initialize the AdaBoost model with Random Forest as the base classifier
```

```

ada_boost = AdaBoostClassifier(RandomForestClassifier(n_estimators=10, max_depth=5,
random_state=42), n_estimators=50, learning_rate=1.0)

# Train the AdaBoost model
ada_boost.fit(X_train, y_train)

# Predictions
y_pred = ada_boost.predict(X_test)

# Evaluate accuracy and confusion matrix accuracy
accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")

```

Accuracy: 0.8368307912785341 Confusion
Matrix:

```

[[7039  375]
 [1219 1136]]

```

In [34]:

```

from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.ensemble import AdaBoostClassifier from
sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

```

```

# Load the Iris
dataset iris =
load_iris() X =
iris.data y = iris.target

```

```

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```

base_classifier = RandomForestClassifier(n_estimators=10, max_depth=5, random_state=42) #
Initialize AdaBoost with Decision Tree as base classifier
ada_boost = AdaBoostClassifier(base_classifier, n_estimators=50)

```

```

# Train the model ada_boost.fit(X_train,
y_train)

```

```

# Evaluate accuracy on the test set y_pred
= ada_boost.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of AdaBoost on Iris dataset: {accuracy}')

```

Accuracy of AdaBoost on Iris dataset: 1.0 In
[35]:

```

# Varying n_estimators and measuring the accuracy
estimators = [10, 50, 100, 200, 500] for n in estimators:
ada_boost = AdaBoostClassifier(base_classifier, n_estimators=n)
    ada_boost.fit(X_train, y_train)    y_pred =
    ada_boost.predict(X_test)    accuracy =
    accuracy_score(y_test, y_pred)    print(f'Accuracy with
n_estimators={n}: {accuracy}')

```

Accuracy with n_estimators=10: 1.0
Accuracy with n_estimators=50: 1.0
Accuracy with n_estimators=100: 1.0
Accuracy with n_estimators=200: 1.0
Accuracy with n_estimators=500: 1.0 In
[36]:

```

# Varying n_estimators and learning_rate
learning_rates = [0.1, 0.5, 1.0, 1.5]
n_estimators_values = [50, 100, 200]

for lr in learning_rates:
    for n in n_estimators_values:
        ada_boost = AdaBoostClassifier(base_classifier,
n_estimators=n, learning_rate=lr)        ada_boost.fit(X_train, y_train)
        y_pred = ada_boost.predict(X_test)    accuracy =
        accuracy_score(y_test, y_pred)
        print(f'Accuracy with n_estimators={n} and learning_rate={lr}: {accuracy}')

```

Accuracy with n_estimators=50 and learning_rate=0.1: 1.0
Accuracy with n_estimators=100 and learning_rate=0.1: 1.0
Accuracy with n_estimators=200 and learning_rate=0.1: 1.0
Accuracy with n_estimators=50 and learning_rate=0.5: 1.0
Accuracy with n_estimators=100 and learning_rate=0.5: 1.0
Accuracy with n_estimators=200 and learning_rate=0.5: 1.0
Accuracy with n_estimators=50 and learning_rate=1.0: 1.0

Accuracy with n_estimators=100 and learning_rate=1.0: 1.0

Accuracy with n_estimators=200 and learning_rate=1.0: 1.0

Accuracy with n_estimators=50 and learning_rate=1.5: 1.0

Accuracy with n_estimators=100 and learning_rate=1.5: 1.0

Accuracy with n_estimators=200 and learning_rate=1.5: 1.0

In [37]:

```
from sklearn.linear_model import LogisticRegression
```

Using Logistic Regression as base estimator in AdaBoost

```
ada_boost_lr = AdaBoostClassifier(LogisticRegression(max_iter=1000), n_estimators=50)
ada_boost_lr.fit(X_train, y_train) y_pred_lr = ada_boost_lr.predict(X_test) accuracy_lr =
accuracy_score(y_test, y_pred_lr)
print(f'Accuracy with Logistic Regression base estimator: {accuracy_lr}')
```

Accuracy with Logistic Regression base estimator: 0.9333333333333333

In [38]:

Using Decision Tree as base estimator in AdaBoost

```
ada_boost_tree = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50)
ada_boost_tree.fit(X_train, y_train) y_pred_tree = ada_boost_tree.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print(f'Accuracy with Decision Tree base estimator: {accuracy_tree}')
```

Accuracy with Decision Tree base estimator: 0.9333333333333333

In []:

Program 10

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Algorithm

LAB - 10 K-means

Q Algorithm for K-means

- (i) Select the number K to decide the number of clusters
- (ii) Select random K points or centroids
- (iii) Assign each data point to their closest centroid, which will form the predefined K clusters.
- (iv) Calculate the variance & place a new centroid of each cluster
- (v) Repeat the above step, which means reassigning each datapoint to the new closest centroid of each cluster
- (vi) If any reassignment occurs, then go to step 4 - Finish

Q K determines the number of clusters

Elbow method

Silhouette score

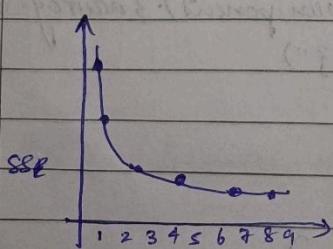
cross validation

gap statistic

Q SSE is defined as the sum of squared Euclidean distances of each point to its closest centroid

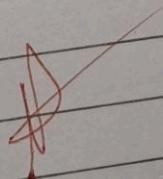
$$SSE_j = \sum_{i=0}^m \text{dist}(x_i - c_j)^2$$

$$SSE = SSE_1 + SSE_2 + \dots + SSE_K$$



Q Elbow method is a technique used to determine the optimal number of clusters (k) in K-means clustering. It involves plotting the within cluster sum of squares (WCSS) against the number of clusters & identifying the "elbow" point, where the decrease in WCSS starts to level off. The elbow point suggests the optimal number of clusters.

- n_clusters : int, default = 8 : number of clusters to form as well as the number of centroids to generate
- init : {‘k-means+’, ‘random’} : chooses a method for initialization
- n_init : number of times the K-means algorithm is run with different centroid seeds
- max_iter : max number of iterations of k-mean algorithm for a single run
- tol : float : Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence
- verbose : verbosity mode
- random_state
- copy_X : bool
- algorithm : {“lloyd”, “elkan”}



code:

`iris = load_iris()``iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)``scaler = StandardScaler()``iris_scaled = scaler.fit_transform(iris_df)``sse = []``k_rng = range(1, 11)``for k in k_rng:``km = KMeans(n_clusters=k, random_state=42)``km.fit(iris_scaled)``sse.append(km.inertia_)``plt.figure(figsize=(8, 5))``plt.plot(k_rng, sse, marker='o')``plt.title("Elbow method: SSE vs K")``plt.xlabel("K")``plt.ylabel("SSE")``plt.grid(True)``plt.show()`

Code

Open In Colab

In [13]:

```
import pandas as pd import numpy as np from
sklearn.cluster import KMeans from
sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from
sklearn.metrics import accuracy_score import
matplotlib.pyplot as plt
import random

# Step 1: Create synthetic income dataset np.random.seed(42)
names = [f'Person_{i}' for i in range(1, 51)] ages =
np.random.randint(20, 60, size=50) incomes =
np.random.randint(20000, 120000, size=50)

income_df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Income': incomes
})

# Save to CSV (optional)
income_df.to_csv("income.csv", index=False)

# Step 2: Load and preprocess
df = pd.read_csv("income.csv")

# Drop name (non-numeric)
df_numeric = df.drop('Name', axis=1)

# Step 3: Scaling scaler
= StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Step 4: Train-test split
X_train, X_test = train_test_split(df_scaled, test_size=0.2, random_state=42)

# Step 5: SSE vs number of clusters (Elbow
method) sse = [] k_range = range(1, 11) for k in
```

```

k_range:   km = KMeans(n_clusters=k,
random_state=42)
    km.fit(X_train)
    sse.append(km.inertia_)

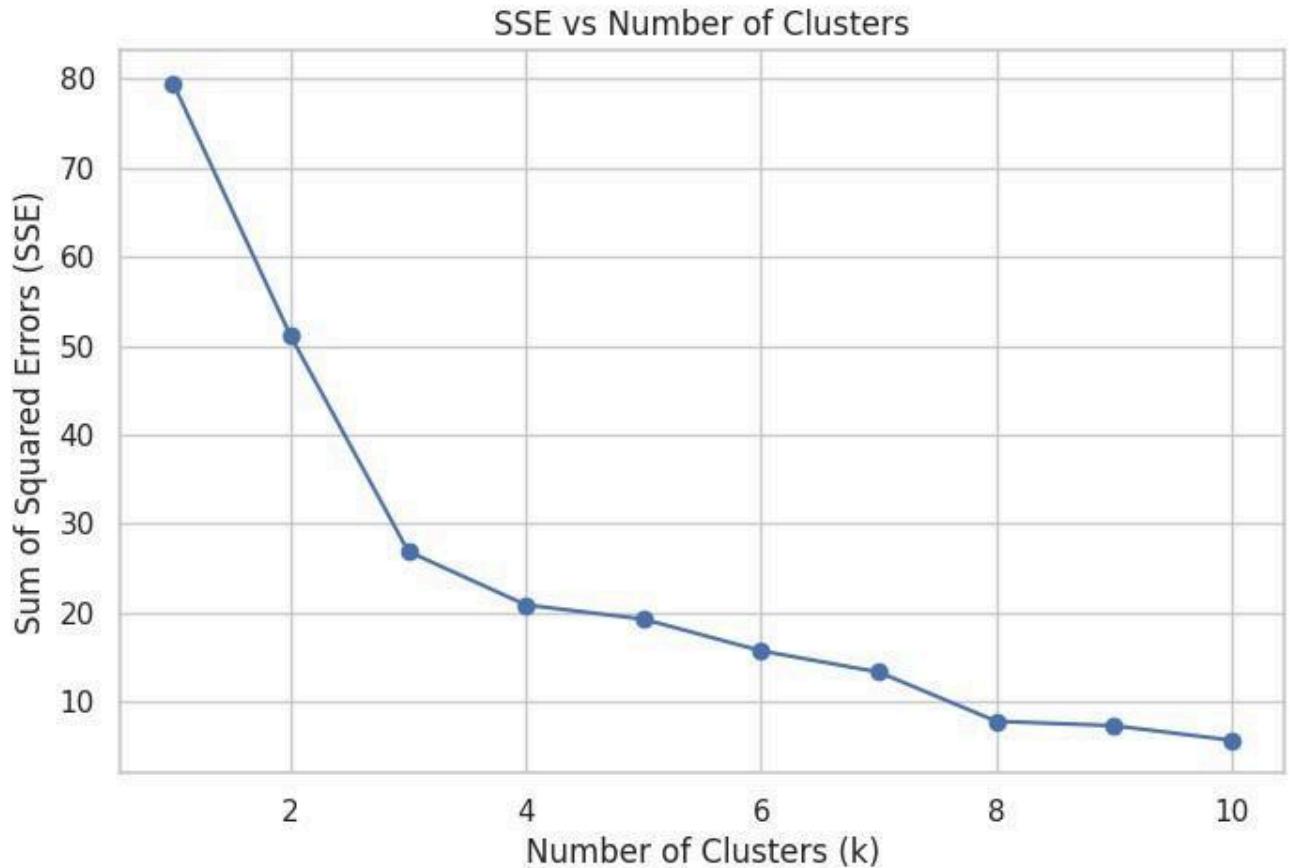
# Step 6: Plot SSE plt.figure(figsize=(8,
5)) plt.plot(k_range, sse, marker='o')
plt.title('SSE vs Number of Clusters')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Errors
(SSE)') plt.grid(True)
plt.show()

# Step 7: Fit KMeans with optimal k (e.g., 3) optimal_k
= 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42) kmeans.fit(X_train)

# Step 8: Predict clusters on test set
y_pred = kmeans.predict(X_test)

# Step 9: "Accuracy" – not meaningful in unsupervised, but just a check
print("Cluster labels for test data:", y_pred)

```



Cluster labels for test data: [2 1 1 1 2 0 2 2 1 0] In
[14]:

```

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

# Step 1: Load iris dataset and use only petal length & width
iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df = iris_df[['petal length (cm)', 'petal width (cm)']] # drop other features

# Step 2: Scaling scaler
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_df)

# Step 3: Elbow method to find optimal k

```

```

sse = [] k_range = range(1, 11) for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(iris_scaled)
    sse.append(km.inertia_)

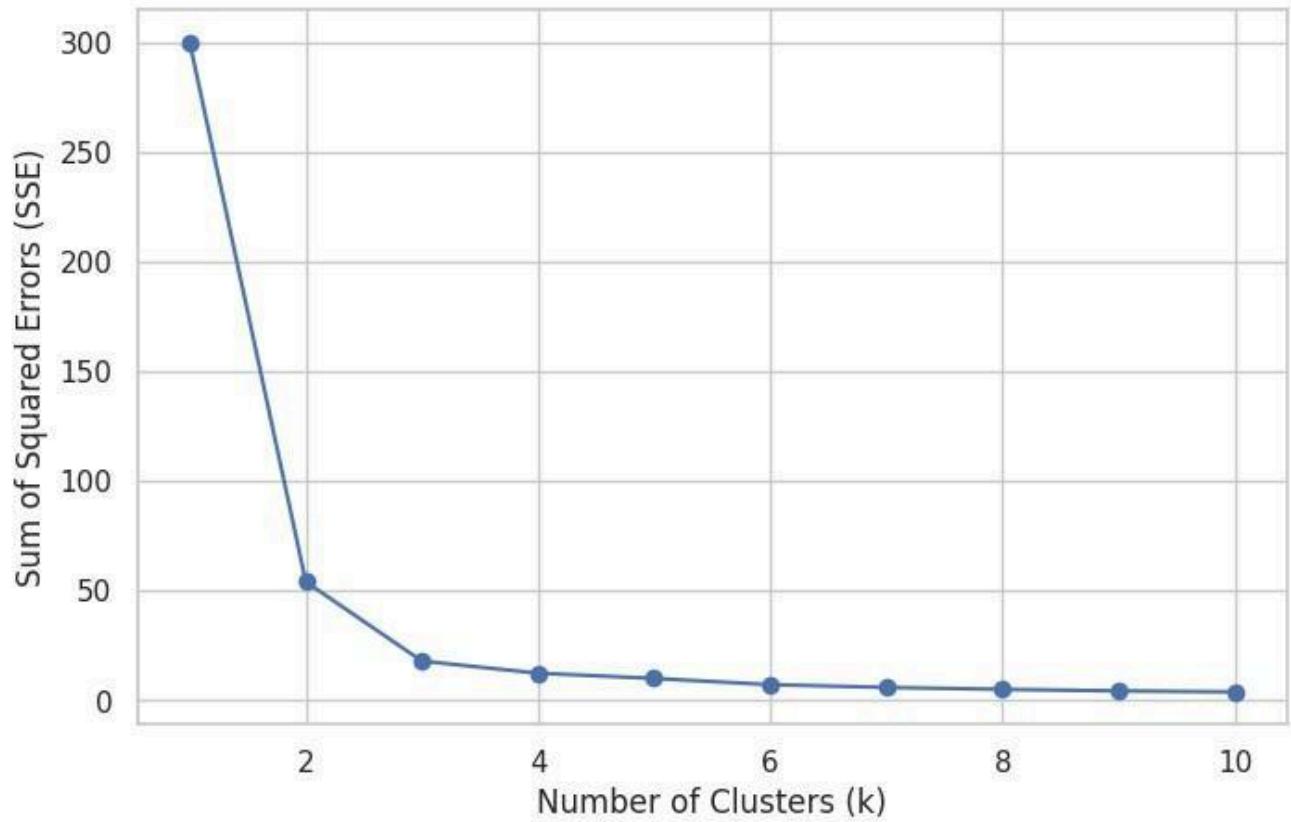
# Step 4: Plot SSE vs k
plt.figure(figsize=(8, 5))
plt.plot(k_range, sse, marker='o')
plt.title("Elbow Method: SSE vs Number of Clusters")
plt.xlabel("Number of Clusters (k)") plt.ylabel("Sum
of Squared Errors (SSE)")
plt.grid(True) plt.show()

# Optional: Fit with optimal k = 3 and visualize clusters
optimal_k
= 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
labels = kmeans.fit_predict(iris_scaled)

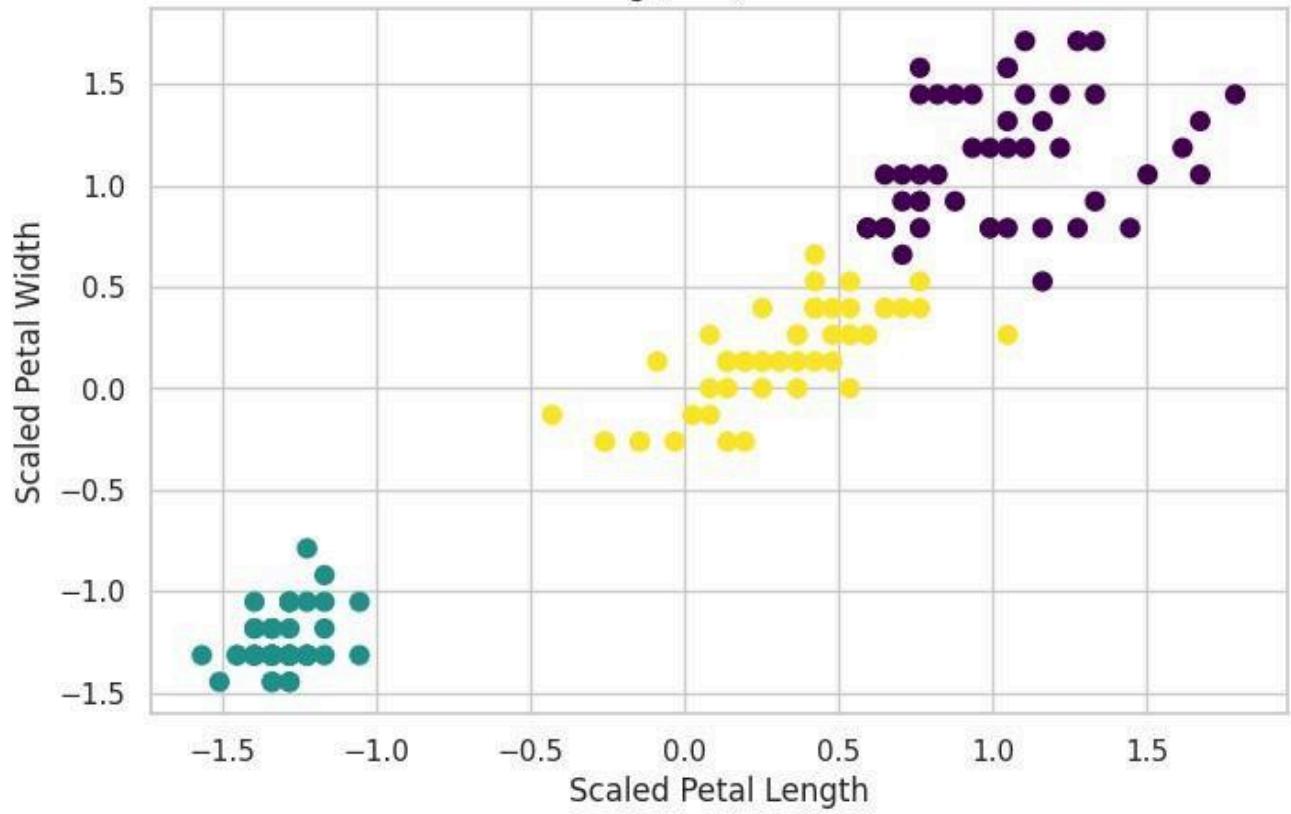
plt.figure(figsize=(8, 5))
plt.scatter(iris_scaled[:, 0], iris_scaled[:, 1], c=labels, cmap='viridis', s=50)
plt.title("KMeans Clustering (k=3) on Iris Petal Features")
plt.xlabel("Scaled Petal Length") plt.ylabel("Scaled Petal
Width") plt.grid(True) plt.show()

```

Elbow Method: SSE vs Number of Clusters



KMeans Clustering (k=3) on Iris Petal Features



In []:

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Algorithm

LAB-11				
PCA	1. calculate the mean	2. calculation of covariance matrix	3. eigen value of the covariance matrix	4. computation of the eigen vectors - Unit eigenvectors
	5. computation of first principal components	6. geometrical meaning of first principal components		
				reduce dimension 2 to 1 using PCA
<u>Feature</u>	Example 1	Example 2	Example 3	Example 4
x_1	4	8	13	7
x_2	11	9	5	14
Step 1:	calculate mean			
	$\bar{x}_1 = 8$			
	$\bar{x}_2 = 8.5$			
Step 2:	calculation of covariance matrix			
	$\text{cov}(x_1, x_1) = \frac{1}{N-1} \sum_{k=1}^N (x_{1k} - \bar{x}_1)^2$			
	$= \frac{1}{3} ((4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2)$			
	$= 14$			
	$\text{cov}(x_1, x_2) = \frac{1}{N-1} \sum_{k=1}^N (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)$			
	$= \frac{1}{3} ((4-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5))$			
	$= -11$			
	$\text{cov}(x_2, x_1) = \text{cov}(x_1, x_2)$			
	$= -11$			
	$\text{cov}(x_2, x_2) = \frac{1}{N-1} \sum_{k=1}^N (x_{2k} - \bar{x}_2)^2$			
	$= \frac{1}{3} ((11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2)$			
	<u>$= 23$</u>			

$$S = \begin{bmatrix} \text{cov}(x_1, x_1) & \text{cov}(x_1, x_2) \\ \text{cov}(x_2, x_1) & \text{cov}(x_2, x_2) \end{bmatrix}$$

Step 3: Eigen values of the covariance matrix

The characteristic equation of the covariance matrix is

$$D^2 \det(S - \lambda I)$$

$$= \begin{vmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{vmatrix}$$

$$= (14-\lambda)(23-\lambda) - (-11) * (-11)$$

$$= \lambda^2 - 37\lambda + 201$$

Solving characteristic equation we get

$$\lambda = \frac{1}{2} (37 \pm \sqrt{565})$$

$$= 30, 3849, 66151$$

$$= \lambda_1, \lambda_2 (\text{say})$$

Step 4: Computation of the eigen vector

largest eigen vector values = λ_1

$$\lambda = \lambda_1$$

$$V^e = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_1 I) X$$

$$= \begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$= \begin{bmatrix} (14-\lambda_1)u_1 - 11u_2 \\ -11u_1 + (23-\lambda_1)u_2 \end{bmatrix}$$

$$= (14-\lambda_1)u_1 - 11u_2 = 0$$

$$-11u_1 + (23-\lambda_1)u_2 = 0$$

$$\frac{u_2}{11} = \frac{u_2}{14 - d_1} = t$$

$$u_1 = 11t, \quad u_2 = (14 - d_1)t$$

where t is any real number

Taking $t = 1$, we get an eigenvector corresponding to d_1 as

$$v_1 = \begin{bmatrix} 11 \\ 14 - d_1 \end{bmatrix}$$

$$\begin{aligned} \|v_1\| &= \sqrt{11^2 + (14 - d_1)^2} \\ &= \sqrt{11^2 + (14 - 30.3849)^2} \\ &= 19.7348 \end{aligned}$$

Therefore, a unit eigenvector corresponding to d_1 is

$$e_1 = \begin{bmatrix} 11/\|v_1\| \\ (14 - d_1)/\|v_1\| \end{bmatrix}$$

$$= \begin{bmatrix} 11/19.7348 \\ (14 - 30.3849)/19.7348 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5534 \\ -0.8303 \end{bmatrix}$$

eigen vector e_2 corresponding to eigen value $d = d_2$

$$e_2 = \begin{bmatrix} 0.8303 \\ 0.5534 \end{bmatrix}$$

Step 5: Computation of 1st principal component

$$\begin{bmatrix} X_{1k} \\ X_{2k} \end{bmatrix}$$

Code

In [12]:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load dataset
digits = load_digits()
X, y = digits.data, digits.target

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Logistic Regression without PCA
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
preds = lr.predict(X_test)
print(f'Accuracy without PCA: {accuracy_score(y_test, preds):.4f}')

# Apply PCA with 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Split PCA-reduced data
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Logistic Regression with PCA-reduced data
lr_pca = LogisticRegression(max_iter=1000)
lr_pca.fit(X_train_pca, y_train_pca)
preds_pca = lr_pca.predict(X_test_pca)
print(f'Accuracy with PCA (2 components): {accuracy_score(y_test_pca, preds_pca):.4f}') Accuracy without PCA: 0.9722
```

Accuracy with PCA (2 components): 0.5389

In [13]:

```
import pandas as pd import numpy as np from
sklearn.preprocessing import StandardScaler from
sklearn.model_selection import train_test_split from
sklearn.ensemble import RandomForestClassifier from
sklearn.linear_model import LogisticRegression from
sklearn.svm import SVC from sklearn.metrics import
accuracy_score from sklearn.decomposition import
PCA from scipy.stats import zscore

# Load dataset
heart_df = pd.read_csv("heart.csv")

# Print column names to verify
print("Column names in dataset:", heart_df.columns.tolist())

# Optional: rename columns if necessary (based on your dataset) #
Example only – skip this if your columns are already clean
heart_df.columns = [col.strip().lower() for col in heart_df.columns]

# Reprint cleaned column names
print("Cleaned column
names:", heart_df.columns.tolist())

# Remove outliers using Z-score
z_scores = np.abs(zscore(heart_df.select_dtypes(include=[np.number])))
heart_df = heart_df[(z_scores < 3).all(axis=1)]

# Determine categorical columns that need encoding
categorical_cols = heart_df.select_dtypes(include=['object', 'category']).columns.tolist()

# Encode categorical columns using one-hot encoding if
categorical_cols:
    heart_df = pd.get_dummies(heart_df, columns=categorical_cols, drop_first=True)

# Split features and target
target_col = 'target' if 'target' in heart_df.columns else heart_df.columns[-1]
X = heart_df.drop(target_col, axis=1)
y = heart_df[target_col]

# Feature scaling
scaler =
StandardScaler()
X_scaled =
scaler.fit_transform(X)
```

```

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Models models
= {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier()
}

print("\nModel Accuracy WITHOUT
PCA:") for name, model in models.items():
model.fit(X_train, y_train)  preds =
model.predict(X_test)
print(f'{name}: {accuracy_score(y_test, preds):.4f}')

# Apply PCA pca =
PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

print("\nModel Accuracy WITH PCA:") for
name, model in models.items():
model.fit(X_train_pca, y_train_pca)
preds = model.predict(X_test_pca)
print(f'{name}: {accuracy_score(y_test_pca, preds):.4f}')

```

Column names in dataset: ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope', 'HeartDisease']
Cleaned column names: ['age', 'sex', 'chestpaintype', 'restingbp', 'cholesterol', 'fastingbs', 'restingecg',
'maxhr', 'exerciseangina', 'oldpeak', 'st_slope', 'heartdisease']

Model Accuracy WITHOUT PCA:

Logistic Regression: 0.9444

SVM: 0.9333

Random Forest: 0.9222

Model Accuracy WITH PCA:

Logistic Regression: 0.8611

SVM: 0.8778 Random

Forest: 0.8833

In []: