

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

ADITHYA RAVIKEERTHI(1BM22CS020)

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 **Adithya Ravikeerthi(1BM22CS038)**, 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.

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	21-2-2025	Write a python program to import and export data using Pandas library functions	2-7
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	8-11
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	12-16
4	17-3-2025	Build Logistic Regression Model for a given dataset	17-21
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	22-24
6	7-4-2025	Build KNN Classification model for a given dataset.	25-28
7	21-4-2025	Build Support vector machine model for a given dataset	29-31
8	5-5-2025	Implement Random forest ensemble method on a given dataset.	32-33
9	5-5-2025	Implement Boosting ensemble method on a given dataset.	34-35
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	36-38
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	39-42

Github Link:

https://github.com/adithyaRk020/ML_LA

B

Program 1

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

Screenshot:

LAB-1	7/3/25	Page
Housing.csv		
1) Load the data frame		
import pandas as pd		
df = pd.read_csv('housing.csv')		
2) Display info about all columns		
print(df.describe())		
#	column	Non-null
0	longitude	20640
1	latitude	20640
2	median_age	20640
3	total_rooms	20640
4	bedrooms	20640
5	pop	20640
6	households	20640
7	med_inc	20640
8	med_housval	20640
9	ocean_proximity	20640
3) Display statistical info of all numerical column		
print(df.describe())		
min	long lat median_age, total_rooms bed	
	-124 39.5 1.0 1.0 2.0	
pop"	households med_inc . med_housval	
	3 1 0.499	

<p>4) Display the count of unique labels</p> <pre>print('off ocean proximity'.value_counts)</pre> <p>output</p> <table border="1"> <thead> <tr> <th>Ocean proximity</th> <th>Count</th> </tr> </thead> <tbody> <tr><td>in ocean</td><td>9136</td></tr> <tr><td>Inland</td><td>6551</td></tr> <tr><td>near ocean</td><td>2658</td></tr> <tr><td>near bay</td><td>2290</td></tr> <tr><td>off island</td><td>5</td></tr> </tbody> </table> <p>5) Display the attributes that have missing values greater than 5%</p> <pre>missing_val = df.isnull().sum() missing_col = missing_val[missing_val > 0] print(missing_col)</pre> <p>output</p> <table border="1"> <thead> <tr> <th>Total bedrooms</th> <th>207</th> </tr> </thead> <tbody> <tr><td>total bedrooms</td><td>207</td></tr> </tbody> </table>	Ocean proximity	Count	in ocean	9136	Inland	6551	near ocean	2658	near bay	2290	off island	5	Total bedrooms	207	total bedrooms	207	<p>Date _____ Page _____</p> <p>To Do</p> <ol style="list-style-type: none"> 1) Which columns in the dataset had missing vals, how did you handle them? 2) ID, Gender, Age, cr, TG ... for numerical columns, were filled nan's with mean of respective columns 3) for categorical columns, missing values are filled with mode of respective columns 4) Which categorical column did you identify in dataset? → Cat column can be identified using select_dtypes and were encoded using LabelEncoder 5) What is the diff b/w min-max scaling & standardization? when to use them? min-max scaling: tech rescale the feature values to fixed range standardization: transforms the data to have a mean of 0 & std of 1.
Ocean proximity	Count																
in ocean	9136																
Inland	6551																
near ocean	2658																
near bay	2290																
off island	5																
Total bedrooms	207																
total bedrooms	207																

Code:

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
print("Sample data:")
print(df.head())
```

```

file_path = 'data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
file_path = 'mobiles-dataset-2025.csv'
df = pd.read_csv(file_path, encoding='latin-1') # or 'cp1252' or other suitable
encoding
print("Sample data:")
print(df.head())
import pandas as pd

data = {
'USN': ['IS001','IS002','IS003','IS004','IS005'],
'Name': ['Alice', 'Bob', 'Charlie', 'David','Eve'],
'Marks': [25, 30, 35, 40,45]
}

df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
from sklearn.datasets import load_diabetes
iris = load_diabetes()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

print("Sample data:")
print(df.head())
file_path = 'sample_sales_data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")

df = pd.read_csv("/content/dataset-of-diabetes .csv",encoding='latin-1')
print("Sample data:")
print(df.head())
print("\n")

df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(df.head())

df.to_csv('output.csv',index=False)
print("Data saved to output.csv")
sales_df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(sales_df.head())
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)
sales_by_region.to_csv('sales_by_region.csv')
best_selling_products.to_csv('best_selling_products.csv')
print("Data saved to sales_by_region.csv and best_selling_products.csv")

import yfinance as yf
import matplotlib.pyplot as plt
tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01", end="2023-10-01",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
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()
reliance_data.to_csv('reliance_stock_data.csv')

tickers = ["HDFCBANK.NS", "ICICI.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['HDFCBANK.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="HDFCIndustries - Daily Returns",

```

```

color='red') plt.tight_layout()
plt.show()
reliance_data.to_csv('hdfc_stock_data.csv')
print("\nhdfc stock data saved to 'hdfc_stock_data.csv'.")

tickers = ["HDFCBANK.NS", "ICICIBANK.NS",
"KOTAKBANK.NS"] data = yf.download(tickers,
start="2024-01-01", end="2024-12-30",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="ICICI Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="ICICI Industries - Daily Returns",
color='BLACK') plt.tight_layout()

```

```

plt.show()
reliance_data.to_csv('icici_stock_data.csv')
print("\nicici stock data saved to 'icici_stock_data.csv'.")

```

```

tickers = ["HDFCBANK.NS", "ICICI.NS",
"KOTAKBANK.NS"] data = yf.download(tickers,
start="2024-01-01", end="2024-12-30",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['KOTAKBANK.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] =
reliance_data['Close'].pct_change() print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)

```

```
reliance_data['Close'].plot(title="KOTAK Industries -  
Closing Price") plt.subplot(2, 1, 2)  
reliance_data['Daily Return'].plot(title="kotak Industries - Daily Returns",  
color='red') plt.tight_layout()  
plt.show()  
reliance_data.to_csv('kotak_stock_data.csv')  
print("\nkotak stock data saved to 'kotak_stock_data.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:

4/3/25
Lab 2

Date _____
Page _____

Diff ways of importing dataset

1) Initializing values directly into df

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Eve'],
    'Age': [25, 26, 30],
    'City': ['New York', 'Delhi', 'Punjab']
}
df = pd.DataFrame(data)
print(df)
```

→ Name Age City
Alice 25 New York
Bob 26 Delhi
Eve 30 Punjab

2) Importing datasets from sklearn datasets

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

Date	Page
1/1	1/1
TO DO	
Consider	
HDFC bank, ICICI, Kotak bank and tickers = ('HDFC.NS', 'ICICI.NS', 'Kotak.NS') start date = 2024-01-01 and end date = 2024-12-31	
plot the closing price & daily returns for all the three banks	
Import yfinance as yf Import pandas as pd Import matplotlib as plt	
tickers = ['HDFC.NS', 'ICICI.NS', 'Kotak.NS'] data = yf.download(tickers, start = '2024-01-01', end = '2024-12-31', group_by = 'ticker') print(df.head())	
→ Ticker rel. HDFC.NS price open close high low Date 2024-10-3 1088 1100 1075	

Code:

```
from google.colab import files
diabetes=files.upload()
```

```
from google.colab import files
adult_income=files.upload()
```

```
df1=pd.read_csv("Dataset of Diabetes .csv")
df1.head()
```

```
df2=pd.read_csv("adult.csv")
df2.head()
```

```
df1.info()
df2.info()
df1.describe()
```

```

df2.describe()

missing_values1 = df1.isnull().sum()
print(missing_values1)
missing_values2 = df2.isnull().sum()
print(missing_values2)

df1['Gender'] = df1['Gender'].replace('f', 'F')
ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]])
df1["Gender_Encoded"] =
ordinal_encoder.fit_transform(df1[["Gender"]]) onehot_encoder =
OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df1[["CLASS"]]) encoded_array
= encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["CLASS"])) df_encoded = pd.concat([df1,
encoded_df], axis=1)
df1 = pd.concat([df1, encoded_df], axis=1)
df1.drop("CLASS", axis=1, inplace=True)
df1.drop("Gender", axis=1, inplace=True)
print(df2.head())
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
df_copy2 = df2
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
df_copy2["Gender_Encoded"] =
ordinal_encoder.fit_transform(df_copy2[["gender"]])
print(df_copy2[["gender", "Gender_Encoded"]])

onehot_encoder = OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df2[["occupation", "workclass", "education",
", "marital status", "relationship", "race", "n ative-country", "income"]])
encoded_array = encoded_data.toarray()
encoded_df =
pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["occupation", "workclass", "education",
", "marital status", "relatio nship", "race", "native-country", "income"]))
df_encoded = pd.concat([df_copy2, encoded_df], axis=1)

df_encoded.drop("gender", axis=1, inplace=True)
df_encoded.drop("occupation", axis=1, inplace=True)
df_encoded.drop("workclass", axis=1, inplace=True)
df_encoded.drop("education", axis=1, inplace=True)
df_encoded.drop("marital-status", axis=1, inplace=True)
df_encoded.drop("relationship", axis=1, inplace=True)
df_encoded.drop("race", axis=1, inplace=True)
df_encoded.drop("native-country", axis=1, nplace=True)
df_encoded.drop("income", axis=1, inplace=True)
print(df_encoded. head())

normalizer = MinMaxScaler()

```

```
df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]] =  
normalizer.fit_transform(df_encoded[["fnlwgt","educational-num","capital-gain","capital-  
loss","hours-per week"]]  
])  
df_encoded.head()  
normalizer = MinMaxScaler()  
df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,  
"Chol","TG","HDL","LDL","VLDL","BMI"]] =  
normalizer.fit_transform(df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,  
"Chol","TG","HDL","LDL","VLDL","BMI"]])  
df1.head()
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

Lab-3				
Instance	a ₁	a ₂	a ₃	Classification
1	T	Hot	High	No
2	T	Hot	H	No
3	F	Hot	H	Yes
4	F	Cool	Normal	Yes
5	F	Cool	Normal	Yes
6	T	Cool	H	No
7	T	Hot	H	No
8	T	Hot	N	Yes
9	F	Cool	N	Yes
10	F	Cool	H	Yes

i) Using TD-3 algo, build a decision tree.

ii) Determine the predicted class (pass/fail) for this student on threshold of 0.5.

$p(x) = 0.6457 \quad \text{if } p(x) \geq 0.5$
 $p(x) < 0.5 \quad y = \begin{cases} 1 & \text{if } p(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

$y = 1$ (pass)

iii) Multiclass Logistic regression (using softmax)

$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$

Suppose $z = [2; 1; 0]$

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

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

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

Q) $x = [6.1, 3.5, 1.4, 0.9]$
 $w_0 = [0.5, -0.2, 0.1, 0.3] \quad b_0 = -1.0$
 $w_1 = [-0.3, 0.4, -0.5, 0.2] \quad b_1 = 0.5$
 $w_2 = [0.2, -0.1, 0.8, -0.4] \quad b_2 = 0.0$

$\text{Sof} = Z_0 = w_0^T x + b_0 \quad \therefore \text{fatora}$ $Z_1 = w_1^T x + b_1 \quad \therefore \text{Verificador}$ $Z_2 = w_2^T x + b_2 \quad \therefore \text{Using income}$ $P(y=0/x) = \frac{e^{Z_0}}{e^{Z_0} + e^{Z_1} + e^{Z_2}} = 0.3671$ $P(y=1/x) = \frac{e^{Z_1}}{e^{Z_0} + e^{Z_1} + e^{Z_2}} = 0.0971$ $P(y=2/x) = \frac{e^{Z_2}}{e^{Z_0} + e^{Z_1} + e^{Z_2}} = 0.5368$ Predicted class: $\therefore 0.5368 \Rightarrow \text{Virginia}$	$\text{Gain}(S, a_1) = S - \sum_{v \in T, F} \frac{ v }{S} \text{Entropy}(S_v)$ $= 0.9709 - (0.9192) \left(\frac{5}{10} \right) - 0$ $= 0.60994$ Attribute a_2: $S = 0.9709$ $S_T = [2+, 3+/-] = 0.9709$ $S_F = [4+, 1-] = 0.72192$ $\text{Gain}(S, a_2) = 0.9709 - \frac{5}{10} (0.9709) - \frac{5}{10} (0.72192)$ $= 0.12449$ Variable a_3: $p = 0.9709$ $S_T = [2+, 4-] = 0.8182$ $S_F = [4+, 0-] = 0$ $\text{Gain}(S, a_3) = 0.141998$ Max gain $\Rightarrow a_3$

Code:

```

from google.colab import files
per_capita_income=files.upload()

from google.colab import files
salary=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder,
OneHotEncoder from sklearn.preprocessing import StandardScaler, MinMaxScaler from scipy import stats
from sklearn import linear_model

```

```

df1=pd.read_csv("canada_per_capita_inc
ome.csv") df1.head()

df2=pd.read_csv("salary.csv")
df2.head()
df2.YearsExperience.median()
df2.YearsExperience =
df2.YearsExperience.fillna(df2.YearsExperience.median()) df2

plt.xlabel("year")
plt.ylabel("per capita income (US$)")
plt.scatter(df1.year, df1['per capita income (US$)'])

plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.scatter(df2.YearsExperience, df2.Salary)

reg1 = linear_model.LinearRegression()
reg1.intercept_
reg1.predict([[2020]])

reg2 = linear_model.LinearRegression()
reg2.fit(df2.drop('Salary', axis='columns'),
df2['Salary']) reg2.coef_
reg2.intercept_
reg2.predict([[12]])

from google.colab import files
hirings=files.upload()

from google.colab import files
companies=files.upload()

df3=pd.read_csv("hirings.csv")
df3.head()

df4=pd.read_csv("1000_Companies.csv")
df4.head()

df3.isnull().sum()
df4.isnull().sum()

df3_copy = df3.copy()
experience_mapping = {'two': 2, 'three': 3, 'five': 5, 'seven': 7, 'ten': 10,
'elevens': 11} df3_copy['experience'] =
df3_copy['experience'].map(experience_mapping)
median_experience = df3_copy['experience'].median()
df3_copy['experience'] = df3_copy['experience'].fillna(median_experience)
df3_copy
df3_copy['test_score(out of 10)'] = df3_copy['test_score(out of
10)'].fillna(df3_copy['test_score(out of 10)'].mean())
reg3 = linear_model.LinearRegression()
reg3.fit(df3_copy.drop('salary($)', axis='columns'),

```

```
df3_copy['salary($)']) reg3.coef_
reg3.intercept_
reg3.predict([[2,9,6]])
reg3.predict([[12,10,10]])
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
state_encoded = ohe.fit_transform(df4[['State']])
state_encoded_df = pd.DataFrame(state_encoded, columns=ohe.get_feature_names_out(['State']))

df4 = pd.concat([df4, state_encoded_df], axis=1).drop(columns=['State'])
print(df4)
reg4 = linear_model.LinearRegression()
reg4.fit(df4.drop('Profit',axis='columns'),df4.Profit)
print(reg4.coef_)
print(reg4.intercept_)
reg4.predict([[91694.48, 515841.3, 11931.24, 0, 1, 0]])
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:

<p>Lab-4</p> <p>Solve using matrix method (linear reg)</p> <p>value of X & Y find with help</p> $X = \begin{bmatrix} 1 & 2 \\ 1 & 4 \\ 1 & 5 \\ 1 & 9 \end{bmatrix}$ $X^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$ $X^T X = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$ $(X^T X)^{-1} = \begin{bmatrix} 9 & 10 \\ 10 & 30 \end{bmatrix}^{-1} = \begin{bmatrix} 1.057 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$ $(X^T X)^{-1} X^T Y = \begin{bmatrix} -0.5 \\ 0.2 \end{bmatrix} \rightarrow \text{Intercept } (\beta_0)$ $Y = \beta_0 + \beta_1 X + \epsilon$	<p>Date / / Page / /</p> <p>10.00</p> <p>Linear Regression</p> <p>Predict Canada's per capita income in year 2020.</p> <pre> import pandas as pd import numpy as np import matplotlib as plt df = pd.read_csv("canada.csv") print(df.describe()) print(df.info()) plt.scatter(data['year'], data['income'], color='blue', label='Actual data') plt.xlabel('year') plt.ylabel('per capita income') plt.legend() plt.show() x = data['year'] y = data['income'] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) model = LinearRegression() model.fit(x_train, y_train) y_pred = model.predict(x_test) </pre>
---	---

```

print("coeff: ", model.coef[0])
print("Intercept: ", model.intercept[0])
y_2020 = model.predict([2020])
print("y_2020: ", y_2020)

```

Metrics

```

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

```

```

print(mae)
print(mse)
print(r2)

```

Output

```

coeff: 815.14
Intercept: -1605760.19
Predicted Income: 410276.68
MAE: 3240.91
MSE: 15147845.57
R-squared: 0.98

```

Q) predict the salary of Employee

Output

```

coeff: 9569.79
Intercept: 95499.97
Prediction salary: 140257
MAE: 4519.58
MSE: 27180506.86
R-squared (R2) error: 0.96

```

Multiple Linear regression

1) Write python code to implement & find the salary of future implemented employee.

```

import pandas as pd
import numpy as np

```

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

```
data['exp'] = data['exp'].apply(lambda x: np.nan if x == 'NaN' else float(x))
data['level'] = data['level'].apply(lambda x: np.nan if x == 'NaN' else float(x))
```

```
print(data.describe())
print(data.info())
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Code:

```
from google.colab import files
hr=files.upload()
```

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model
import seaborn as sns

```

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df1=pd.read_csv("HR_comma_sep.csv")
df1.head()
df1.isnull().sum()
plt.figure(figsize=(12, 6))
sns.barplot(x='Department', y='left', data=df1)
plt.title('Employee Retention Rate by Department')
plt.xlabel('Department')
plt.ylabel('Proportion of Employees Left')
plt.xticks(rotation=45, ha='right')
plt.show()

ohe = OneHotEncoder(handle_unknown='ignore',
sparse_output=False) department_encoded =
ohe.fit_transform(df1[['Department']])
department_encoded_df = pd.DataFrame(department_encoded,
columns=ohe.get_feature_names_out(['Department']))
df1 = pd.concat([df1, department_encoded_df], axis=1)
df1 = df1.drop('Department', axis=1)
ordinal_encoder = OrdinalEncoder(categories=[[ 'low', 'medium', 'high']],
dtype=np.int64) salary_encoded =
ordinal_encoder.fit_transform(df1[['salary']])
df1['salary_encoded'] = salary_encoded
df1 = df1.drop('salary', axis=1)
df1.head()

correlation_matrix = df1.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f") plt.title('Correlation Matrix of Features')
plt.show()
plt.figure(figsize=(8, 6))
sns.barplot(x='salary_encoded', y='left', data=df1)
plt.title('Impact of Employee Salary on Retention')
plt.xlabel('Salary Level (Encoded)')
plt.ylabel('Proportion of Employees Left')
plt.show()

df_copy = df1[['number_project', 'average_montly_hours', 'time_spend_company',
'left','salary_encoded', 'satisfaction_level','Work_accident']]
df_copy.head()
X = df_copy.drop('left', axis=1)
y = df_copy['left']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

```

```

print(f"Accuracy of the Logistic Regression model: {accuracy}")

from google.colab import files
zodata=files.upload()
zotype=files.upload()

zoo_data = pd.read_csv('zoo-data.csv')
zoo_class = pd.read_csv('zoo-class-type.csv')
merged_data = pd.merge(zoo_data, zoo_class, left_on='class_type',
right_on='Class_Number') merged_data = merged_data.drop(['Animal_Names',
'Number_Of_Animal_Species_In_Class',
'Class_Number','class_type','animal_name'], axis=1)
X = merged_data.drop('Class_Type', axis=1)
y = merged_data['Class_Type']
print(merged_data.head())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test)) disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix")
plt.show()

```

Program 5

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

Screenshot:

Lab -5 Decision tree				
Date _____	Page _____			
a)	Instance	a_2	a_3	class
1	Hot	High	No	
2	Hot	High	No	
3	Cool	High	No	
7	Hot	High	No	
8	Hot	Normal	Yes	
Entropy (I) = $-\frac{4}{5} \log_2 \left(\frac{4}{5}\right) - \frac{1}{5} \log_2 \left(\frac{1}{5}\right)$ = 0.7219				
for a_2				
$I_{hot} [1+, 3-] = -\frac{1}{4} \log_2 \left(\frac{1}{4}\right) - \frac{3}{4} \log_2 \left(\frac{3}{4}\right)$ = 0.8113				
$I_{cool} [0+, 1-] = 0$				
$Gain(I, a_2) = 0.7219 - \frac{4}{5} (0.8113) = 0.07286$				
for a_3				
$I_{high} [0+, 4-] = 0$				
$I_{normal} [1+, 0-] = 0$				
$Gain(I, a_3) = 0.7219 - 0 - 0 = 0.7219$				

Code:

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris.csv")
df1.head()

df1.isnull().sum()

X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns,
class_names=y.unique()) plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=clf.classes_) cmap = plt.cm.get_cmap('PuBuGn')
disp.plot(cmap=cmap)
plt.show()

drug=files.upload()
df2=pd.read_csv("drug.csv")
df2.head()
df2.isnull().sum()

label_encoders = {}
for column in df2.columns:
    le = LabelEncoder()
    df2[column] = le.fit_transform(df2[column])
    label_encoders[column] = le
X = df2.drop('Drug', axis=1)
y = df2['Drug']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(c) for c in
y.unique()]) plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
```

```

cmap = plt.cm.Blues
disp.plot(cmap=cmap)
plt.show()

pc=files.upload()
df3=pd.read_csv("petrol_consumption.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('Petrol_Consumption', axis=1)
y = df3['Petrol_Consumption']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
y_pred =
regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error:
{rmse:.2f}') print(f'Mean Absolute Error:
{mae:.2f}') print(f'R-squared: {r2:.2f}')
plt.figure(figsize=(30, 30))
plot_tree(regressor, filled=True, feature_names=X.columns,
fontsize=10) plt.show()

```

Program 6

Build KNN Classification model for a given dataset.

Screenshot :

Date _____
Page _____

Lab - 6 - K.NN. Codes

Q1) Apply the KNN to predict the score for IRIS flower dataset.

import pandas as pd

```
irisdata = pd.read_csv("iris.csv")
x, y = irisdata.iloc[:, :-1], irisdata['species']
```

x_train, x_test, y_train, y_test
= train_test_split(x, y, test_size=0.2,
random_state=42, stratify=y)

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)

print("accuracy score: ", accuracy_score(y_test, y_pred))
print("confusion matrix: ", confusion_matrix(y_test, y_pred))
print("classification report: ", classification_report(y_test, y_pred))
```

Output .. Accuracy score: 1.0 ..
confusion matrix ..

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

Classification report

	precision	recall	f1-score	support
setosa	1.00	1.00	0.98	50
Versicolor	1.00	1.00	0.95	54
Virginica	1.00	1.00	0.90	54

	accuracy	precision	recall	f1-score	support
macro avg	1.00	1.00	0.64	154	
weighted avg	1.00	1.00	0.70	154	

Q2) Diabetes dataset

Accuracy : 0.701

confusion matrix
 $\begin{bmatrix} 80 & 20 \\ 26 & 28 \end{bmatrix}$

Classification report

	precision	recall	f1-score	support
0	0.75	0.80	0.78	100
1	0.58	0.52	0.55	54

Accuracy

	macro avg	0.67	0.66	0.66	154
weighted avg	0.69	0.70	0.70	0.70	154

Q3) heart dataset

Accuracy : 0.672

confusion matrix

$\begin{bmatrix} 17 & 17 \\ 9 & 24 \end{bmatrix}$

Classification report

	precision	recall	f1-score	support
0	0.65	0.61	0.63	28
1	0.69	0.73	0.71	33

accuracy

	macro avg	0.67	0.67	0.67	61
weighted avg	0.67	0.67	0.67	0.67	61

Questions

- 1) for iris dataset, how to choose the k value? demonstrate using accuracy & error rate

To determine the best k in knn, we test multiple values & analyze both acc & error rate. A very small k value leads to overfitting, very large k value leads to underfitting.

Code:

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (2).csv")
df1.head()
df1.isnull().sum()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
best_k = 1
best_accuracy = 0
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
```

```

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-accuracy}")if accuracy >
    best_accuracy: best_accuracy = accuracy best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

diabetes=files.upload()
df2=pd.read_csv("diabetes.csv")
df2.head()
df2.isnull().sum()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df2.drop('Outcome', axis=1))
X_train, X_test, y_train, y_test = train_test_split(X_scaled, df2['Outcome'], test_size=0.2,
random_state=42) best_k = 1
best_accuracy = 0
for k in range(1,
11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train) y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
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()
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

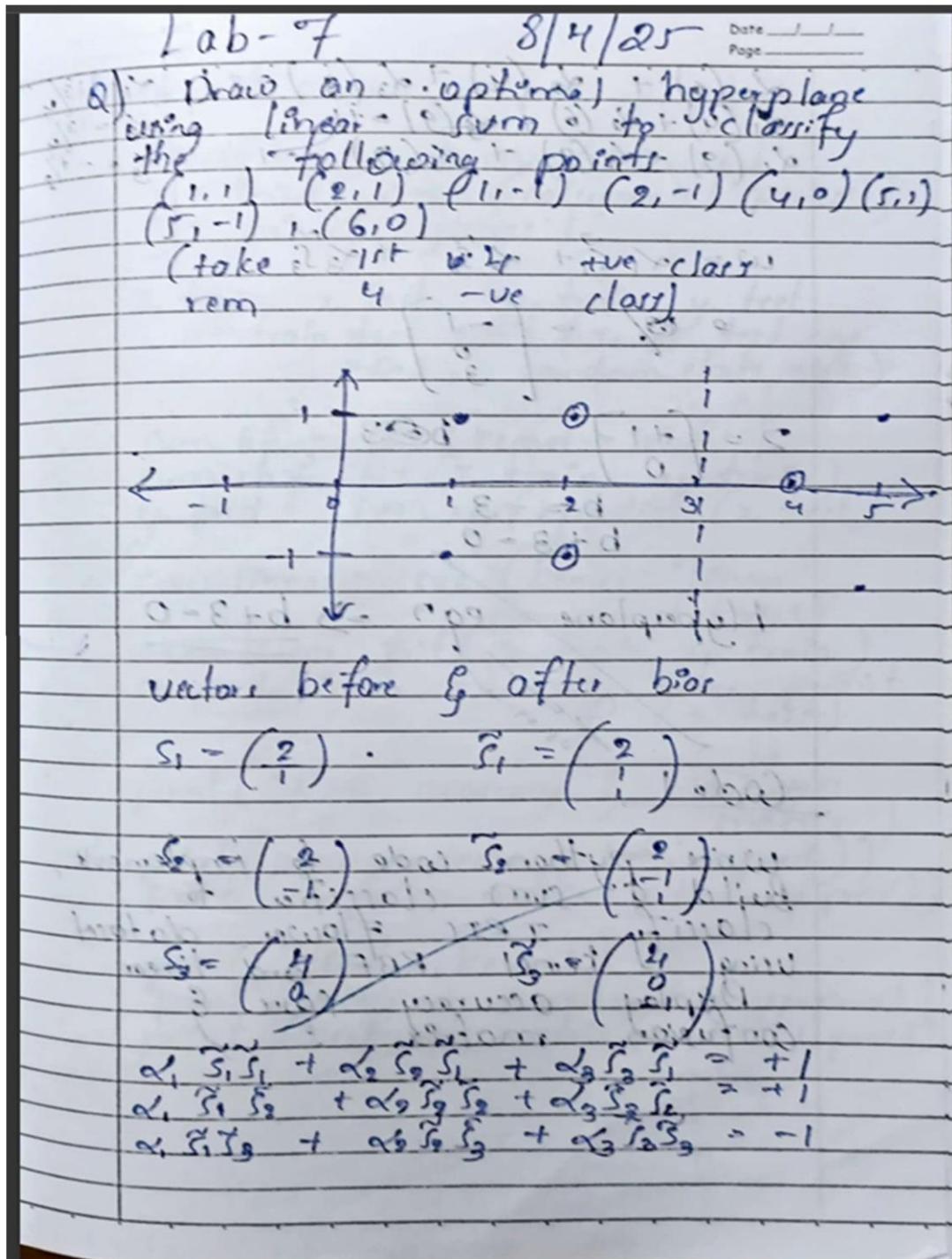
heart=files.upload()
df3=pd.read_csv("heart.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('target', axis=1)
y = df3['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) best_k = 1
best_accuracy = 0
for k in range(1,
11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-
accuracy}") if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

```

Program7

Build Support vector machine model for a given dataset

Screenshot:



$\alpha_1(0) + \alpha_2(4) + \alpha_3(9) = +1$; $x_1 = 13/4$
 $\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1$; $x_2 = 13/4$
 $\alpha_1(9) + \alpha_2(7) + \alpha_3(13) = -1$; $x_3 = -7/2$
 $w = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3$
 $\Rightarrow w = \begin{bmatrix} -1 \\ 0 \\ 3 \end{bmatrix}$
 $b = \begin{bmatrix} +1 \\ 0 \end{bmatrix}$; $b = 3$
 $b = -3$
 $b + 3 = 0$
 Hyperplane eqn $\Rightarrow b + 3 = 0$

Code 1:
 Write a python code to implement, build a svm classifier to classify IRIS flower dataset using kernel RBF and linear. Display accuracy score & confusion matrix.

import pandas as pd
 $df = pd.read_csv('iris.csv')$
 $x = df.drop(['species'], axis=1)$
 $y = df['species']$
 $x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)$
 $rbf_svm = SVC(kernel='rbf')$
 $rbf_svm.fit(x_train, y_train)$
 $rbf_y_pred = rbf_svm.predict(x_test)$
 $print("RBF Kernel SVM:")$
 $print("Accuracy:", accuracy_score(y_test, rbf_y_pred))$
 $cm = confusion_matrix(y_test, rbf_y_pred)$

Code:

```

from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (1).csv")
df1.head()
X=df1.drop('species', axis=1)
y=df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_y_pred = rbf_svm.predict(X_test)
print("RBF Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, rbf_y_pred))
cm = confusion_matrix(y_test, rbf_y_pred)

```

```

sns.heatmap(cm, annot=True, fmt='d', cmap="Blues")
plt.title('Confusion Matrix for RBF Kernel SVM')
plt.xlabel('Predicted')
plt.ylabel('True') plt.show()
print(classification_report(y_test, rbf_y_pred))
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
linear_y_pred = linear_svm.predict(X_test)
print("\nLinear Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, linear_y_pred))
cm = confusion_matrix(y_test, linear_y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues")
plt.title('Confusion Matrix for Linear Kernel SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, linear_y_pred))
letter=files.upload()
df2=pd.read_csv("letter-recognition.csv")
df2.head()
X = df2.drop('letter', axis=1)
y = df2['letter']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) svm_classifier = SVC(kernel='linear', probability=True)
svm_classifier.fit(X_train, y_train)
y_pred =
svm_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues")
plt.title('Confusion Matrix for SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
lb = LabelBinarizer()
lb.fit(y_test)

y_test_lb = lb.transform(y_test)
y_pred_prob =
svm_classifier.predict_proba(X_test) fpr = { }
tpr = { }
thresh = { }
roc_auc = dict()
n_class = y_test_lb.shape[1]
for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test_lb[:,i], y_pred_prob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])

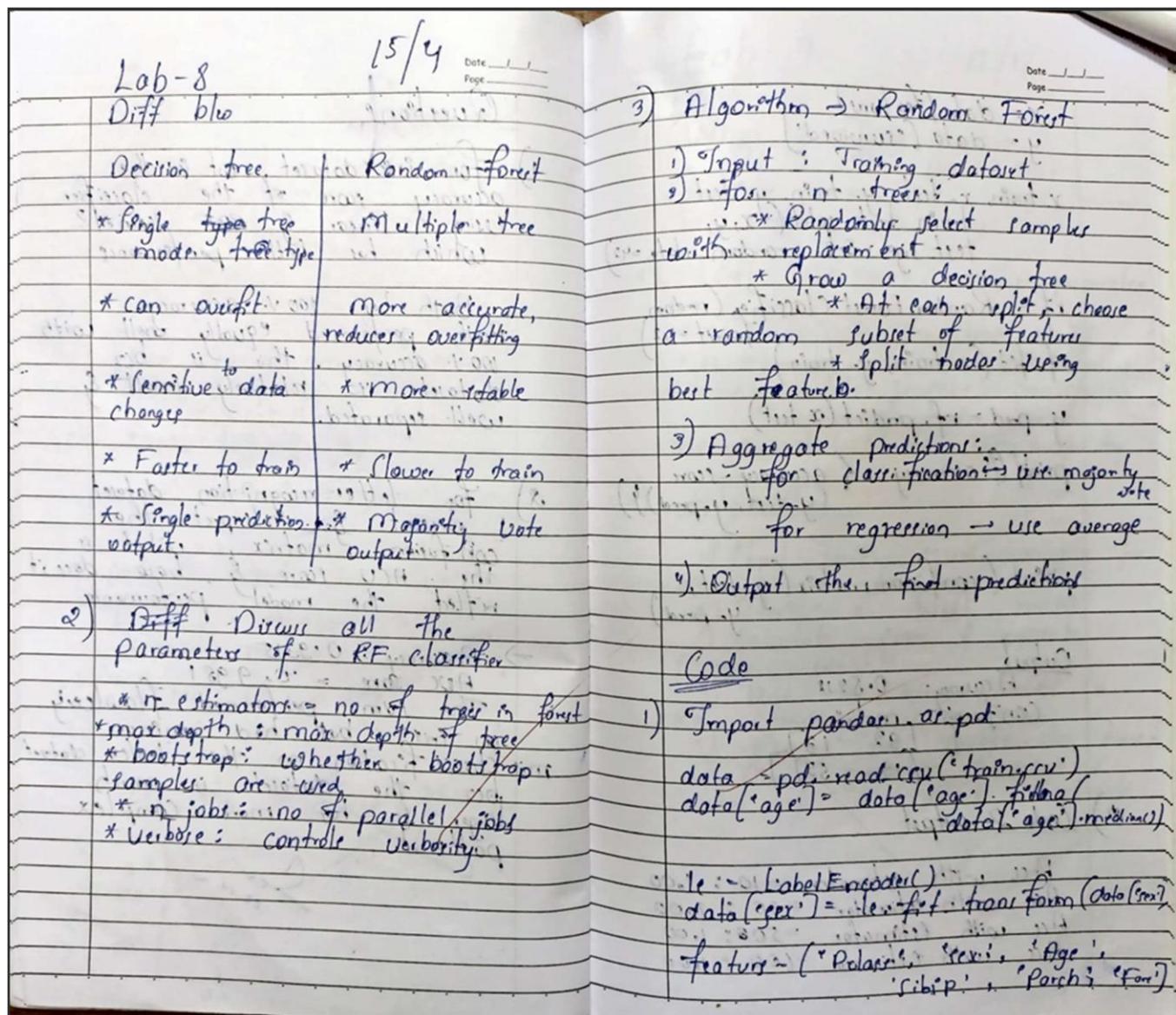
```

```
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='SVM (AUC = %0.2f)' %  
roc_auc[0]) plt.title('ROC Curve for Class 0')  
plt.xlabel('False Positive  
Rate') plt.ylabel('True Positive  
rate') plt.legend(loc='best')  
plt.show()  
print(f"AUC score for class 0: {roc_auc[0]}")
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:



Code:

```

from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (4).csv")
df1.head()
X=df1.drop('species', axis=1)
y=df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
rf_classifier = RandomForestClassifier(random_state=0)
rf_classifier.fit(X_train, y_train)
y_pred =
rf_classifier.predict(X_test)

```

```

default_accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with default n_estimators: {default_accuracy}")
best_accuracy = 0
best_n_estimators = 0
for n_estimators in range(1, 101):
    rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=0)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy: best_accuracy
        = accuracy best_n_estimators =
            n_estimators
print(f"\nBest accuracy: {best_accuracy} achieved with n_estimators =
{best_n_estimators}") cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:

Lab - 9 15/4 Date _____
Page _____

Ques.

- 1) What is AdaBoosting?
It is the process that combines multiple weak learners to create a strong learner
- 2) List AdaBoost Classifier parameters
 - * estimator: The base model
 - * n_estimators: no. of weak learners
 - * learning_rate: shrinkage contribution of each learner
 - * random_state: for reproducibility
- 3) Algorithm.
 - 1) Start with equal weights for all training samples
 - 2) Train a weak model
 - 3) Calculate error & update sample weights
 - 4) Add a weak model to the ensemble
 - 5) Repeat for n-estimator times
 - 6) final prediction = weighted votes of all weak models

Ans.

Date _____
Page _____

Code

```

1. ins = load_csv()
2. X = ins.data
3. y = ins.target
4. x_train, x_test, y_train, y_test =
   train_test_split(x, y, test_size=0.2)
base_classifier = DecisionTreeClassifier(
    max_depth=5),
adaBoost = AdaBoostClassifier(base_estimator=
    base_classifier,
    n_estimators=50, learning_rate=1.0)
adaBoost.fit(x_train, y_train)
y_pred = adaBoost.predict(x_test)
accuracy = accuracy(y_test, y_pred)
conf_matrix = confusion_matrix(
    y_test, y_pred)

```

Output

- Accuracy: 0.933
- Conf matrix

10	0.73
0.48	1.55
0.1	1.0

Code:

```

from google.colab import files
income=files.upload()
df1=pd.read_csv("income.csv")
df1.head()
X = df1.drop('income_level', axis=1)
y = df1['income_level']
X = pd.get_dummies(X)

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) abc = AdaBoostClassifier(n_estimators=10,
random_state=42)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Initial AdaBoost accuracy (10 trees): {accuracy}")
param_grid = {'n_estimators': [50, 100, 150, 200]}
grid_search = GridSearchCV(AdaBoostClassifier(random_state=42), param_grid, cv=5,
scoring='accuracy') grid_search.fit(X_train, y_train)
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_}")
best_abc = grid_search.best_estimator_
y_pred_best = best_abc.predict(X_test)
best_accuracy = accuracy_score(y_test,
y_pred_best)
print(f"Accuracy of the best model on the test set: {best_accuracy}")
cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

```

Program 10

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

Screenshot:

Lab-10 K-Means

1) Write K-means algo

- 1) choose no. of clusters k
- 2) initialize "k" centroids
- 3) Assign each point to nearest centroid
- 4) Recompute centroids of the mean of assigned points
- 5) Repeat steps 3-4 until centroids stop changing.

2) How to determine no. of clusters

- * Use elbow method
- * try diff values of k
- * calculate SSE
- * Plot SSE vs k

3) Parameters of K-means, don't

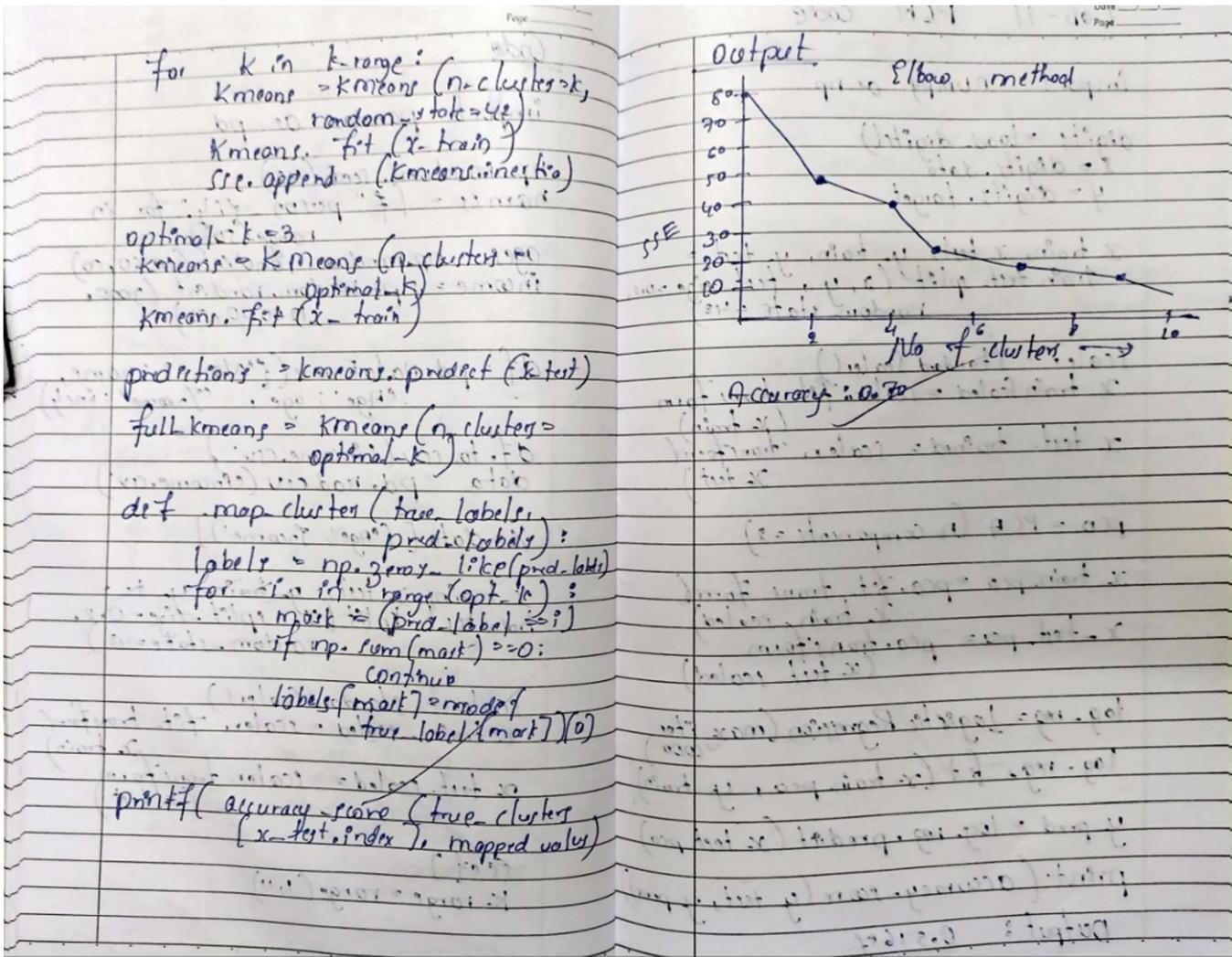
- n_clusters : no of clusters
- init : initialization method
- n_init : no. of times algo
- max_iter : max iterations per run
- tol : tolerance to declare convergence

Code

```

import pandas as pd
np.random.seed(42)
names = [f"person_{i}" for i in
range(50)]
age = np.random.randint(10,60,50)
income = np.random.randint(3000,
12000,50)
df = pd.DataFrame({"name": names,
"age": age, "income": income})
df.to_csv("income.csv")
data = pd.read_csv("income.csv")
X = data[["age", "income"]]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
SSE = []
K_range = range(1,11)

```



Code:

```

from google.colab import files
iris=files.upload()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df1=pd.read_csv("iris (4).csv")
df1.head()
df=df1.drop(['sepal_length','sepal_width','species'],axis=1)
scaler = StandardScaler()
    
```

```

scaled_df = scaler.fit_transform(df) wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
                    random_state=0) kmeans.fit(scaled_df)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10,
                random_state=0) pred_y = kmeans.fit_predict(scaled_df)
df['cluster'] = pred_y
plt.scatter(df['petal_length'], df['petal_width'], c=df['cluster'])
plt.title('Clusters of Iris Flowers')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()

```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA)

method.

Screenshot:

Lab-11 PCA code

```
import numpy as np
digits = load_digits()
X = digits.data
y = digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_pca, y_train)
y_pred = logreg.predict(X_test_pca)
print(accuracy_score(y_test, y_pred))
Output: 0.51607
```

code & output

svm accuracy : 0.8869
logistic Reg acc : 0.886
R.F accuracy : 0.8889

Best model : svm
No of PCA components : 10
svm acc with PCA : 0.8904
Acc. change with import: 0.008

Ques. 1.1

Lab-11 29/4/25

Date _____
Page _____

Principle Component Analysis
(PCA)

Given the data in table, reduce the dim from 2 to 1

Feature	cx1	cx2	cx3	cx4	...
x_1	4	8	13	7	
x_2	11	4	5	14	

i) mean

$$\bar{x}_1 = \frac{1}{4} (8 + 4 + 13 + 7) = 8$$

$$\bar{x}_2 = \frac{1}{4} (11 + 4 + 5 + 14) = 8.5$$

ii) Covariance matrix

$$\text{cov}(x_1, x_1) = \frac{1}{n-1} \sum_{k=1}^n (x_{1k} - \bar{x}_1)^2$$

$$= \frac{1}{4-1} ((4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2)$$

$$= 14$$

$$\text{cov}(x_2, x_2) = \frac{1}{4-1} \left((11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2 \right)$$

$$= 29$$

$$\text{cov}(x_1, x_2) = \text{cov}(x_2, x_1) = \frac{1}{n-1} \sum_{k=1}^n (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2) = -11$$

Date _____
Page _____

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

$\Rightarrow S = \begin{bmatrix} 14 & -4 \\ -4 & 23 \end{bmatrix}$

(iii) Eigen values

$$\begin{bmatrix} 14 - \lambda & -4 \\ -4 & 23 - \lambda \end{bmatrix}$$

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

$$\lambda_1 = 37 + \sqrt{566}, \quad \lambda_2 = 37 - \sqrt{565}$$

$$\lambda_1 = 30.3849, \quad \lambda_2 = 6.6151$$

$$\lambda_1 > \lambda_2$$

(iv) Eigen vector for largest eigen value

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

$$= \begin{bmatrix} 14 - \lambda_1 & -4 \\ -4 & 23 - \lambda_1 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = ((14 - \lambda_1)U_1 - 4U_2) \quad (1)$$

$$= -4U_1 + (23 - \lambda_1)U_2 \quad (2)$$

$$U_1 = U_2 = t_1$$

$$14 - \lambda_1 = 23 - \lambda_1$$

Date _____
Page _____

$\|t_1\| = \sqrt{11^2 + (14 - \lambda)^2} = 10.7348$

unit eigen vector to λ_1 is

$$e_1 = \frac{1}{\|t_1\|} \begin{bmatrix} 11 \\ 14 - \lambda_1 \end{bmatrix}$$

$$e_1 = \begin{bmatrix} 0.5574 \\ 0.8303 \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$$

(v) Let $\begin{bmatrix} x_{ik} \\ x_{ek} \end{bmatrix}$

$$e_1^T \begin{bmatrix} x_{ik} \\ x_{ek} \end{bmatrix} = (0.5574 \quad 0.8303)$$

$$= 0.5574(x_{ik} - x_1) - 0.8303(x_{ek} - x_2)$$

A algorithm of PCA

- 1) Calculate mean.
- 2) Calculate covariance matrix.
- 3) Eigen val of covariance matrix
- 4) Computation of eigen vectors
unit eigen vector
- 5) Computation of first principle component
- 6) Geometric meaning of 1st principle component

Code:

```
from google.colab import files
heart=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.preprocessing import LabelEncoder,
OneHotEncoder from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

```

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

df1=pd.read_csv("heart (1).csv")
df1.head()
text_cols = df1.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in text_cols:
    df1[col] =
label_encoder.fit_transform(df1[col])
print(df1.head())
X = df1.drop('HeartDisease', axis=1)
y = df1['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f"SVM Accuracy: {svm_accuracy}")

# Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train) lr_predictions =
lr_model.predict(X_test) lr_accuracy =
accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy}")

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")

models = {
    "SVM": svm_accuracy,
    "Logistic Regression":
        lr_accuracy, "Random Forest":
            rf_accuracy
}

best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)

```

```

X_test_pca = pca.transform(X_test)

svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")

lr_model_pca = LogisticRegression(random_state=42)
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")

rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")

models_pca = {
    "SVM": svm_accuracy_pca,
    "Logistic Regression": lr_accuracy_pca,
    "Random Forest": rf_accuracy_pca
}

best_model_pca = max(models_pca, key=models_pca.get)
print(f"\nBest Model (with PCA): {best_model_pca} with accuracy {models_pca[best_model_pca]}")

```