# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Ganesh Bhandari (1BM22CS098)**

#### 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 **Ganesh Bhandari (1BM22CS098),** 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.

**Dr. Kavitha Sooda** Professor & HOD

Department of CSE, BMSCE

**[Supreeth S](https://bmsce.ac.in/home/facultyProfile/649/Supreeth-S)**

Assistant Professor Department of CSE, BMSCE

# Index

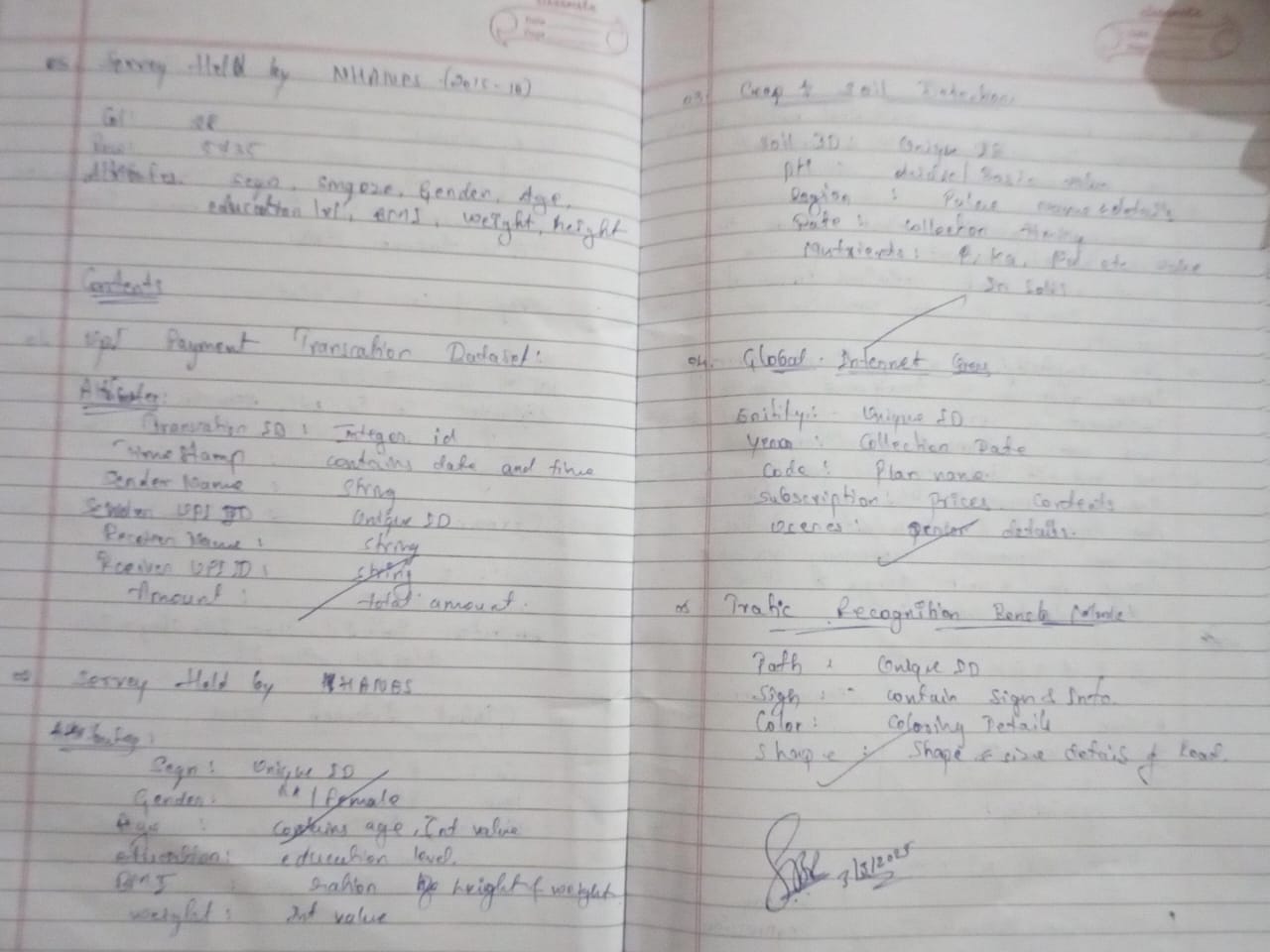
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 17-3-2025 | Write a python program to import and export data using Pandas library functions |  |
| 2 | 17-3-2025 | Demonstrate various data pre-processing techniques for a given dataset |  |
| 3 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. |  |
| 4 | 11-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset |  |
| 5 | 11-3-2025 | Build Logistic Regression Model for a given dataset |  |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset. |  |
| 7 | 7-4-2025 | Build Support vector machine model for a given dataset |  |
| 8 | 21-4-2025 | Implement Random Forest ensemble method on a given dataset. |  |
| 9 | 5-5-2025 | Implement Boosting ensemble method on a given dataset. |  |
| 10 | 5-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. |  |
| 11 | 12-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. |  |

Github Link: <https://github.com/GaneshVBhandari/ML-LAB>

##### Program 1

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

Screenshot:



Code:

import pandas as pd

# ----------- Step 1: Import Data from CSV -----------

# If you already have a CSV file, replace 'data.csv' with the filename.

# For this demo, let's create a DataFrame and save it to CSV first.

data = {

    'Name': ['Alice', 'Bob', 'Charlie'],

    'Age': [24, 30, 22],

    'City': ['New York', 'Los Angeles', 'Chicago']

}

# Create DataFrame

df = pd.DataFrame(data)

# Exporting DataFrame to CSV

df.to\_csv('people.csv', index=False)

print("Data exported to 'people.csv'.")

# ----------- Step 2: Import Data from CSV -----------

imported\_df = pd.read\_csv('people.csv')

print("\nData imported from 'people.csv':")

print(imported\_df)

# ----------- Step 3: Export Data to Excel -----------

df.to\_excel('people.xlsx', index=False)

print("\nData exported to 'people.xlsx'.")

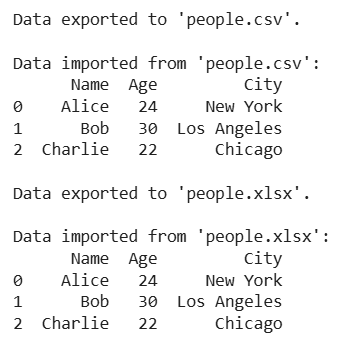
# ----------- Step 4: Import Data from Excel -----------

imported\_excel\_df = pd.read\_excel('people.xlsx')

print("\nData imported from 'people.xlsx':")

print(imported\_excel\_df)

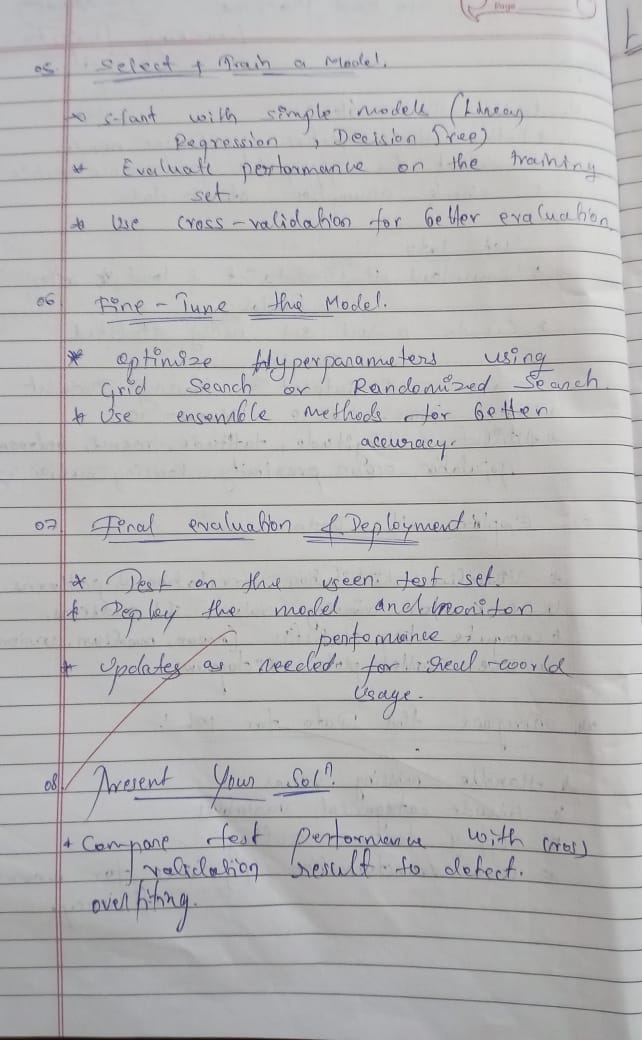
OUTPUT:

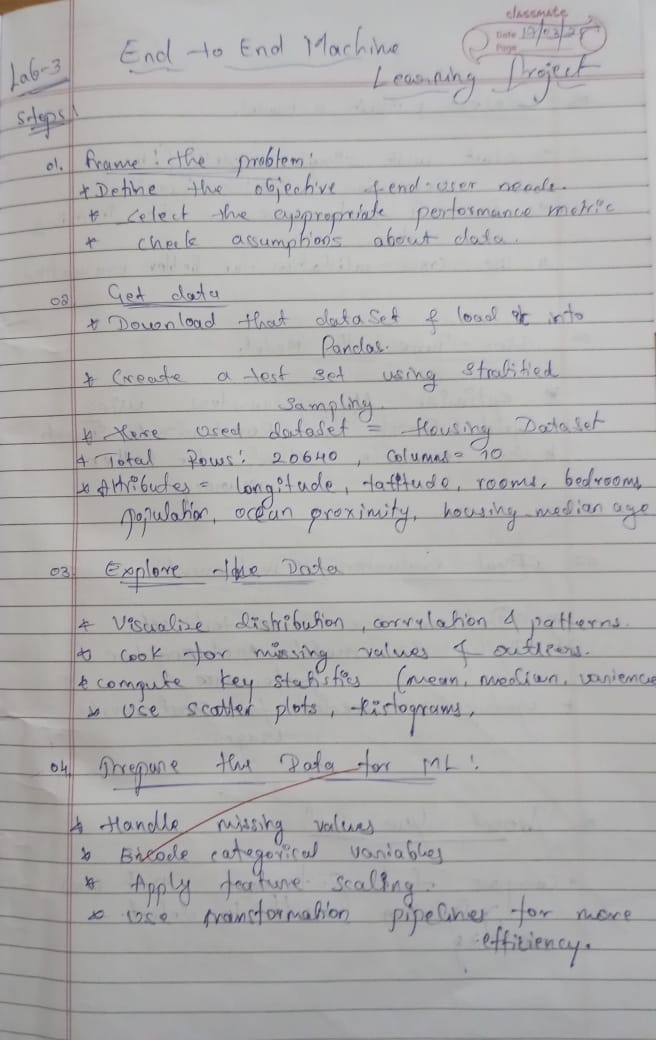


##### Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:





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()

import os

import tarfile

import urllib

import pandas as pd

DOWNLOAD\_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"

HOUSING\_PATH = os.path.join("datasets", "housing")

HOUSING\_URL = DOWNLOAD\_ROOT + "datasets/housing/housing.tgz"

def fetch\_housing\_data(housing\_url=HOUSING\_URL, housing\_path=HOUSING\_PATH):

    os.makedirs(housing\_path, exist\_ok=True)

    tgz\_path = os.path.join(housing\_path, "housing.tgz")

    urllib.request.urlretrieve(housing\_url, tgz\_path)

    housing\_tgz = tarfile.open(tgz\_path)

    housing\_tgz.extractall(path=housing\_path)

    housing\_tgz.close()

fetch\_housing\_data()

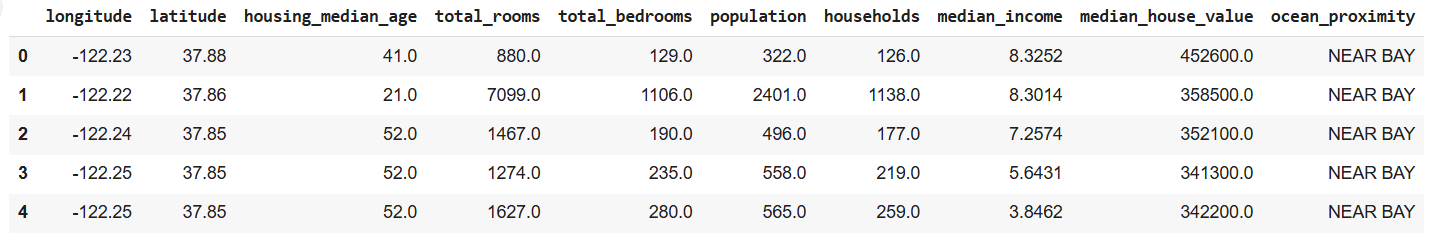
def load\_housing\_data(housing\_path=HOUSING\_PATH):

    csv\_path = os.path.join(housing\_path, "housing.csv")

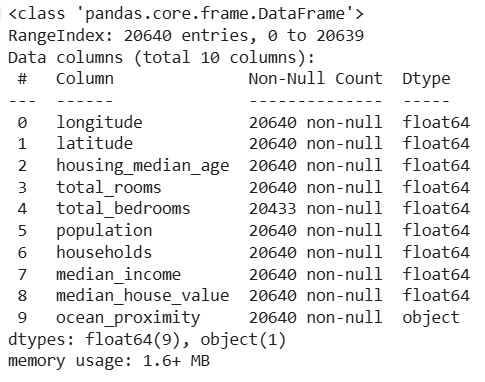
    return pd.read\_csv(csv\_path)

housing = load\_housing\_data()

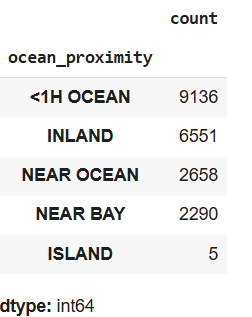
housing.head()



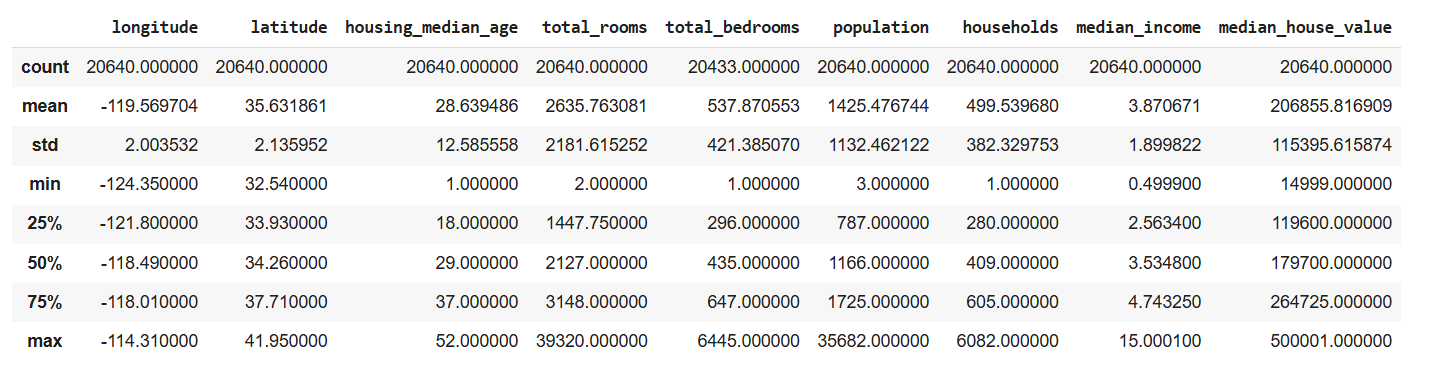
housing.info()



housing['ocean\_proximity'].value\_counts()



housing.describe()

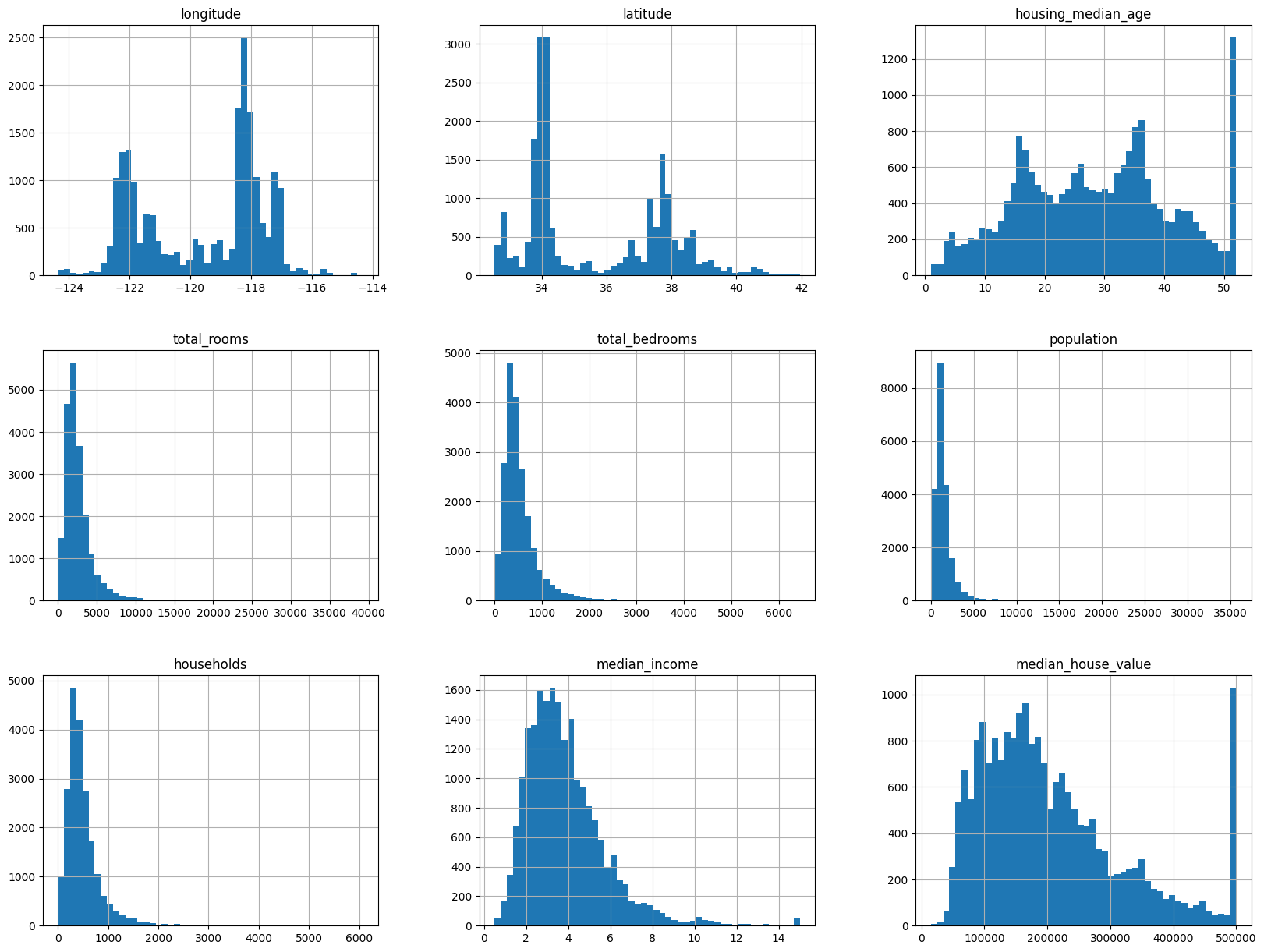


%matplotlib inline

import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(20,15))

plt.show()



import numpy as np

# def split\_train\_test(data, test\_ratio):

#   np.random.seed(42)

#   shuffled\_indices = np.random.permutation(len(data))

#   test\_set\_size = int(len(data) \* test\_ratio)

#   test\_indices = shuffled\_indices[:test\_set\_size]

#   train\_indices = shuffled\_indices[test\_set\_size:]

#   return data.iloc[train\_indices], data.iloc[test\_indices]

from sklearn.model\_selection import train\_test\_split, StratifiedShuffleSplit

train\_set, test\_set = train\_test\_split(housing, test\_size=0.2, random\_state=42)

len(test\_set)

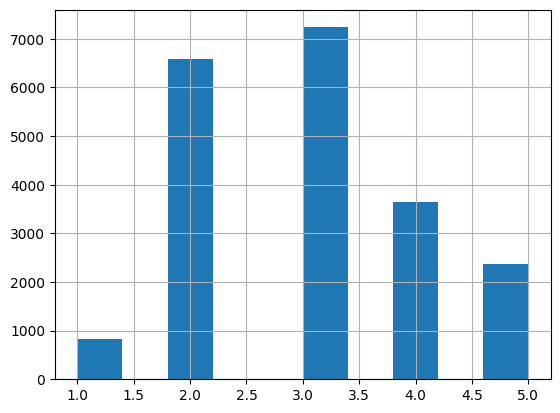


housing['income\_cat'] = pd.cut(housing['median\_income'],

                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],

                               labels=[1, 2, 3, 4, 5])

housing['income\_cat'].hist()



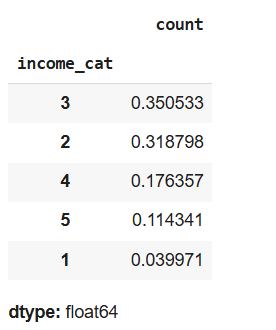
split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)

for train\_index, test\_index in split.split(housing, housing['income\_cat']):

    strat\_train\_set = housing.loc[train\_index]

    strat\_test\_set = housing.loc[test\_index]

strat\_test\_set['income\_cat'].value\_counts() / len(strat\_test\_set)

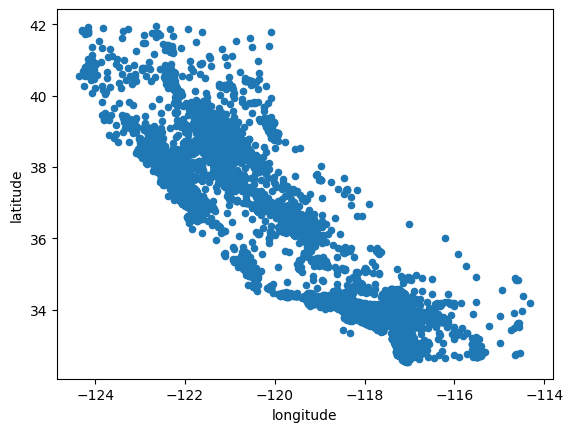


for set\_ in (strat\_train\_set, strat\_test\_set):

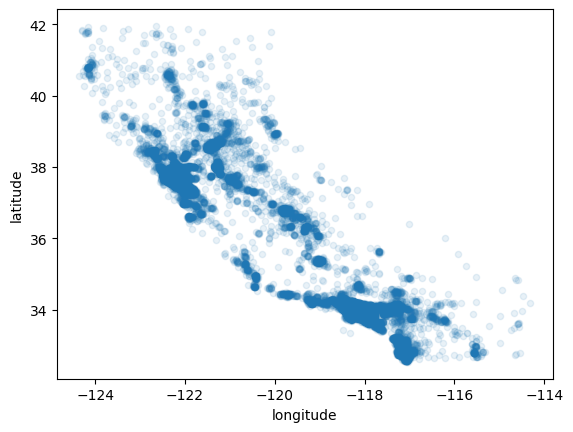
    set\_.drop('income\_cat', axis=1, inplace=True)

housing = strat\_train\_set.copy()

housing.plot(kind='scatter', x='longitude', y='latitude')



housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)

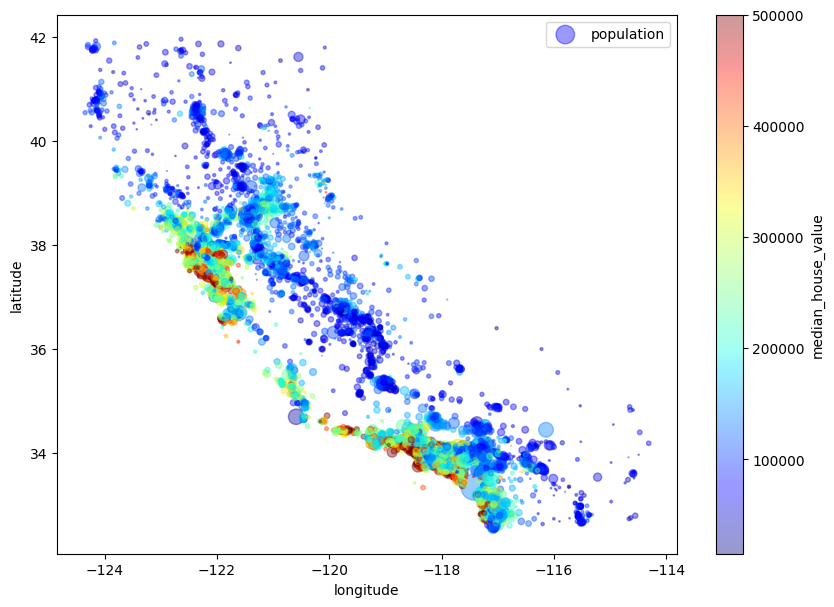


housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,

             s=housing['population']/100, label='population', figsize=(10, 7),

             c='median\_house\_value', cmap=plt.get\_cmap('jet'), colorbar=True)

plt.legend()

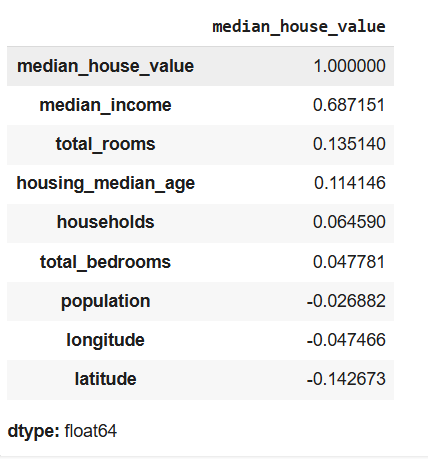


# Select only numerical features for correlation calculation

numerical\_features = housing.select\_dtypes(include=['number'])  # Exclude non-numeric columns

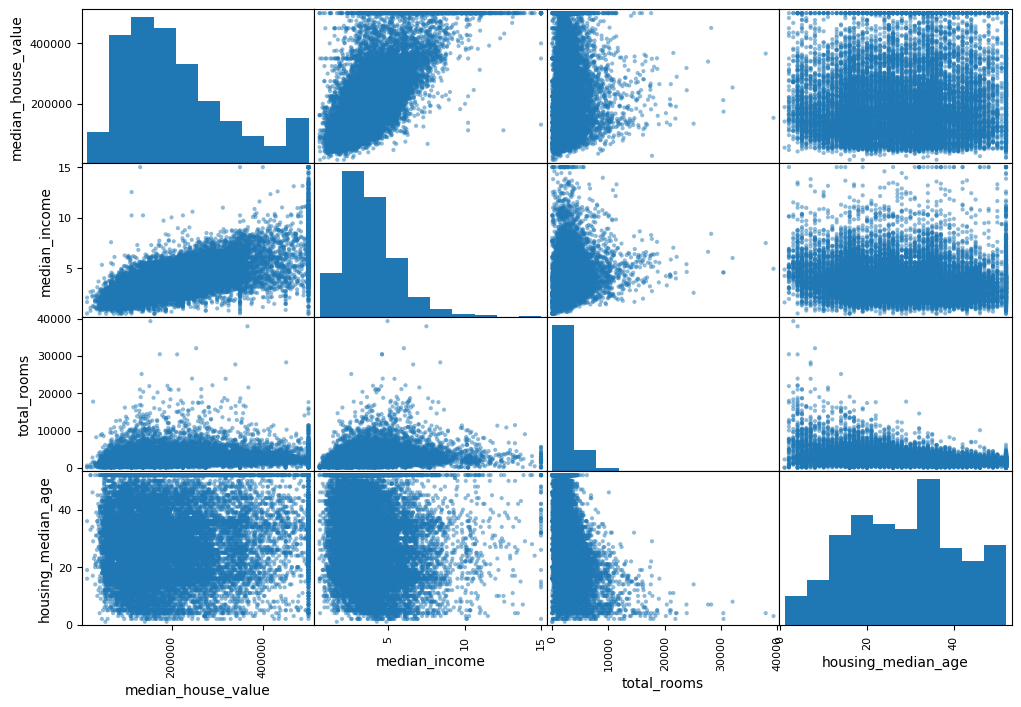
corr\_matrix = numerical\_features.corr()  # Calculate correlation for numerical features only

corr\_matrix['median\_house\_value'].sort\_values(ascending=False)

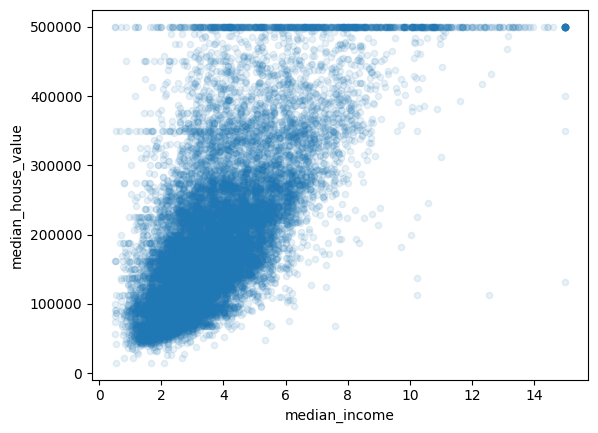
  
from pandas.plotting import scatter\_matrix

attributes = ['median\_house\_value', 'median\_income', 'total\_rooms', 'housing\_median\_age']

scatter\_matrix(housing[attributes], figsize=(12, 8))



housing.plot(kind='scatter', x='median\_income', y='median\_house\_value', alpha=0.1)



housing['rooms\_per\_household'] = housing['total\_rooms']/housing['households']

housing['bedrooms\_per\_room'] = housing['total\_bedrooms']/housing['total\_rooms']

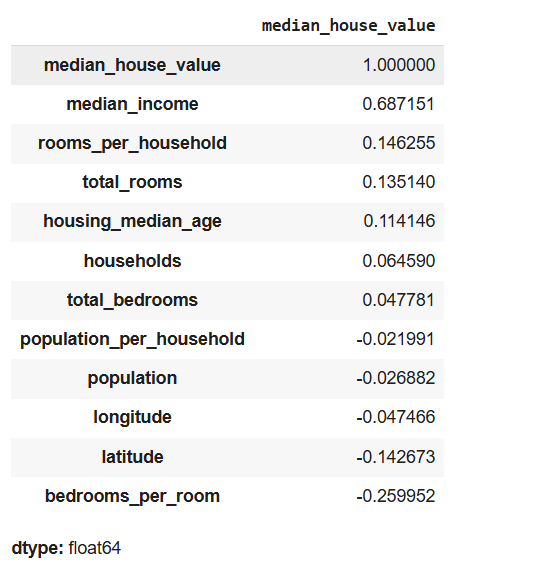
housing['population\_per\_household'] = housing['population']/housing['households']

# Select only numerical features for correlation calculation

numerical\_features = housing.select\_dtypes(include=['number'])  # Exclude non-numeric columns

corr\_matrix = numerical\_features.corr()  # Calculate correlation for numerical features only

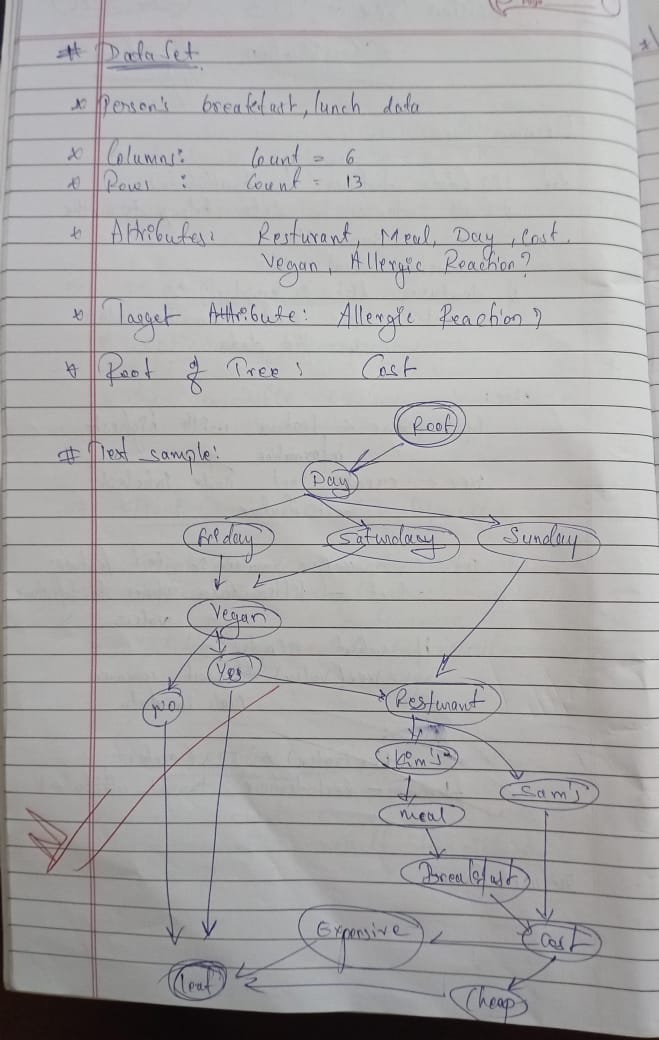
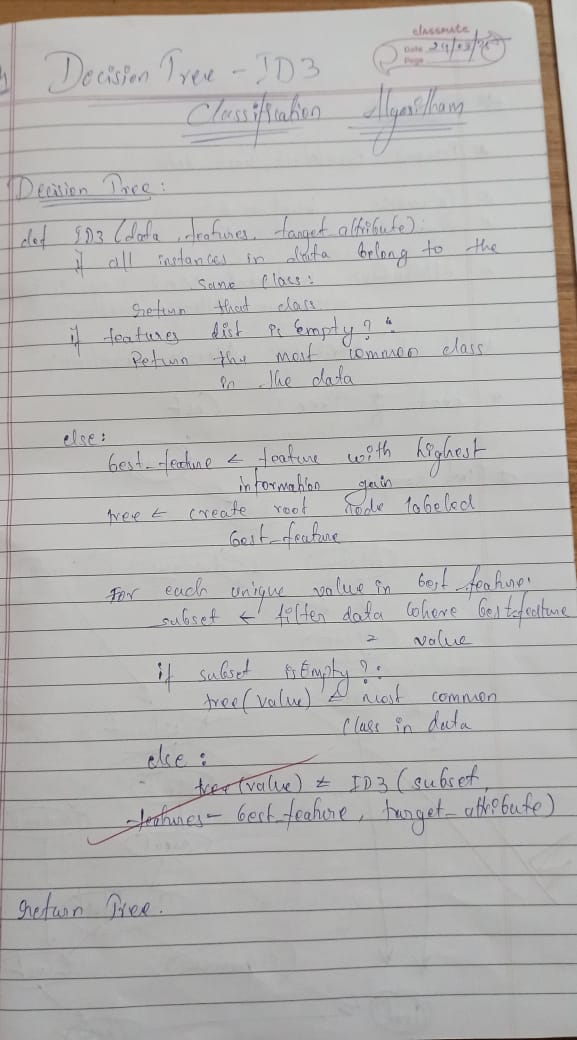
corr\_matrix['median\_house\_value'].sort\_values(ascending=False)



##### Program 3

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

Screenshot:



Code:

import pandas as pd

import numpy as np

# Load the dataset

file\_path = "/content/diabetes.csv"

df = pd.read\_csv(file\_path)

def entropy(df, col):

ppos = (df[col] == 1).sum()

pneg = len(df[col]) - ppos

total = ppos + pneg

if ppos == 0 or pneg == 0:

return 0

ppos /= total

pneg /= total

return -ppos \* np.log2(ppos) - pneg \* np.log2(pneg)

def information\_gain(df, feature, target):

total\_entropy = entropy(df, target)

values, counts = np.unique(df[feature], return\_counts=True)

weighted\_entropy = sum((counts[i] / sum(counts)) \* entropy(df[df[feature] == values[i]], target)

for i in range(len(values)))

return total\_entropy - weighted\_entropy

def id3(df, features, target):

# If all target values are the same, return that class

if len(df[target].unique()) == 1:

return df[target].iloc[0]

# If no features left, return the most common target value

if not features:

return df[target].mode()[0]

# Compute information gain for all features

info\_gains = {feature: information\_gain(df, feature, target) for feature in features}

# Print information gain for each feature

print("Information Gain for each feature:")

for feature, gain in info\_gains.items():

print(f"{feature}: {gain}")

# Select the best feature

best\_feature = max(info\_gains, key=info\_gains.get)

tree = {best\_feature: {}}

# Recur for each unique value in the selected feature

for value in df[best\_feature].unique():

sub\_df = df[df[best\_feature] == value].drop(columns=[best\_feature])

tree[best\_feature][value] = id3(sub\_df, [f for f in features if f != best\_feature], target)

return tree

def print\_tree(tree, indent=""):

if not isinstance(tree, dict):

print(indent + "→ " + str(tree))

return

for key, value in tree.items():

print(indent + str(key))

for sub\_key in value:

print(indent + " ├── " + str(sub\_key))

print\_tree(value[sub\_key], indent + " │ ")

# Compute the decision tree

features = [col for col in df.columns if col != "Outcome"]

decision\_tree = id3(df, features, "Outcome")

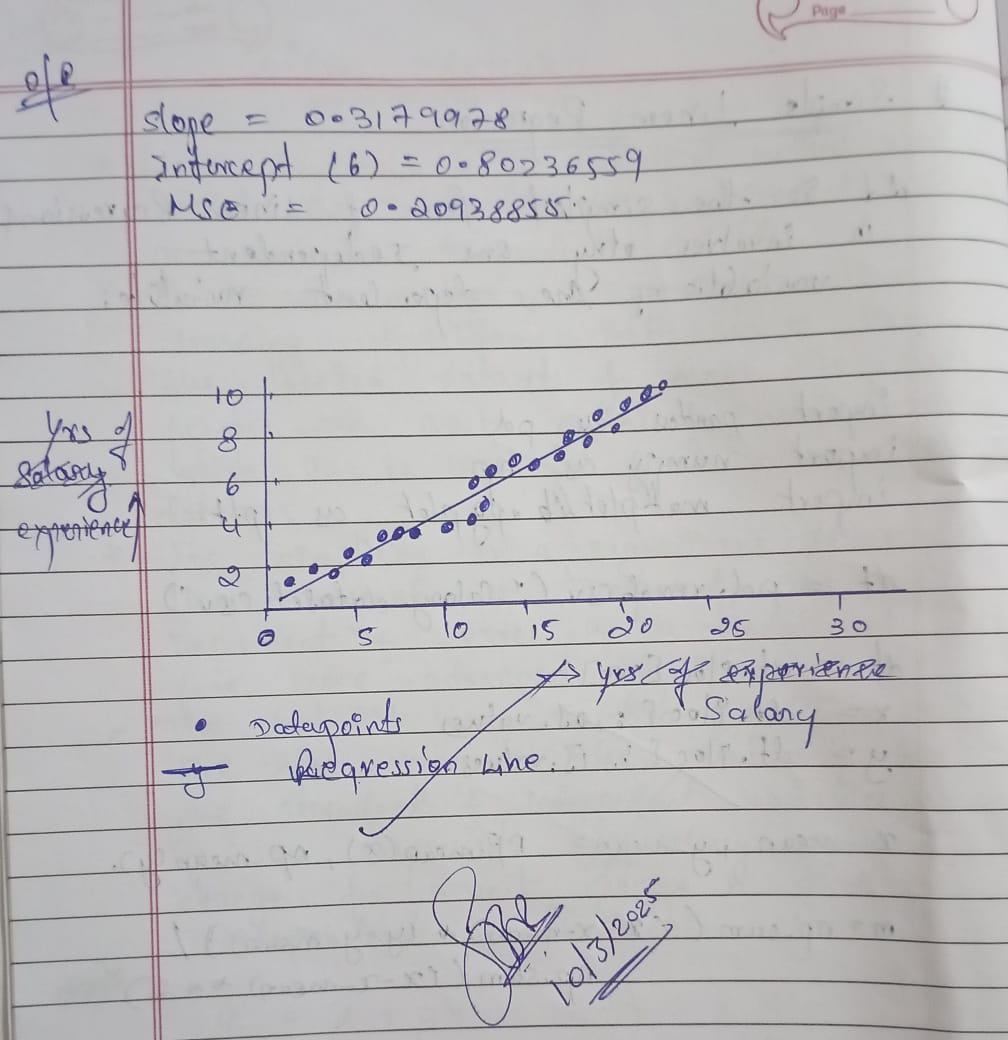
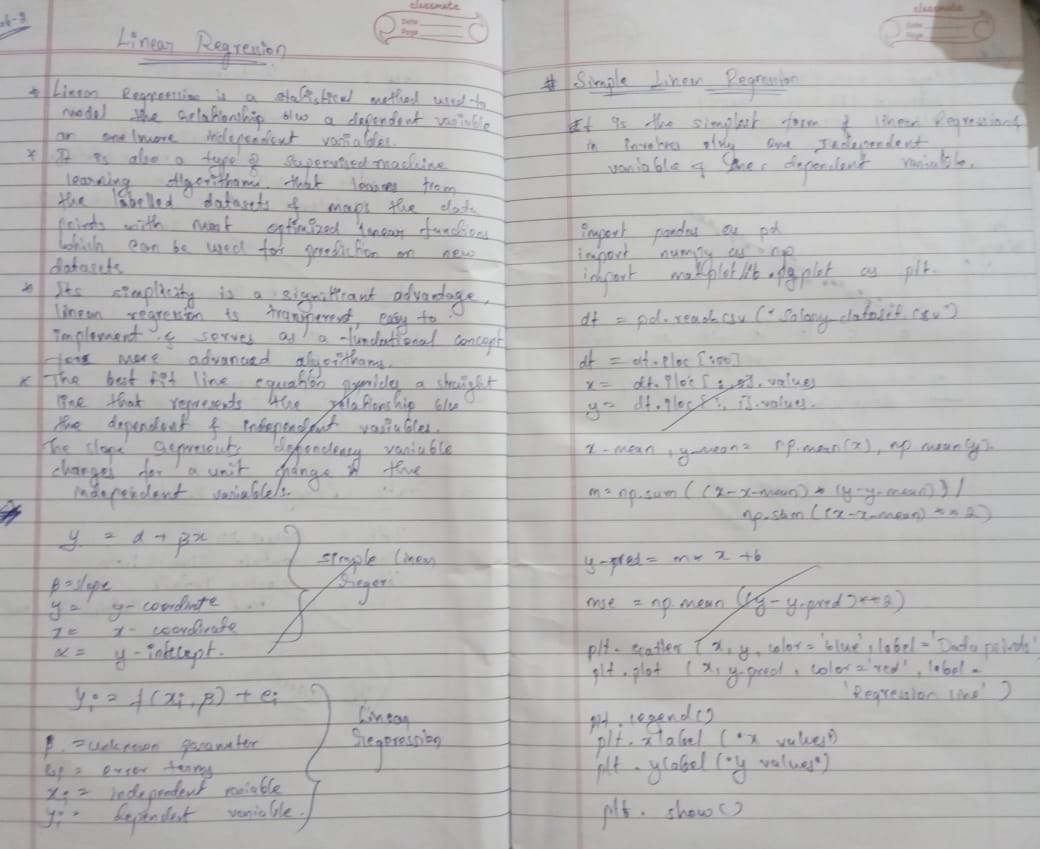
print("Decision Tree:")

print\_tree(decision\_tree)

##### Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



Code:

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

df = pd.read\_csv("/content/sample\_data/linear\_regression\_dataset.csv")

df1 = pd.read\_csv("/content/sample\_data/data\_for\_lr.csv")

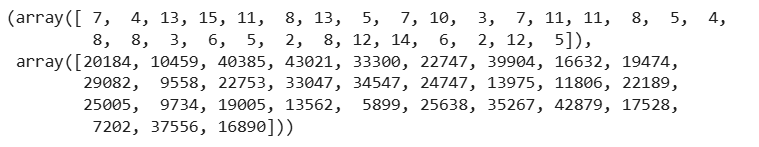
X = df[['Experience (Years)']].values.flatten()

y = df['Salary ($1000s)'].values

# X = df1['x'].values.flatten()

# y = df1['y'].values.flatten()

X, y



mean\_x = np.mean(X)

mean\_y = np.mean(y)

numerator = np.mean(X \* y) - mean\_x \* mean\_y

denominator = np.mean(X\*\*2) - mean\_x\*\*2

beta1 = numerator / denominator

beta0 = mean\_y - beta1 \* mean\_x

y\_pred = beta0 + beta1 \* X

error = np.sum((y - y\_pred) \*\* 2)

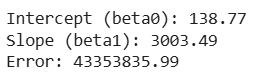
mean\_x, mean\_y



print(f"Intercept (beta0): {beta0:.2f}")

print(f"Slope (beta1): {beta1:.2f}")

print(f"Error: {error:.2f}")



plt.scatter(X, y, color='blue', label='Actual')

plt.plot(X, y\_pred, color='red', linewidth=2, label='Predicted')

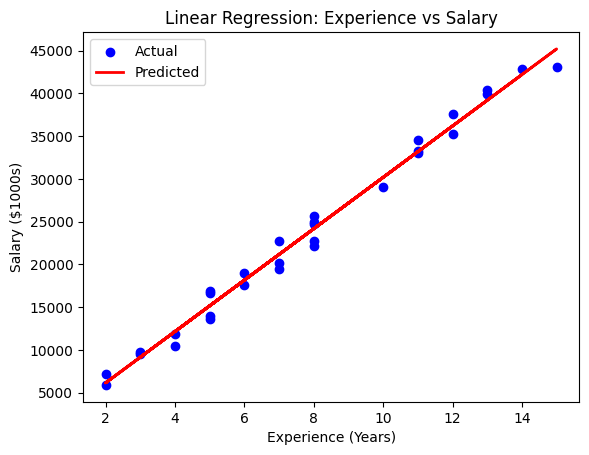
plt.xlabel('Experience (Years)')

plt.ylabel('Salary ($1000s)')

plt.title('Linear Regression: Experience vs Salary')

plt.legend()

plt.show()



result = f"Linear Equation: Y = {beta0:.2f} + {beta1:.2f}X + {error:.2f}"

result



##### Program 5

Build Logistic Regression Model for a given dataset

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

zoodata=files.upload()

zootype=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")

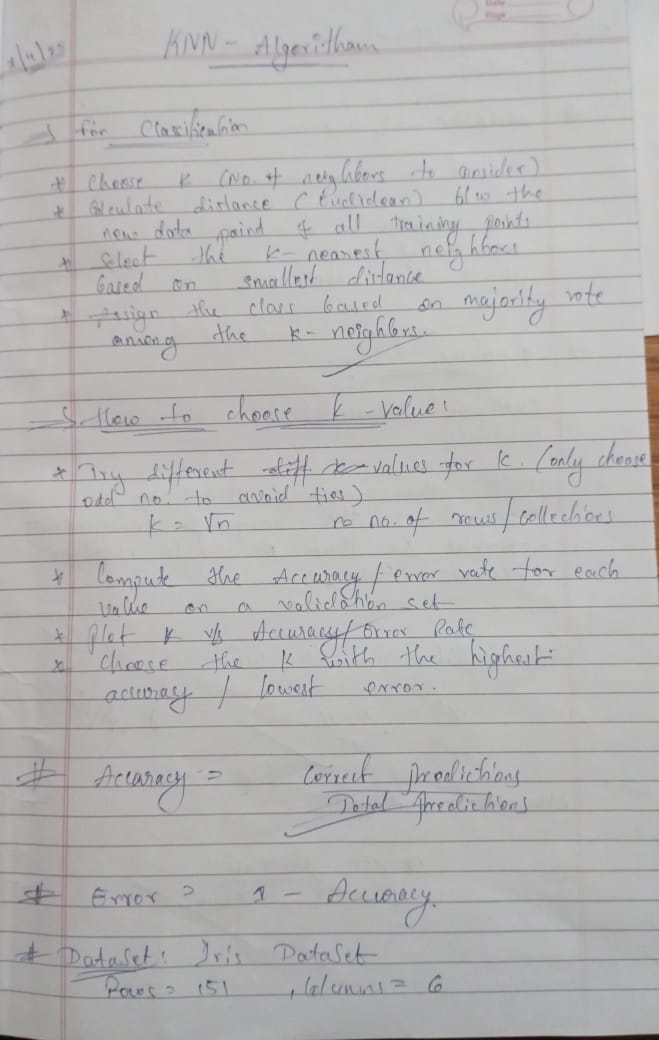
plt.title("Confusion Matrix")

plt.show()

##### Program 6

Build KNN Classification model for a given dataset.

Screenshot



Code:

import csv

import math

import random

from collections import Counter

# ---------- KNN Implementation ----------

def load\_dataset(filename):

    with open(filename, 'r') as f:

        data = list(csv.reader(f))

        header = data[0]

        rows = data[1:]

        for row in rows:

            for i in range(4):

                row[i] = float(row[i])

        return rows

def compute\_mean\_std(dataset):

    means, stds = [], []

    for i in range(4):

        col = [row[i] for row in dataset]

        mean = sum(col) / len(col)

        std = (sum((x - mean) \*\* 2 for x in col) / len(col)) \*\* 0.5

        means.append(mean)

        stds.append(std)

    return means, stds

def normalize\_dataset(dataset, means, stds):

    for row in dataset:

        for i in range(4):

            row[i] = (row[i] - means[i]) / stds[i]

    return dataset

def split\_dataset(dataset, test\_size=0.2):

    random.shuffle(dataset)

    split\_index = int(len(dataset) \* (1 - test\_size))

    return dataset[:split\_index], dataset[split\_index:]

def euclidean\_distance(row1, row2):

    return math.sqrt(sum((row1[i] - row2[i]) \*\* 2 for i in range(4)))

def knn\_predict(train, test\_row, k):

    distances = []

    for train\_row in train:

        dist = euclidean\_distance(test\_row, train\_row)

        distances.append((train\_row, dist))

    distances.sort(key=lambda x: x[1])

    neighbors = distances[:k]

    labels = [neighbor[0][4] for neighbor in neighbors]

    prediction = Counter(labels).most\_common(1)[0][0]

    return prediction

def evaluate\_model(train, test, k):

    correct = 0

    predictions = []

    for row in test:

        prediction = knn\_predict(train, row, k)

        predictions.append(prediction)

        if prediction == row[4]:

            correct += 1

    accuracy = correct / len(test)

    return accuracy, predictions

filename = '/content/sample\_data/IRIS.csv'

dataset = load\_dataset(filename)

means, stds = compute\_mean\_std(dataset)

dataset = normalize\_dataset(dataset, means, stds)

train\_data, test\_data = split\_dataset(dataset)

k = 3

accuracy, predictions = evaluate\_model(train\_data, test\_data, k)

print(f"Model Accuracy: {accuracy:.2f}\n\n")

print("\n--- Predict Iris Species from Your Input ---")

try:

    user\_input = []

    user\_input.append(float(input("Enter Sepal Length (cm): ")))

    user\_input.append(float(input("Enter Sepal Width (cm): ")))

    user\_input.append(float(input("Enter Petal Length (cm): ")))

    user\_input.append(float(input("Enter Petal Width (cm): ")))

    for i in range(4):

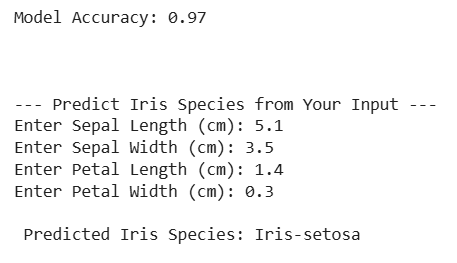
        user\_input[i] = (user\_input[i] - means[i]) / stds[i]

    predicted\_species = knn\_predict(train\_data, user\_input, k)

    print(f"\n Predicted Iris Species: {predicted\_species}")

except ValueError:

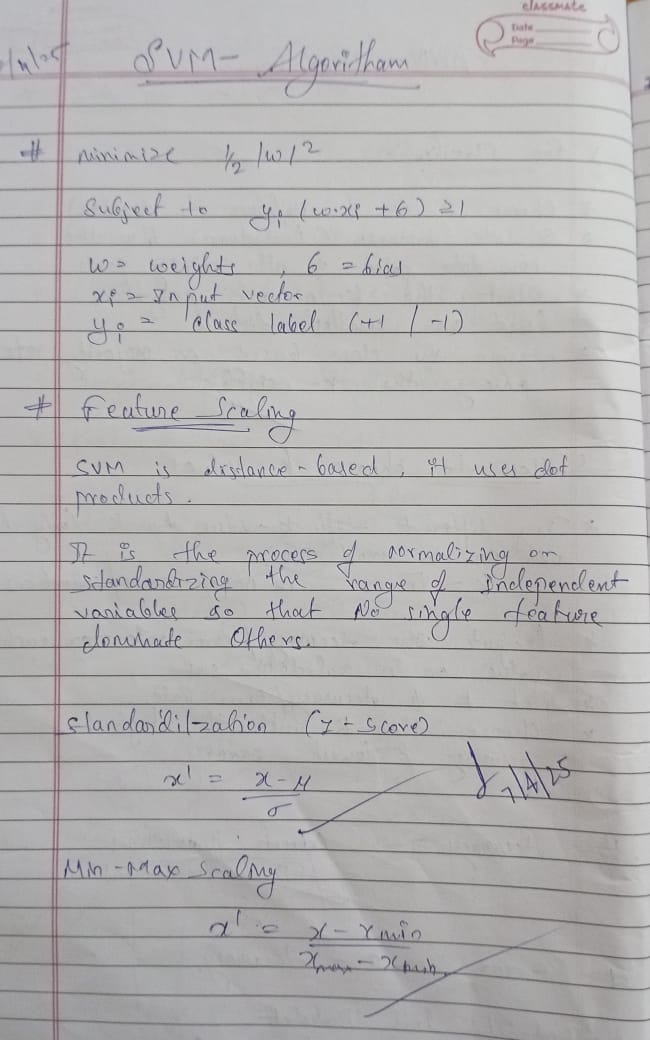
    print("Invalid input. Please enter numeric values for all measurements.")



##### Program 7

Build Support vector machine model for a given dataset

Screenshot



Code:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def PlotData(x,y):

    pos = np.argwhere(y == 1)

    neg = np.argwhere(y == 0)

    plt.plot(x[pos, 0], x[pos, 1], linestyle='', marker='+', color='k')

    plt.plot(x[neg, 0], x[neg, 1], linestyle='', marker='o', color='y')

    plt.xlabel('Exam 1 score')

    plt.ylabel('Exam 2 score')

    plt.legend(['Admitted', 'Not admitted'], loc='upper right', numpoints=1)

    plt.figure()

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from sklearn.svm import SVC

df = pd.read\_csv("/content/sample\_data/IRIS.csv")

X = df[["petal\_length", "petal\_width"]].values

y = df["species"].values

label\_map = {"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 2}

label\_map\_rev = {v: k for k, v in label\_map.items()}

y\_numeric = np.array([label\_map[label] for label in y])

svm\_model = SVC(kernel="linear")

svm\_model.fit(X, y\_numeric)

x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 300),

                     np.linspace(y\_min, y\_max, 300))

Z = svm\_model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

colors = ["red", "green", "blue"]

plt.figure(figsize=(8, 6))

plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)

for label in np.unique(y\_numeric):

    plt.scatter(X[y\_numeric == label, 0], X[y\_numeric == label, 1],

                label=label\_map\_rev[label], color=colors[label])

plt.xlabel("Petal Length (cm)")

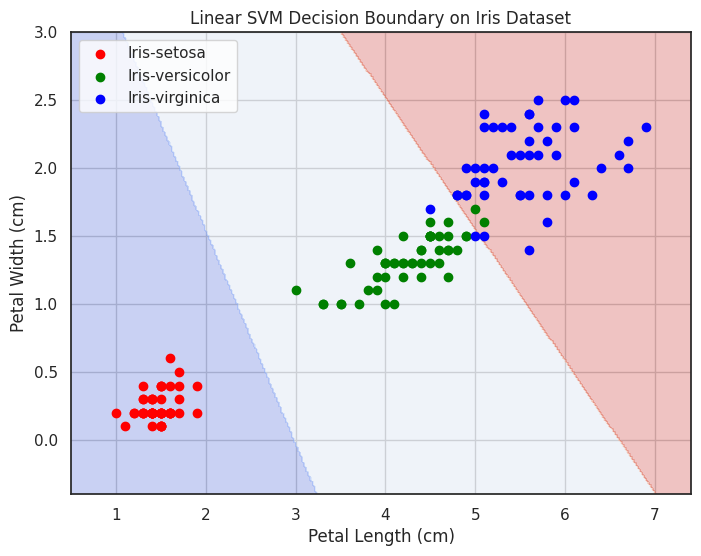
plt.ylabel("Petal Width (cm)")

plt.title("Linear SVM Decision Boundary on Iris Dataset")

plt.legend()

plt.grid(True)

plt.show()



import csv

import math

import random

from collections import Counter

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def load\_dataset(filename):

    df = pd.read\_csv("/content/sample\_data/IRIS.csv")

    X  = df[["petal\_length", "petal\_width"]].values

    y = df["species"].values

    label\_map = {"Iris-setosa": 0, "Iris-versicolor": 1, "Iris-virginica": 2}

    label\_map\_rev = {v: k for k, v in label\_map.items()}

    y\_numeric = np.array([label\_map[label] for label in y])

class SimpleLinearModel:

    def \_\_init\_\_(self, learning\_rate=0.01, n\_iters=1000):

        self.lr = learning\_rate

        self.n\_iters = n\_iters

        self.activation = self.\_unit\_step\_func

        self.weights = None

        self.bias = None

    def fit(self, X, y):

        n\_samples, n\_features = X.shape

        self.weights = np.zeros(n\_features)

        self.bias = 0

        y\_ = np.array([1 if i > 0 else 0 for i in y])

        for \_ in range(self.n\_iters):

            for idx, x\_i in enumerate(X):

                linear\_output = np.dot(x\_i, self.weights) + self.bias

                y\_predicted = self.activation(linear\_output)

                update = self.lr \* (y\_[idx] - y\_predicted)

                self.weights += update \* x\_i

                self.bias += update

    def predict(self, X):

        linear\_output = np.dot(X, self.weights) + self.bias

        y\_predicted = self.activation(linear\_output)

        return y\_predicted

    def \_unit\_step\_func(self, x):

        return np.where(x>=0, 1, 0)

svm\_model = SimpleLinearModel()

svm\_model.fit(X, y\_numeric)

x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 300),

                     np.linspace(y\_min, y\_max, 300))

Z = svm\_model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

colors = ["red", "green", "blue"]

plt.figure(figsize=(8, 6))

plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)

for label in np.unique(y\_numeric):

    plt.scatter(X[y\_numeric == label, 0], X[y\_numeric == label, 1],

                label=label\_map\_rev[label], color=colors[label])

plt.xlabel("Petal Length (cm)")

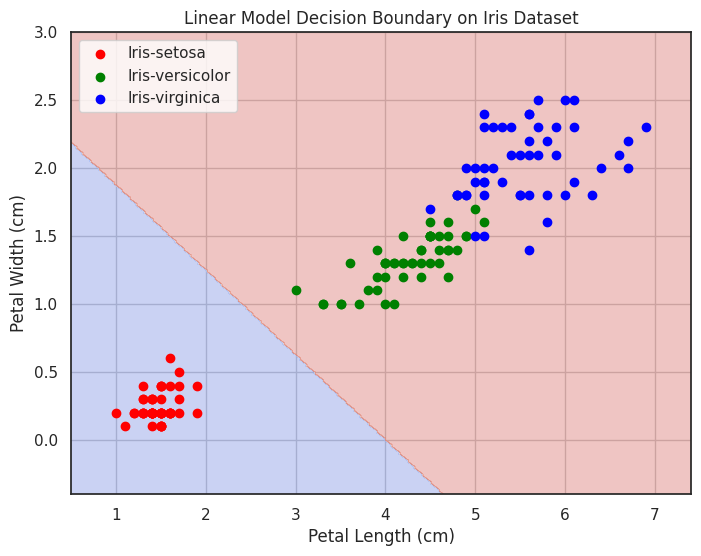
plt.ylabel("Petal Width (cm)")

plt.title("Linear Model Decision Boundary on Iris Dataset")

plt.legend()

plt.grid(True)

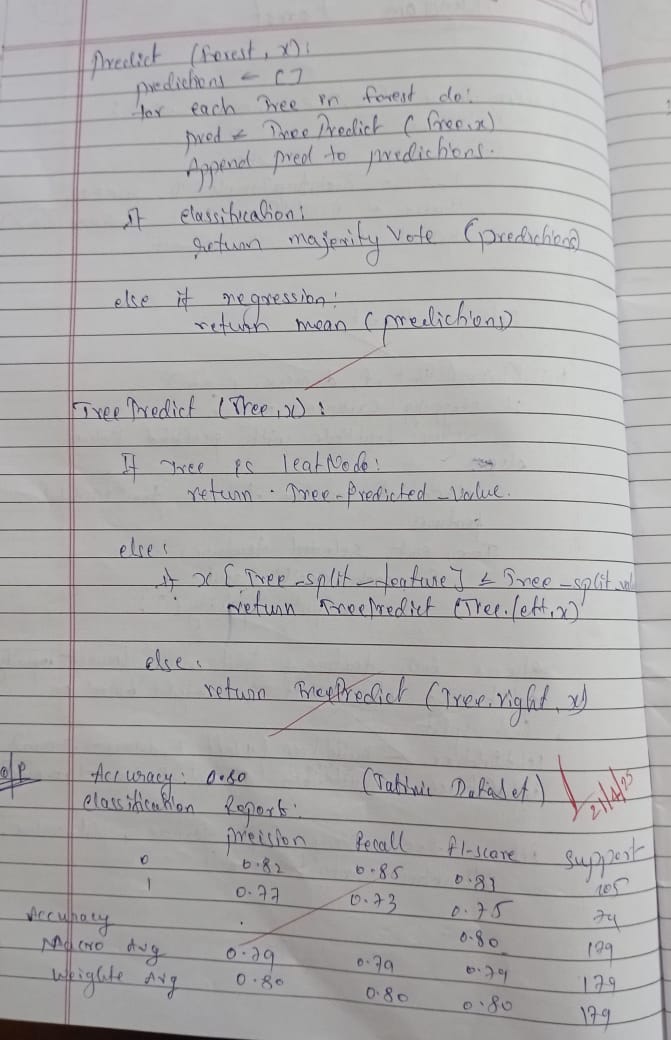
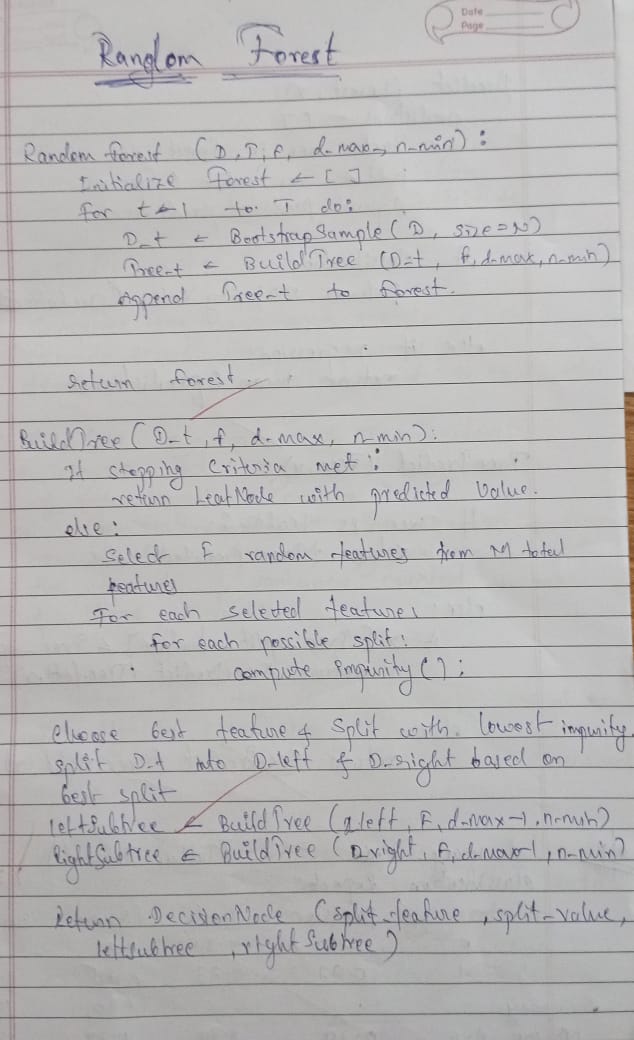
plt.show()



##### Program 8

Implement Random Forest ensemble method on a given dataset.

Screenshot



Code:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset

file\_path = "/content/Pumpkin\_Seeds\_Dataset.xlsx"

df = pd.read\_excel(file\_path, sheet\_name='Pumpkin\_Seeds\_Dataset')

# Separate features and target

X = df.drop(columns=['Class'])

y = df['Class']

# Encode the target labels

le = LabelEncoder()

y\_encoded = le.fit\_transform(y)

# Split into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y\_encoded, test\_size=0.2, random\_state=42, stratify=y\_encoded

)

# Train the Random Forest classifier

rf\_model = RandomForestClassifier(n\_estimators=100, max\_depth=None, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predict on test data

y\_pred = rf\_model.predict(X\_test)

# Evaluate the model

print("✅ Accuracy:", accuracy\_score(y\_test, y\_pred))

print("\n📊 Classification Report:\n")

print(classification\_report(y\_test, y\_pred, target\_names=le.classes\_))

# Plot the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=le.classes\_, yticklabels=le.classes\_)

plt.title("Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.tight\_layout()

plt.show()

# Plot top 10 feature importances

importances = rf\_model.feature\_importances\_

features = X.columns

indices = importances.argsort()[::-1][:10]  # Top 10

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

sns.barplot(x=importances[indices], y=features[indices], palette="viridis")

plt.title("Top 10 Feature Importances")

plt.xlabel("Importance")

plt.ylabel("Feature")

plt.tight\_layout()

plt.show()

##### Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot

Code:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.tree import DecisionTreeRegressor

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier as KNN

from sklearn.metrics import accuracy\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import mean\_squared\_error as MSE

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingRegressor

SEED =1

# Dataset

liver = pd.read\_csv('/content/sample\_data/indian\_liver\_patient\_preprocessed.csv', index\_col = 0)

X = liver.drop('Liver\_disease', axis = 1)

y = liver['Liver\_disease']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=SEED)

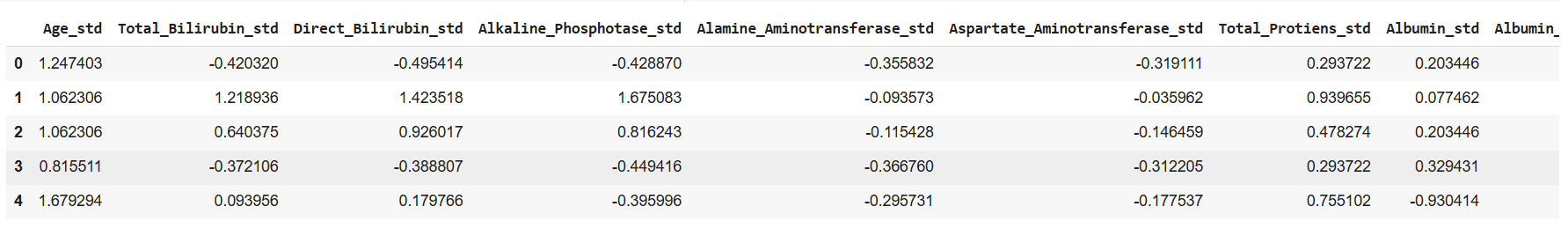
# Split data into 80% train and 20% test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

stratify=y,

random\_state=SEED)

liver.head()



# Import AdaBoostClassifier

from sklearn.ensemble import AdaBoostClassifier

# Instantiate dt

dt = DecisionTreeClassifier(max\_depth=2, random\_state=1)

# Instantiate ada

ada = AdaBoostClassifier(estimator=dt, n\_estimators=180, random\_state=1)

# Fit ada to the training set

ada.fit(X\_train, y\_train)

# Compute the probabilities of obtaining the positive class

y\_pred\_proba = ada.predict\_proba(X\_test)[:,1]

# Import roc\_auc\_score

#from sklearn.metrics import roc\_auc\_score

# Evaluate test-set roc\_auc\_score

ada\_roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

# Print roc\_auc\_score

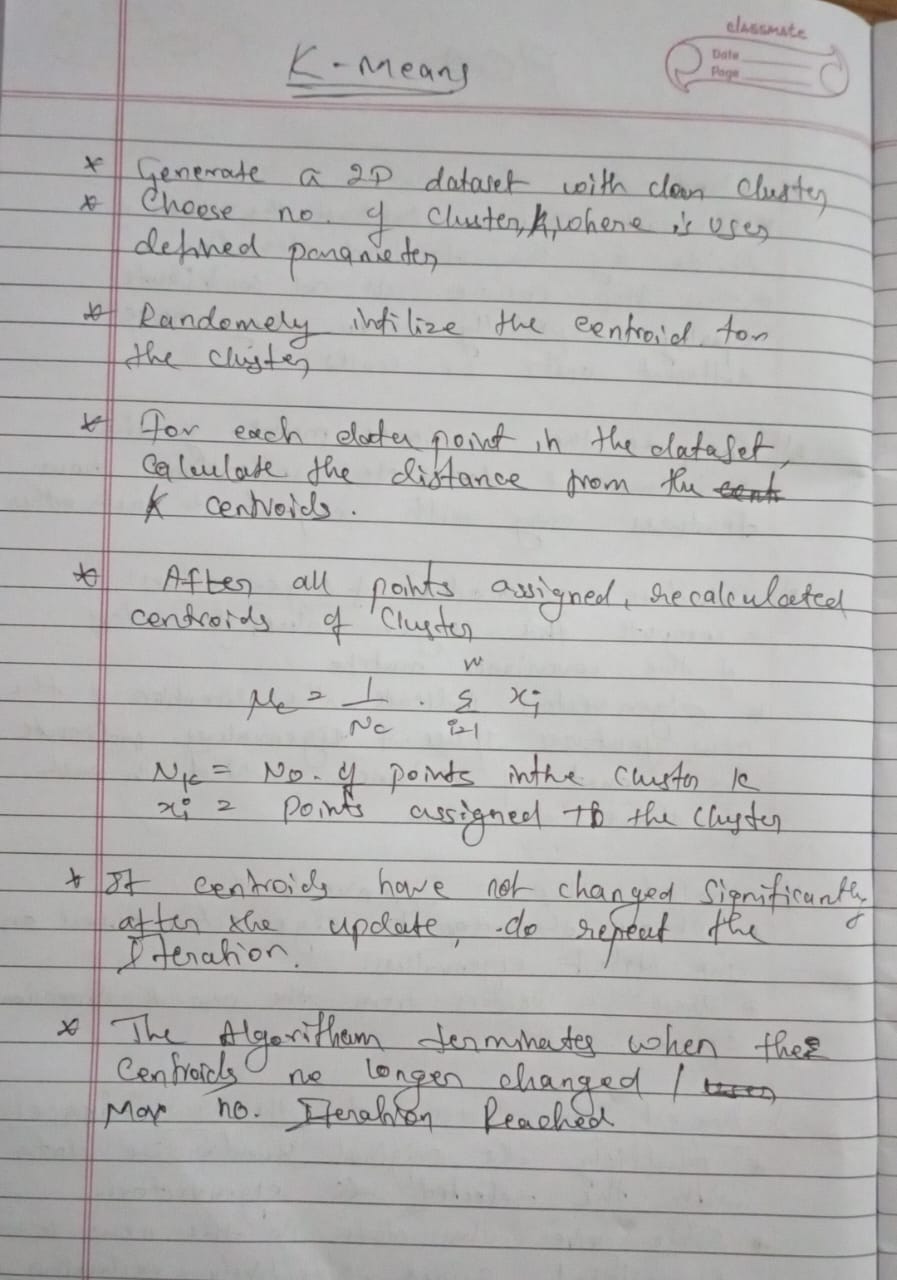
print('ROC AUC score: {:.2f}'.format(ada\_roc\_auc))



##### Program 10

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

Screenshot



Code:

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt # for data visualization

import seaborn as sns # for statistical data visualization

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv('/content/sample\_data/Live.csv')

df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=2, random\_state=0)

kmeans.fit(X)

kmeans.cluster\_centers\_

kmeans.inertia\_

labels = kmeans.labels\_

# check how many of the samples were correctly labeled

correct\_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))



print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))



from sklearn.cluster import KMeans

cs = []

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(X)

    cs.append(kmeans.inertia\_)

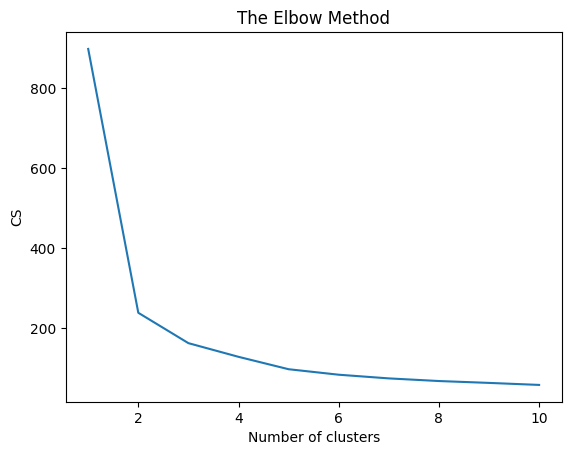
plt.plot(range(1, 11), cs)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('CS')

plt.show()



from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=2,random\_state=0)

kmeans.fit(X)

labels = kmeans.labels\_

# check how many of the samples were correctly labeled

correct\_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))

print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))



kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled

labels = kmeans.labels\_

correct\_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))

print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))



kmeans = KMeans(n\_clusters=4, random\_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled

labels = kmeans.labels\_

correct\_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))

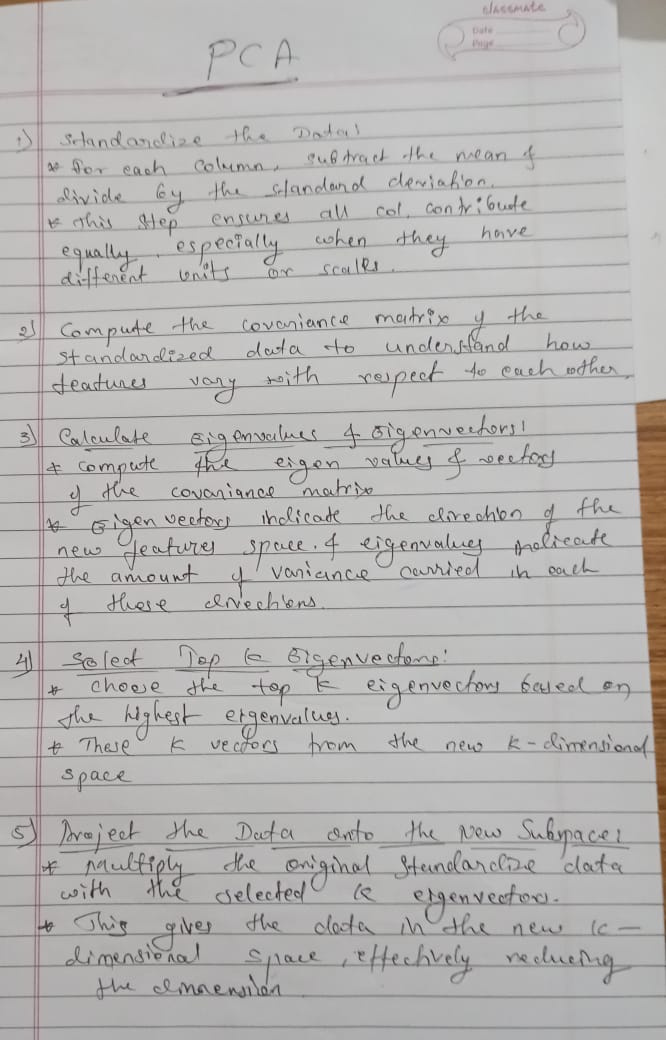
print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))



##### Program 11

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

Screenshot



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]}")