VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Dheemanth G Athreya (1BM22CS347)

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 **Dheemanth G Athreya (1BM22CS347),** who is a 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. Seema Patil Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

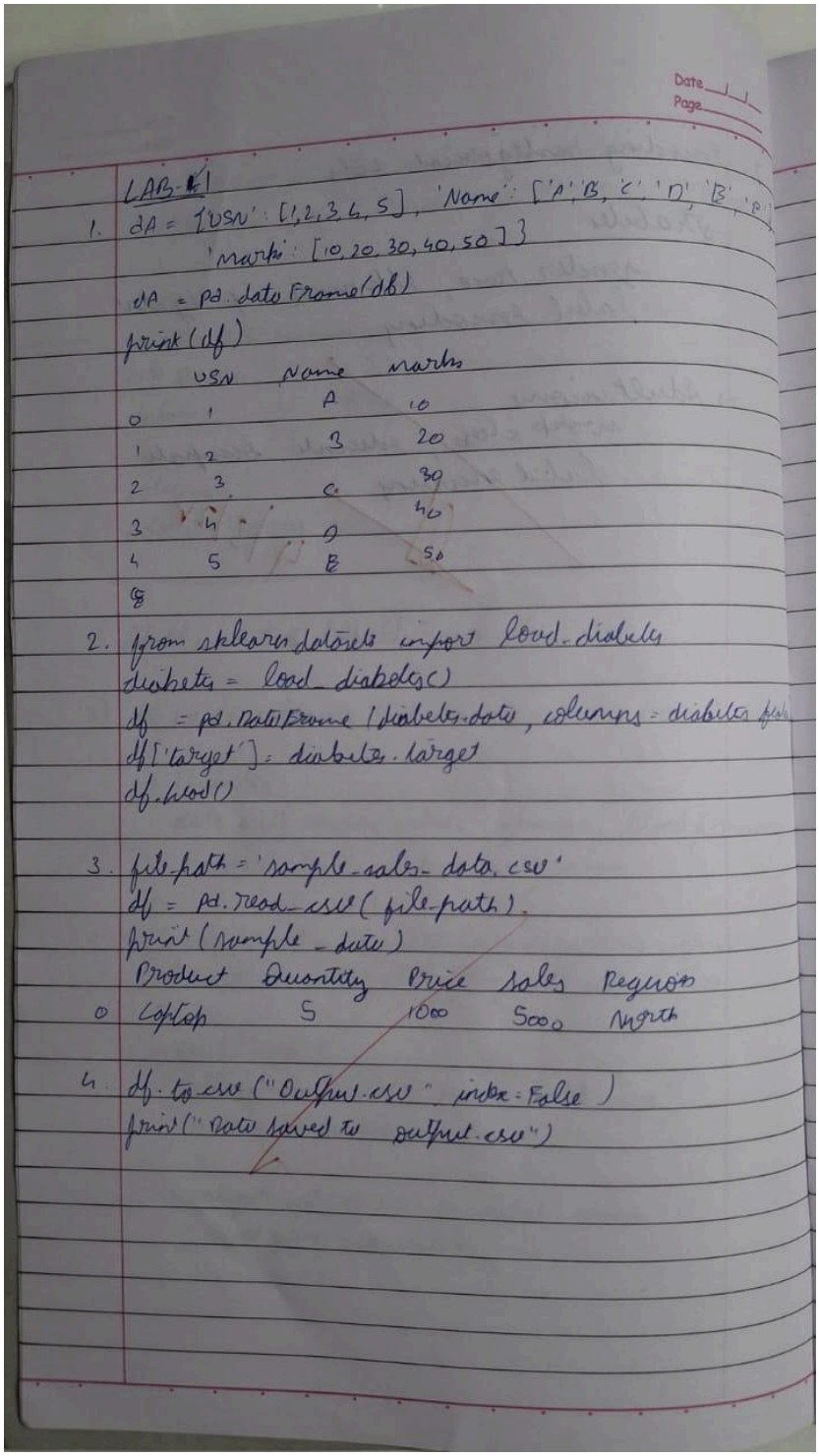
Index

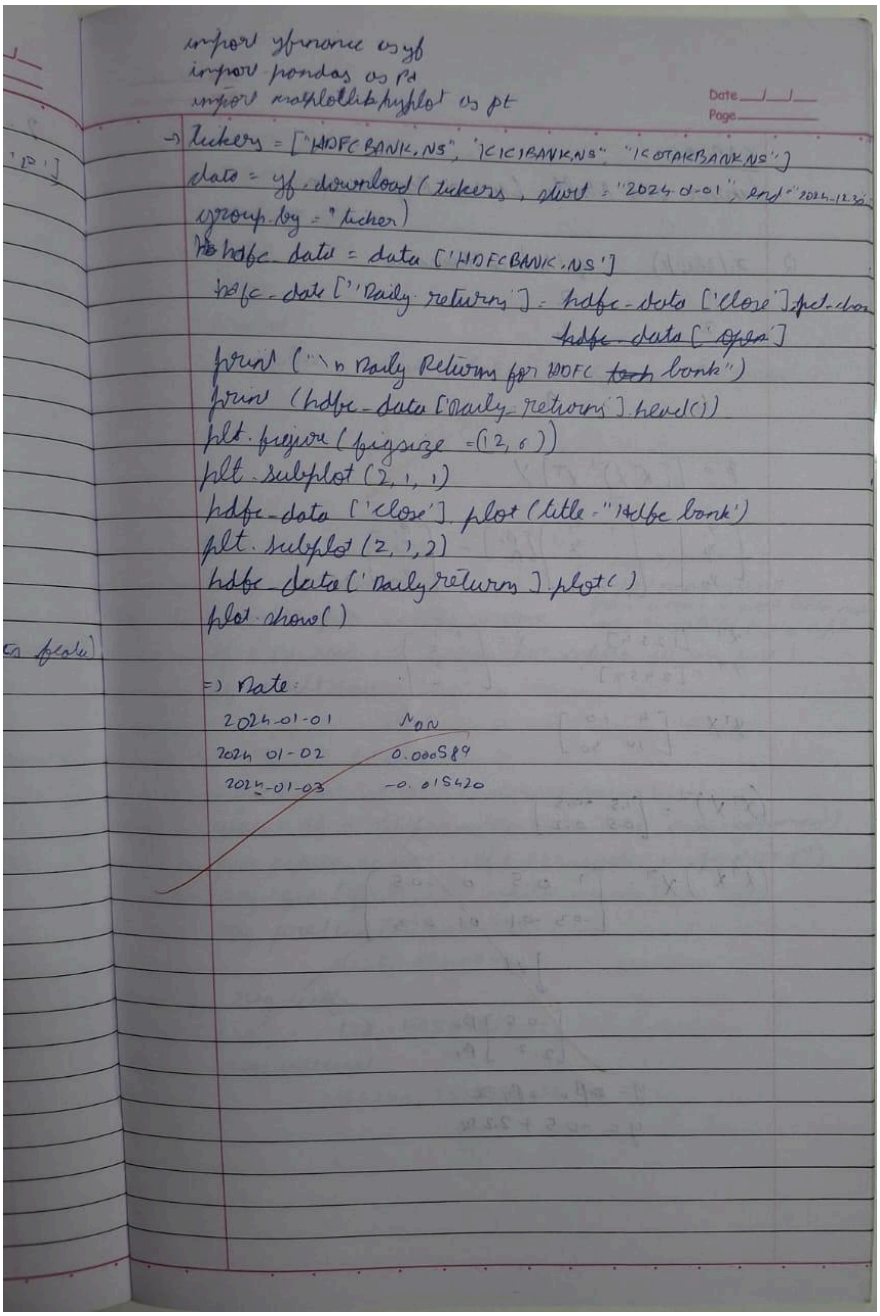
Github Link: https://github.com/dheemanthAthreya/6thSem-ML-Lab

Program 1

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

Screenshot:





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

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "ICICIBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-31",
group_by="ticker")
hdfc_data = data['HDFCBANK.NS']
hdfc_data['Daily returns'] = hdfc_data['Close'].pct_change()
hdfc_data['Open']
print("\n Daily Returns for HDFC bank")
print(hdfc_data['Daily returns'].head())
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
hdfc_data['Close'].plot(title="Hdfc bank")
plt.subplot(2,1,2)
hdfc_data['Daily returns'].plot()
plt.show()

=) Date:

2024-01-01	NaN
2024-01-02	0.000589
2024-01-03	-0.015420

Code:

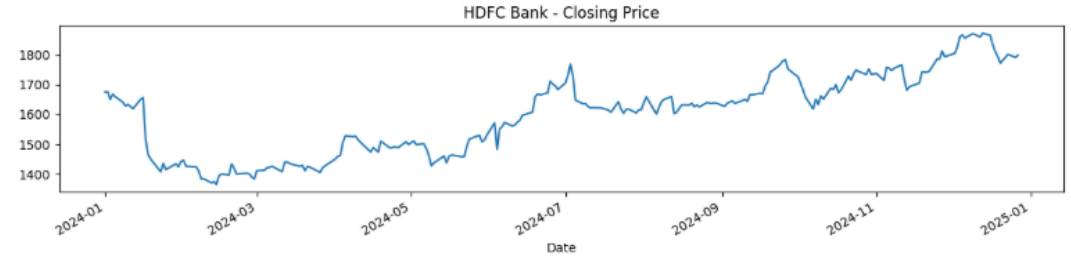
```
import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')

print(data.head())

hdfc_data = data['HDFCBANK.NS']

print("\nSummary statistics for HDFC Bank:")

print(hdfc_data.describe())

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")

plt.subplot(2, 1, 2)

hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns",
color='orange')

plt.tight_layout()

plt.show()
```

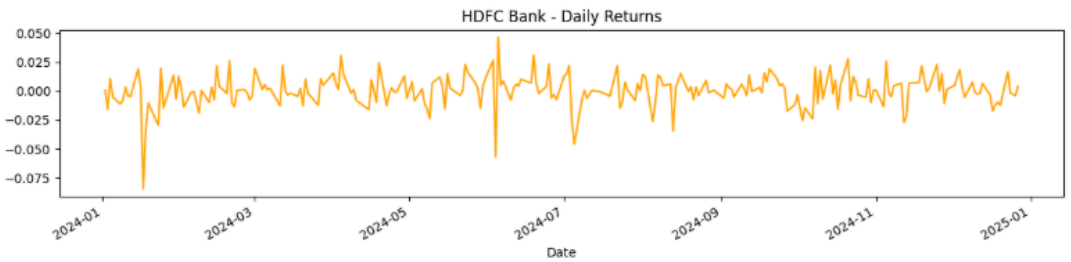
Output:

```
[**********************100%**********************]  3 of 3 completed
Ticker     ICICIBANK.NS                                                  \
Price              Open        High         Low       Close    Volume
Date
2024-01-01   983.086778  996.273246  982.541485  990.869812   7683792
2024-01-02   988.490253  989.134730  971.883221  973.866150  16263825
2024-01-03   976.295294  979.567116  966.777197  975.650818  16826752
2024-01-04   977.980767  980.707295  973.519176  978.724365  22789140
2024-01-05   979.567084  989.779158  975.402920  985.218445  14875499

Ticker     KOTAKBANK.NS                                                    \
Price              Open         High          Low        Close   Volume
Date
2024-01-01  1906.909954  1916.899006  1891.027338  1907.059814  1425902
2024-01-02  1905.911108  1905.911108  1858.063525  1863.008179  5120796
2024-01-03  1861.959234  1867.952665  1845.627158  1863.857178  3781515
2024-01-04  1869.451068  1869.451068  1858.513105  1861.559692  2865766
2024-01-05  1863.457575  1867.852782  1839.383985  1845.577148  7799341

Ticker      HDFCBANK.NS
Price              Open         High          Low        Close    Volume
Date
2024-01-01  1683.017598  1686.125187  1669.206199  1675.223999   7119843
2024-01-02  1675.914685  1679.860799  1665.950651  1676.210571  14621046
2024-01-03  1679.071480  1681.735059  1646.466666  1650.363525  14194881
2024-01-04  1655.394910  1672.116520  1648.193203  1668.071777  13367028
2024-01-05  1664.421596  1681.932477  1645.628180  1659.538208  15944735
```

```
Summary statistics for HDFC Bank:
Price         Open         High          Low        Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1601.375295  1615.443664  1588.221245  1601.898968  2.119658e+07
std     134.648125   134.183203   132.796819   133.748372  2.133860e+07
min    1357.463183  1372.754374  1345.180951  1365.404785  8.798460e+05
25%    1475.316358  1494.072805  1460.259509  1474.564087  1.274850e+07
50%    1627.724976  1638.350037  1616.000000  1625.950012  1.686810e+07
75%    1696.474976  1711.425018  1679.250000  1697.062531  2.295014e+07
max    1877.699951  1880.000000  1858.550049  1871.750000  2.226710e+08
```

```
<ipython-input-11-044396f9f2dc>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy
  hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
```

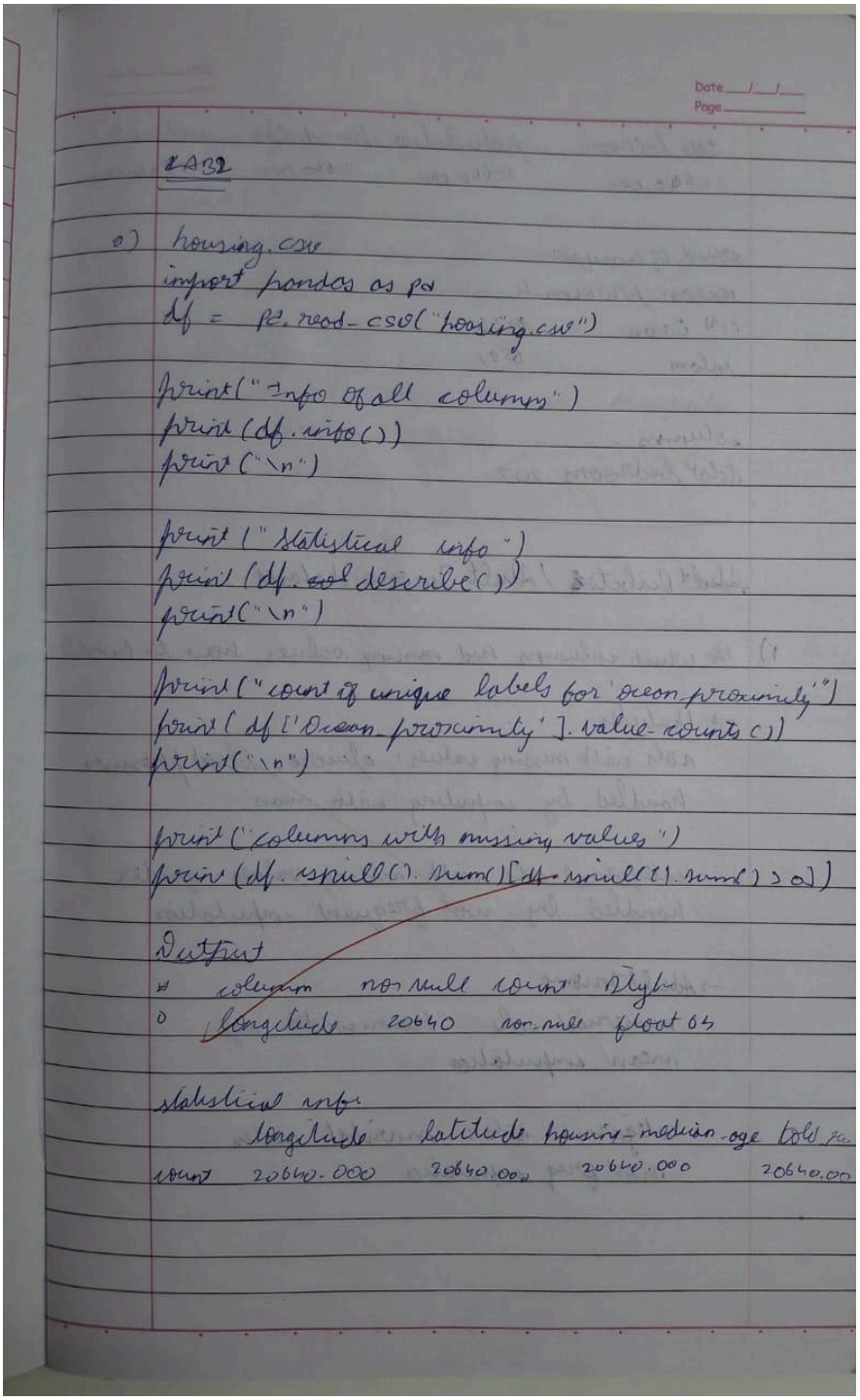




Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



LAB2

a) housing.csv

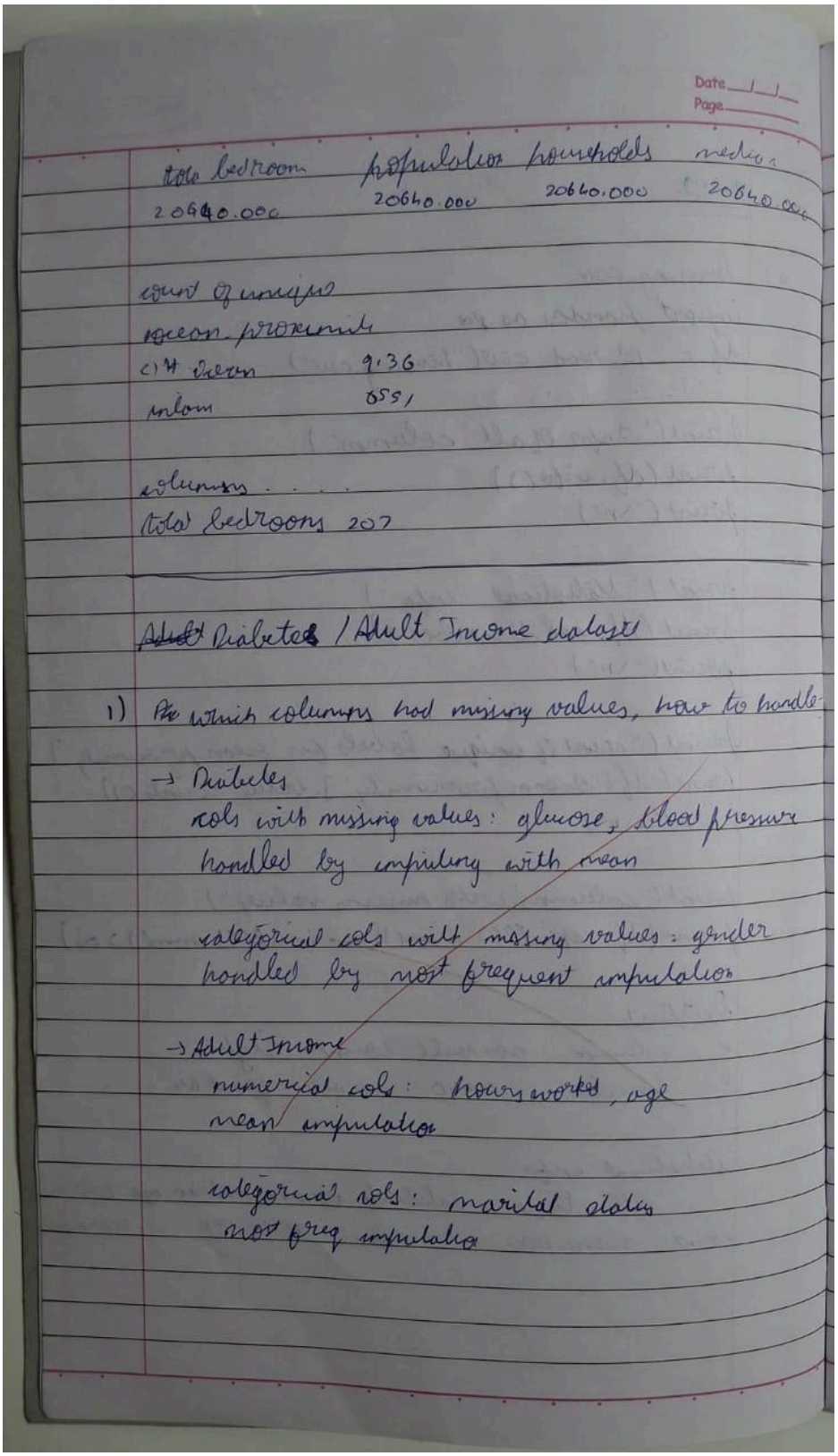
```
import pandas as pd
df = pd.read_csv("housing.csv")

print("Info of all columns")
print(df.info())
print("\n")

print("Statistical info")
print(df.describe())
print("\n")

print("count of unique labels for 'ocean_proximity'")
print(df['Ocean_proximity'].value_counts())
print("\n")

print("columns with missing values")
print(df.isnull().sum()[df.isnull().sum() > 0])
```

Output

#	column	non-null count	dtype
0	longitude	20640	non-null float 64

statistical info

	longitude	latitude	housing_median_age	total_ro
count	20640.000	20640.000	20640.000	20640.00


Date / /
Page

total bedroom	population	households	median
20640.000	20640.000	20640.000	20640.000

count of unique
ocean_proximity
<1H ocean 9136
inland 6551

columns . . . .
total bedrooms 207

~~Adult~~ Diabetes / Adult Income dataset

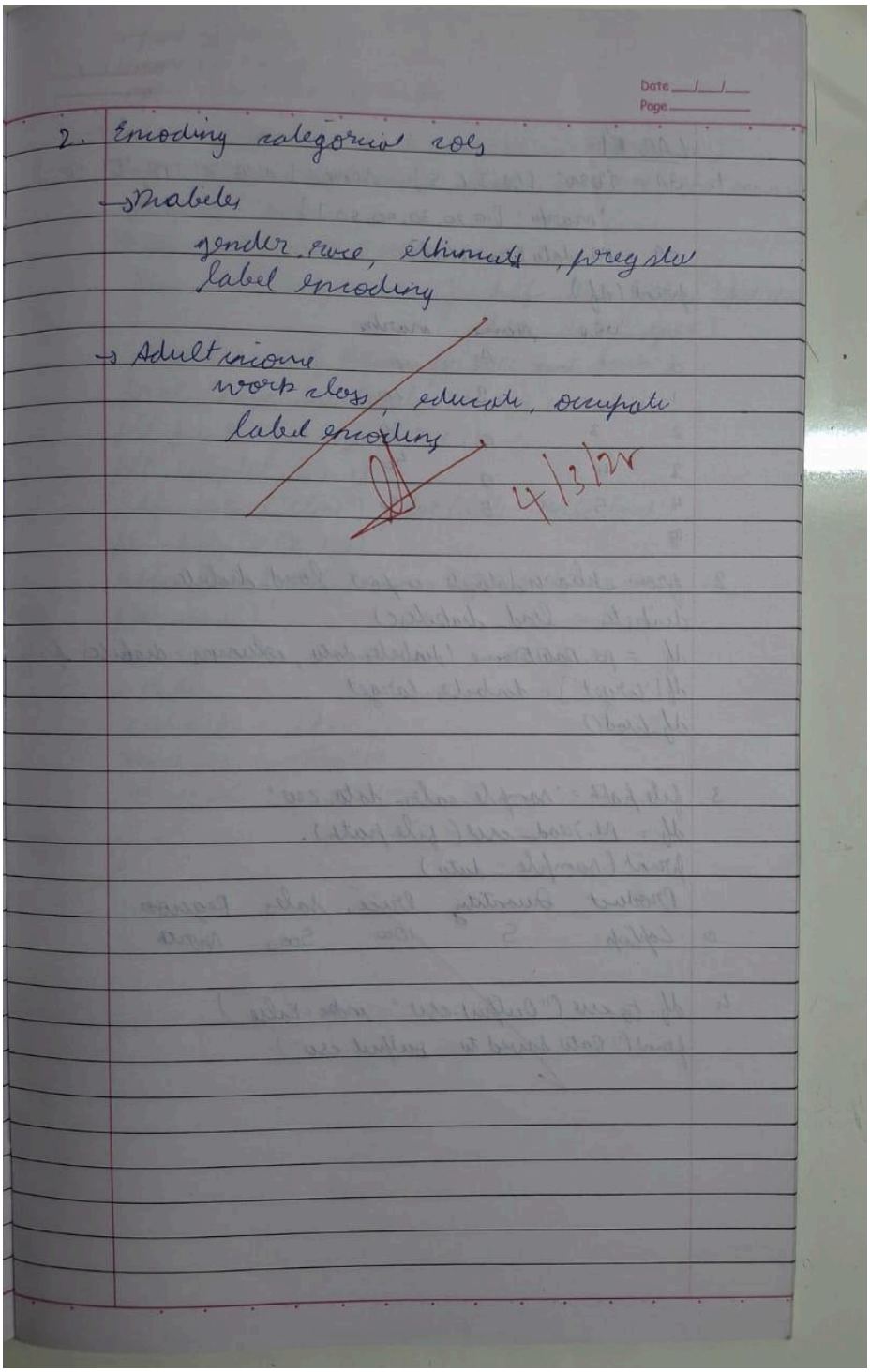
1) Which columns had missing values, how to handle

→ Diabetes
cols with missing values: glucose, blood pressure
handled by imputing with mean

categorical cols with missing values: gender
handled by most frequent imputation

→ Adult Income
numerical cols: hours worked, age
mean imputation

categorical cols: marital status
most freq imputation


Date __/__/__
Page _____

2. Encoding categorical cols

→ Models

gender, race, ethnicity, pregstw
label encoding

→ Adult income

workclass, education, occupation
label encoding

4/3/22

Code:

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

```
# i. Load .csv file into the DataFrame
df = pd.read_csv("housing.csv")

# ii. Display information of all columns
print("Information of all columns:")
print(df.info())
print("\n")

# iii. Display statistical information of all numerical columns
print("Statistical information of all numerical columns:")
print(df.describe())
print("\n")

# iv. Display the count of unique labels for the 'Ocean Proximity' column
print("Count of unique labels for 'Ocean Proximity' column:")
print(df['ocean_proximity'].value_counts())
print("\n")

# v. Display which attributes (columns) have missing values count greater than
zero
print("Columns with missing values count greater than zero:")
print(df.isnull().sum()[df.isnull().sum() > 0])
```

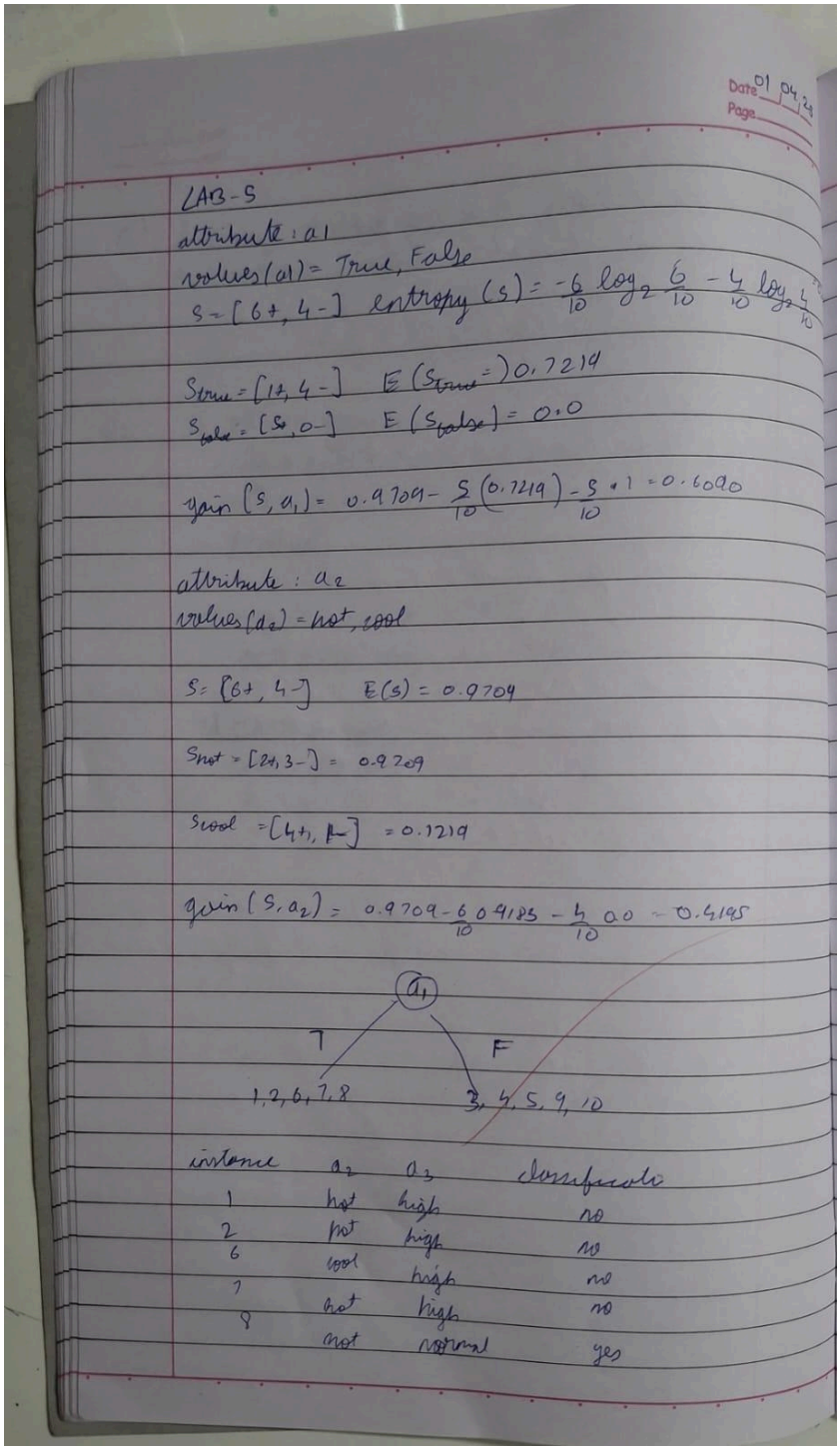
Output:

```
Information of all columns:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  int64
 3   total_rooms         20640 non-null  int64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64
 6   households          20640 non-null  int64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  int64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
None


Statistical information of all numerical columns:
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000


Count of unique labels for 'Ocean Proximity' column:
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64


Columns with missing values count greater than zero:
total_bedrooms    207
dtype: int64
```

Program 3

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

Screenshot:



Date / /
Page

attribute: a2

$S_{a_1} = [1+, 4-] = E(a_1) = 0.7219$

$S_{hot} = [1+, 3-] = 0.8112$

$S_{cool} = [0+, 1-]$ $E = 0.0$

gain $(S, a_2) = 0.3219$

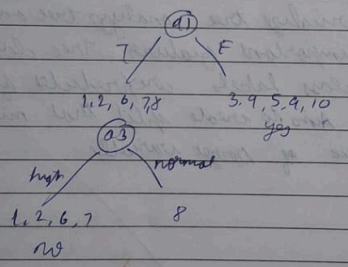
attribute : a3

$S_{a_1} = 0.7219$

$S_{high} = 0$

$S_{normal} = 0$

gain $(S, a_3) = 0.7219$



code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
df = pd.read_csv('petrol.csv')
x = df.drop('Petrol', axis=1)
y = df['Petrol']
X_train, X_test, y_train, y_test = train_test_split
(x, y, test_size=0.2, random_state=42)
```

Date
Page

```
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mae = mean_absolute_error()
mse = mean_squared_error()
rmse = mse ** 0.5
```

output:

mae : 96.2

mse : 17858.2

rmse : 133.63

(1) Accuracy score for iris dataset is 1.0
conf mat shows no misclassification

(2) need to visualize tree, analyze tree and identify important features, Tree classifies discrete class labels we predicted for each instance. Aims to create split that minimizes the variance of target variable

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

# Split data into features (X) and target (y)
X = df.drop('Petrol_Consumption', axis=1)
y = df['Petrol_Consumption']

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

# Initialize the DecisionTreeRegressor
regressor = DecisionTreeRegressor()

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

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

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

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

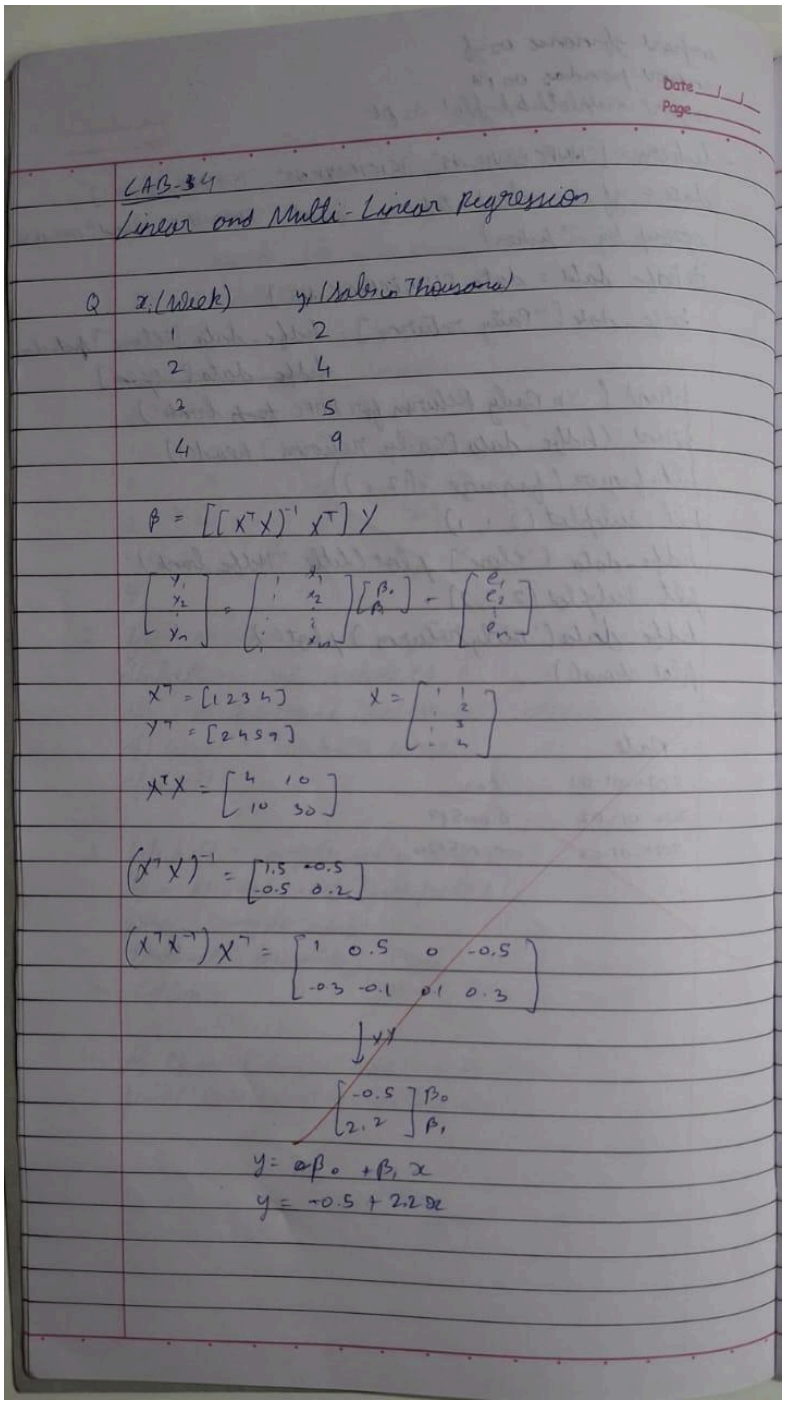
Output:

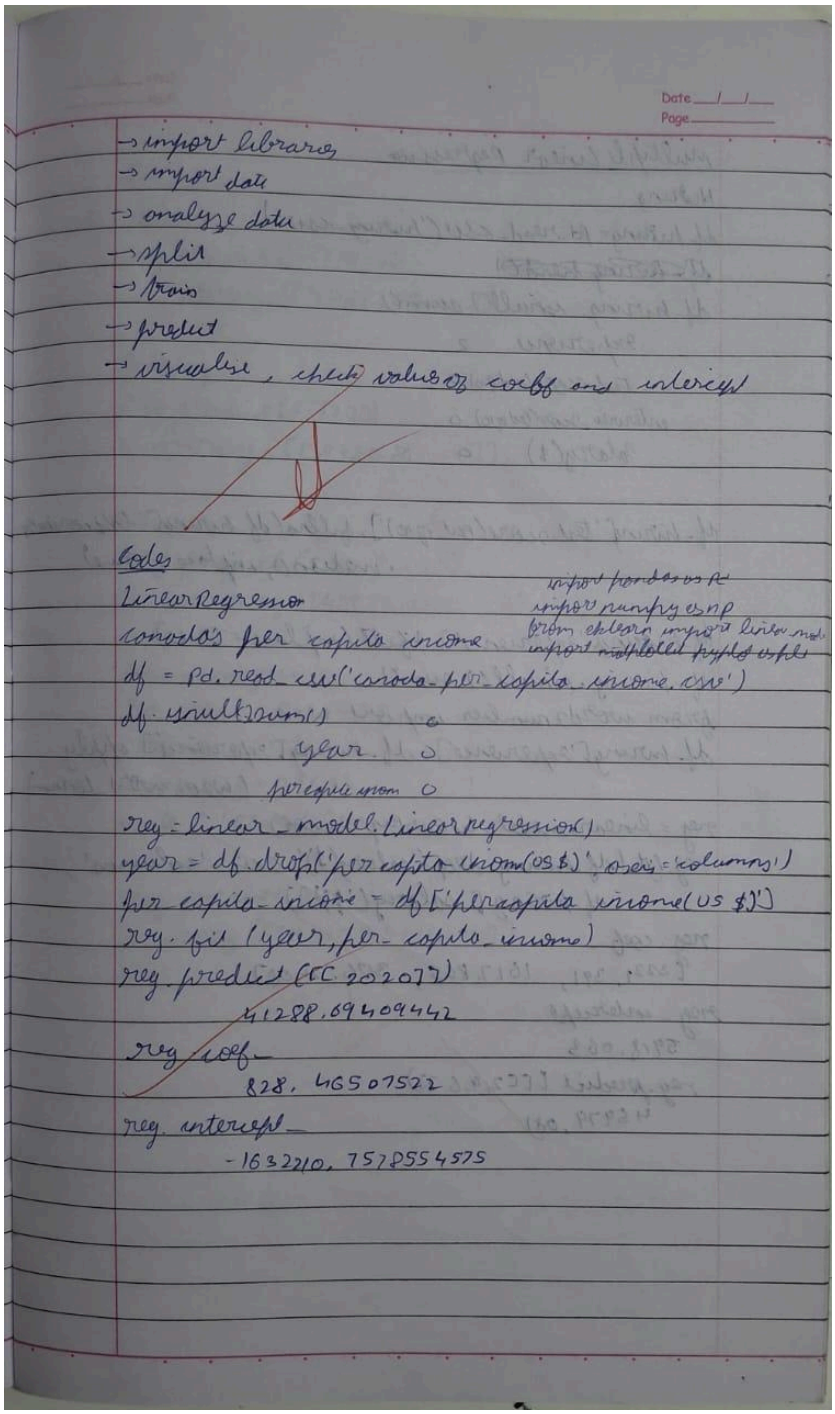
Mean Absolute Error: 90.6
Mean Squared Error: 16267.8
Root Mean Squared Error: 127.54528607518193

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

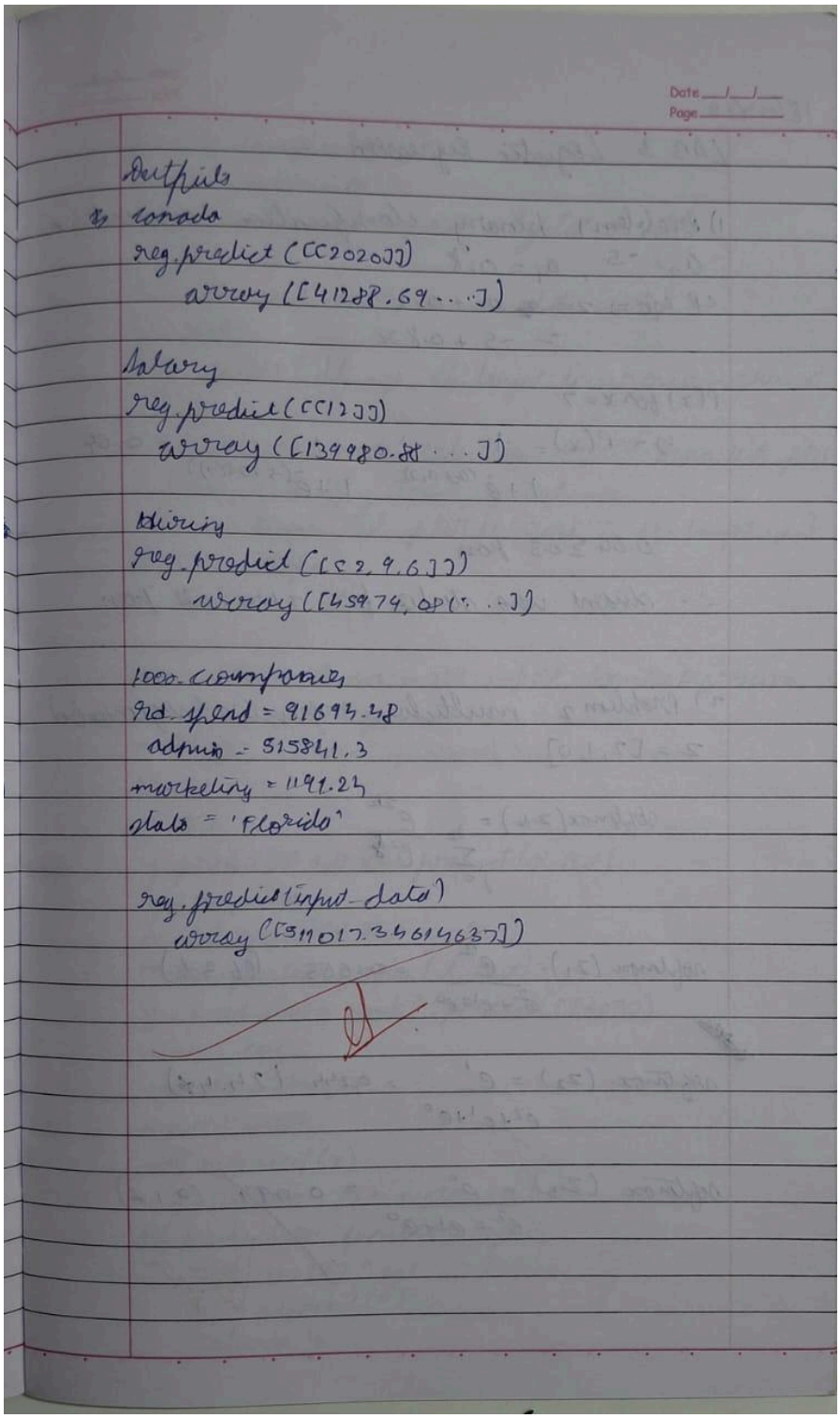



Date
Page
→ import libraries
→ import data
→ analyse data
→ split
→ train
→ predict
→ visualise, check values of coeff and intercept
Codes
LinearRegression
canada's per capita income
df = pd.read_csv('canada_per_capita_income.csv')
df.isnull().sum()
year 0
per capita income (US $) 0
reg = linear_model.LinearRegression()
year = df.drop('per capita income(US $)', axis='columns')
per_capita_income = df['per capita income(US $)']
reg.fit(year, per_capita_income)
reg.predict([[2020]])
41288.69409442
reg.coef_
828.46507522
reg.intercept_
-1632210.7578554575

Multiple Linear Regression

Hiring

```
df_hiring = pd.read_csv('hiring.csv')
df_hiring.head()
df_hiring.isnull().sum()
    experience      2
    test_score(out of 10)   0
    interview_score(out of 10)  0
    salary($)       0

df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of 10)']
                .median(), inplace = True)

df_hiring['experience'].bfill(inplace = True)
df_hiring.isnull().sum()
from word2number import w2n
df_hiring['experience'] = df_hiring['experience'].apply
                (w2n.word_to_num)

reg = linear_model.LinearRegression()
reg.fit(df_hiring.drop('salary($)', axis = 'columns'),
        df_hiring['salary($)'])
reg.coef_
  [8221.341, 1017.865, 3176.240]
reg.intercept_
  5978.063
reg.predict([[2,9,6]])
  45979.08
```

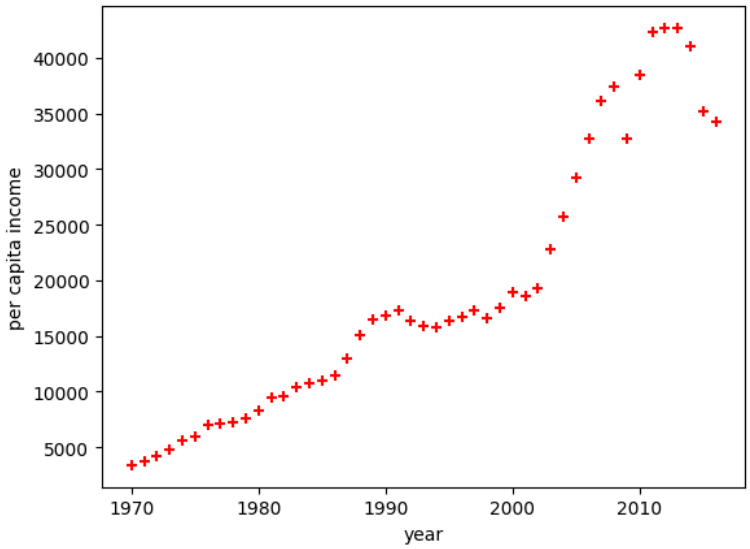

Date
Page
Output:
Canada
reg.predict([[2020]])
array([41288.69...])
Salary
reg.predict([[12]])
array([139480.88...])
Hiring
reg.predict([[2,9,6]])
array([45974.08...])
1000 Companies
rd_spend = 91694.48
admin = 515841.3
marketing = 1191.24
state = 'Florida'
reg.predict(input_data)
array([[53017.34614637]])

Code:

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

df = pd.read_csv('canada_per_capita_income.csv')
plt.xlabel('year')
plt.ylabel('per capita income')
plt.scatter(df['year'],df['per capita income (US$)'
],color='red',marker='+')
reg = linear_model.LinearRegression()
year = df.drop('per capita income (US$)',axis='columns')
per_capita_income = df['per capita income (US$)' ]
reg.fit(year, per_capita_income)
reg.predict([[2020]])
reg.coef_
reg.intercept_


df_hiring = pd.read_csv('hiring.csv')
df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of
10)'].median(),inplace=True)
#forward fill experience
df_hiring['experience'].bfill(inplace=True)
df_hiring.isnull().sum()
#convert experiece to numerical
from word2number import w2n
df_hiring['experience'] = df_hiring['experience'].apply(w2n.word_to_num)
reg = linear_model.LinearRegression()
reg.fit(df_hiring.drop('salary($)',axis='columns'),df_hiring['salary($)'])
reg.coef_
reg.intercept_
reg.predict([[2, 9, 6]])
reg.predict([[12, 10, 10]])
```

Output:

```
Out[6]: <matplotlib.collections.PathCollection at 0x7d54e02fd250>
```



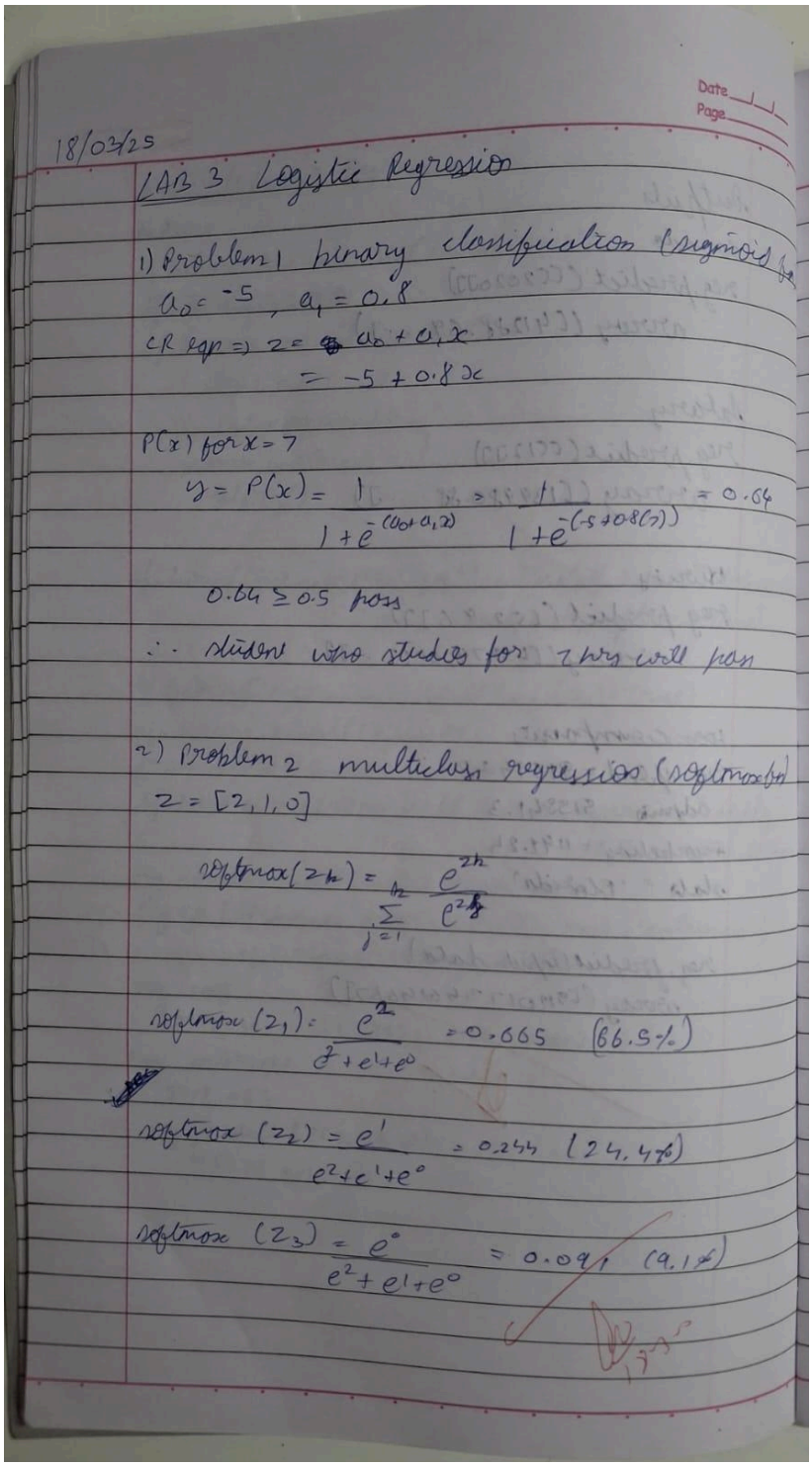
array([41288.69409442])
array([828.46507522])
-1632210.7578554575

array([3221.39134934, 1617.86554643, 3176.24086827])
5918.063888238692
array([45979.08171436])
array([92515.82422727])

Program 5

Build Logistic Regression Model for a given dataset

Screenshot:



Date __/__/____
Page _____

Binary classification

```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("insurance_data.csv")
df.head()
plt.scatter(df.age, df.bought_insurance, marker='+')

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

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test
y_predicted = model.predict(X_test)
y_predicted
model.score(X_test, y_test)
model.predict_proba(X_test)
y_predicted = model.predict([[60]])
model.coef_
model.intercept_
import math
def sigmoid(x):
    return 1/(1+math.exp(-x))
def prediction_function(age):
    z = 0.127 * age - 4.973
    y = sigmoid(z)
    return y
```

age = 35
prediction_function(age)

0.370948
coef = 0.127
intercept = -4.973

Multiclass classification
iris = pd.read_csv("iris.csv")
iris.head
X = iris.drop('species', axis='columns')
y = iris.species
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = LogisticRegression(multi_class='multinomial')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
confusion_matrix = metrics.confusion_matrix()
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix)

accuracy = 1.00

Date __/__/__
Page _____

Q1.

i) time-yard company as correlation is highest
ii) accuracy is 76%

Q2 i) dropped animal name is non-numeric
80-20 ratio
ii) no missing value
iii) iw matrix shows class 7 & invertebrate was frequently misclassified as bug.
Overall accuracy is quite good as inferred from cm

18/5/25

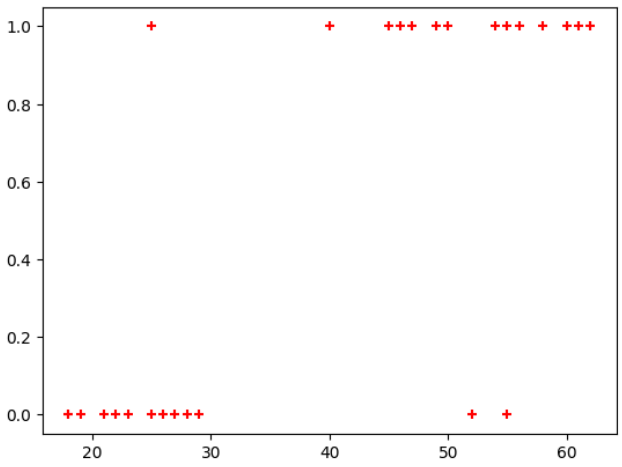
Code:

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import math
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn import metrics

df = pd.read_csv("insurance_data.csv")
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
X_train, X_test, y_train, y_test =
train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10
)
model = LogisticRegression()
model.fit(X_train, y_train)
y_predicted = model.predict(X_test)
y_predicted
model.score(X_test,y_test)
model.predict_proba(X_test)
y_predicted = model.predict([[60]])
y_predicted
model.coef_
model.intercept_
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
   y = sigmoid(z)
   return y
age = 35
prediction_function(age)


iris =  pd.read_csv("iris.csv")
X=iris.drop('species',axis='columns')# Features (sepal length, sepal width,
petal length, petal width)
y = iris.species
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LogisticRegression(multi_class='multinomial')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Multinomial Logistic Regression model on the test set:
{accuracy:.2f}")
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = ["Setosa", "Versicolor", "Virginica"])
cm_display.plot()
plt.show()
```

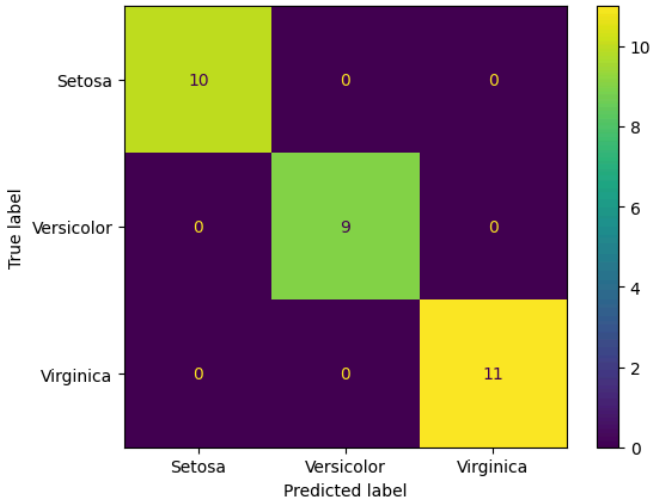
Output:

```
Out[4]: <matplotlib.collections.PathCollection at 0x7f42ebbb4d10>
```




```
array([1, 1, 0])
1.0
array([[0.06470655, 0.93529345],
       [0.10327333, 0.89672667],
       [0.92775258, 0.07224742]])
array([1])
array([[0.1274065]])
array([-4.97339194])
0.3709834769552775
```

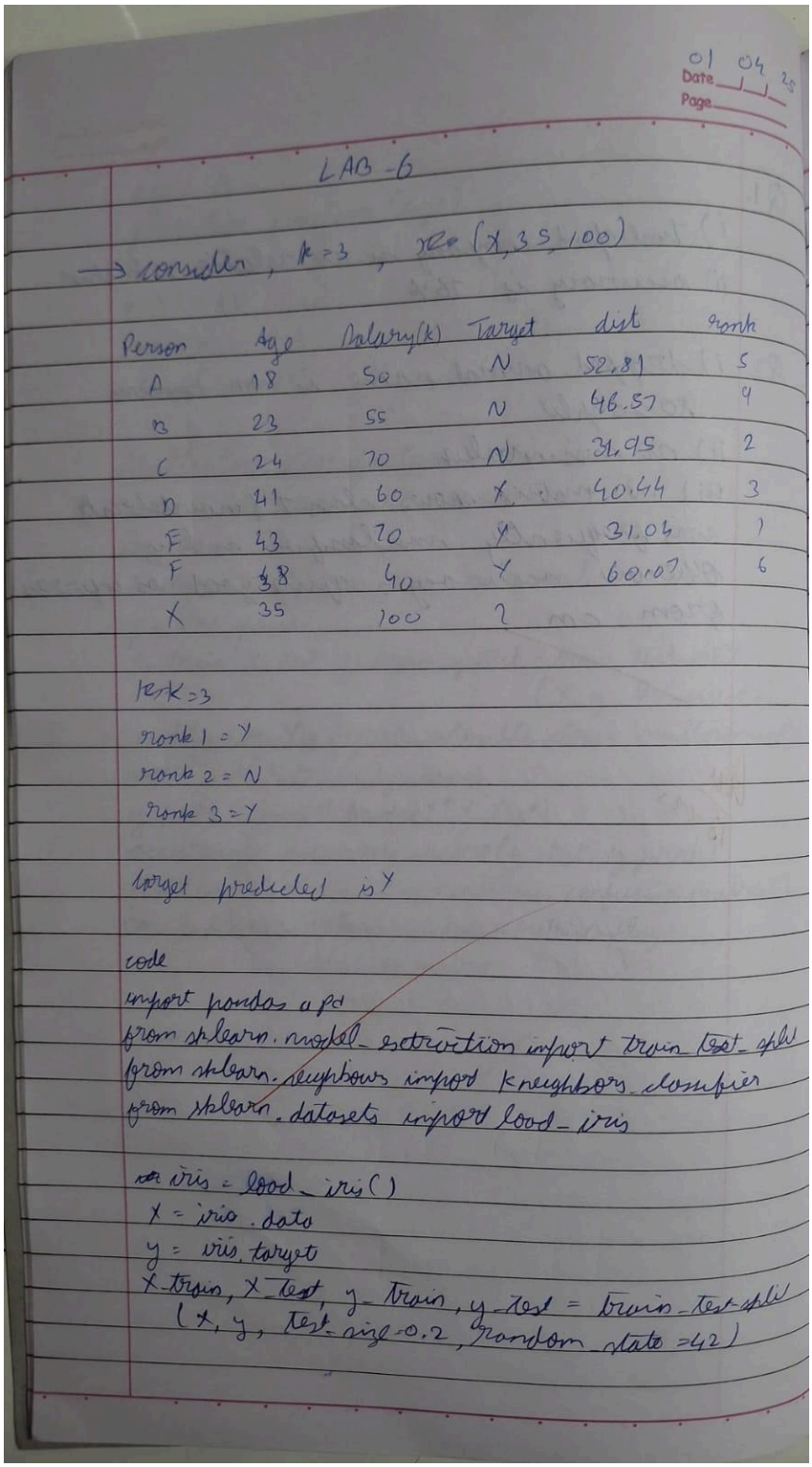
Accuracy of the Multinomial Logistic Regression model on the test set: 1.00



Program 6

Build KNN Classification model for a given dataset.

Screenshot:



LAB-6

→ consider, k=3, x= (X, 35, 100)

Person	age	Salary(k)	Target	dist	rank
A	18	50	N	52.81	5
B	23	55	N	46.57	4
C	24	70	N	31.95	2
D	41	60	X	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	6
X	35	100	?		

let k=3

rank 1 = Y

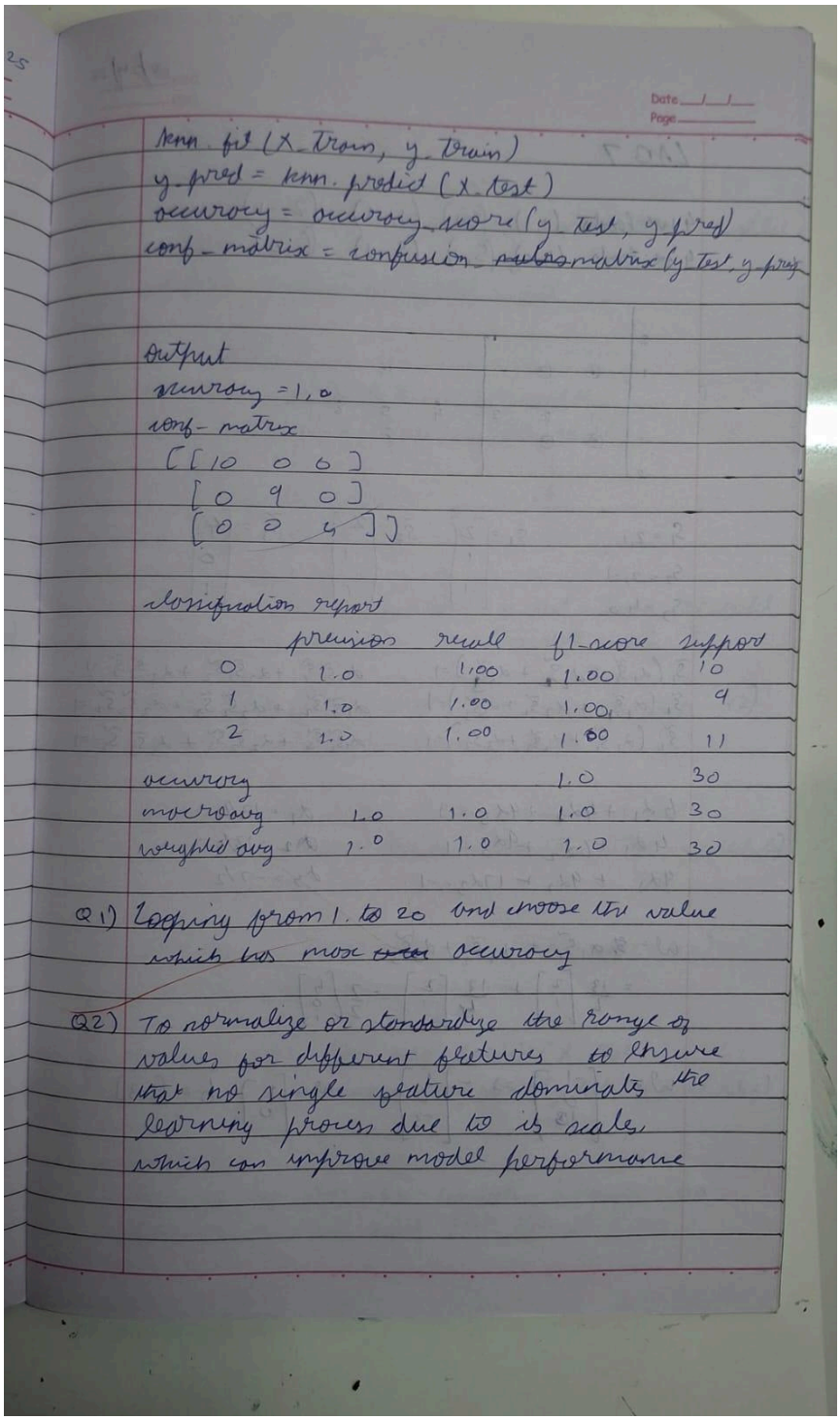
rank 2 = N

rank 3 = Y

target predicted is Y

code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size=0.2, random_state=42)
```



knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_~~submatrix~~matrix(y_test, y_pred

Output
accuracy = 1.0
conf-matrix
[[10 0 0]
[0 9 0]
[0 0 11]]

classification report

	precision	recall	f1-score	support
0	1.0	1.00	1.00	10
1	1.0	1.00	1.00	9
2	1.0	1.00	1.00	11
accuracy			1.0	30
macro avg	1.0	1.0	1.0	30
weighted avg	1.0	1.0	1.0	30

Q1) Looping from 1. to 20 and choose the value which has more ~~error~~ accuracy

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

Code:

```
import pandas as pd

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

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

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

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

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

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

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

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

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

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

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

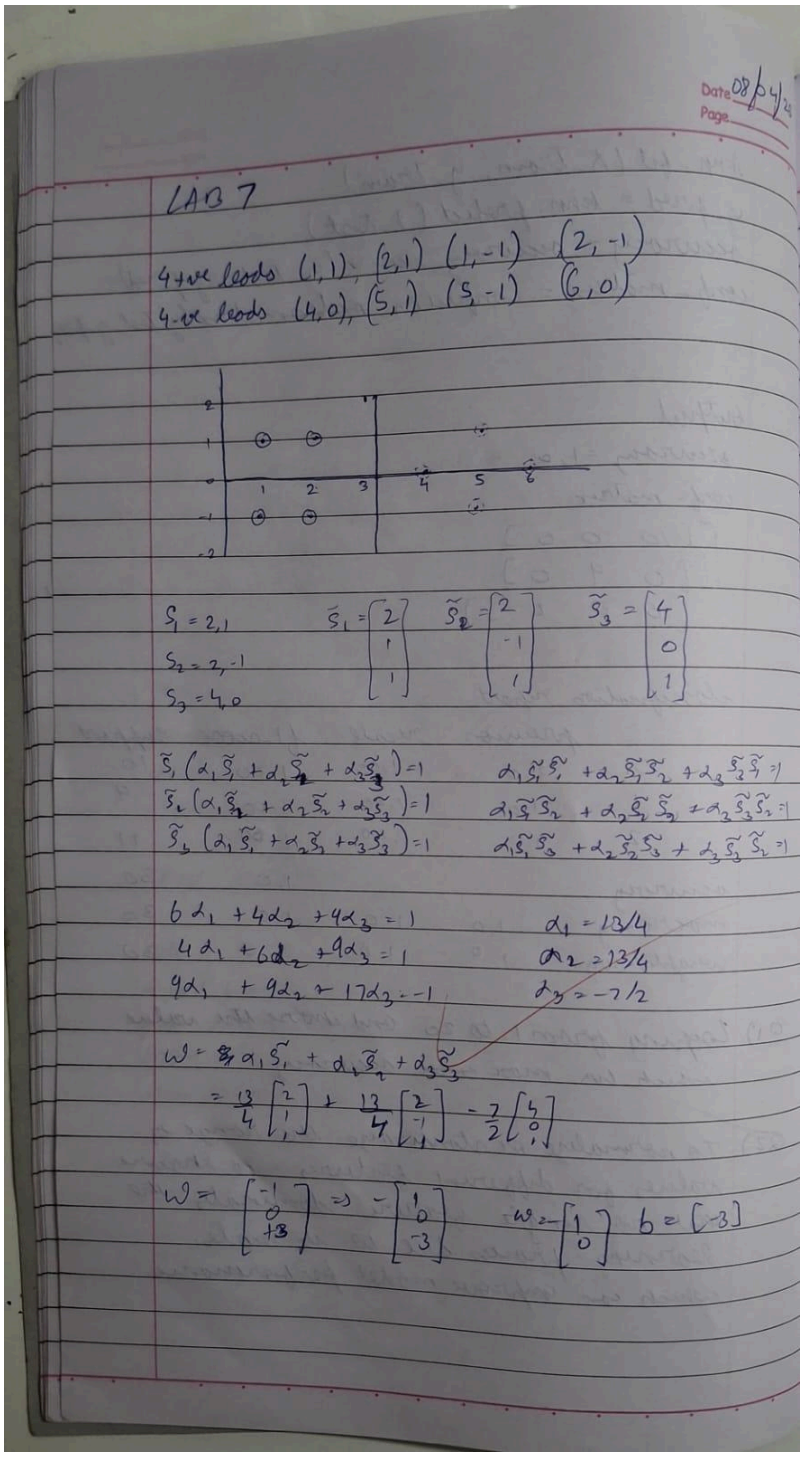
Output:

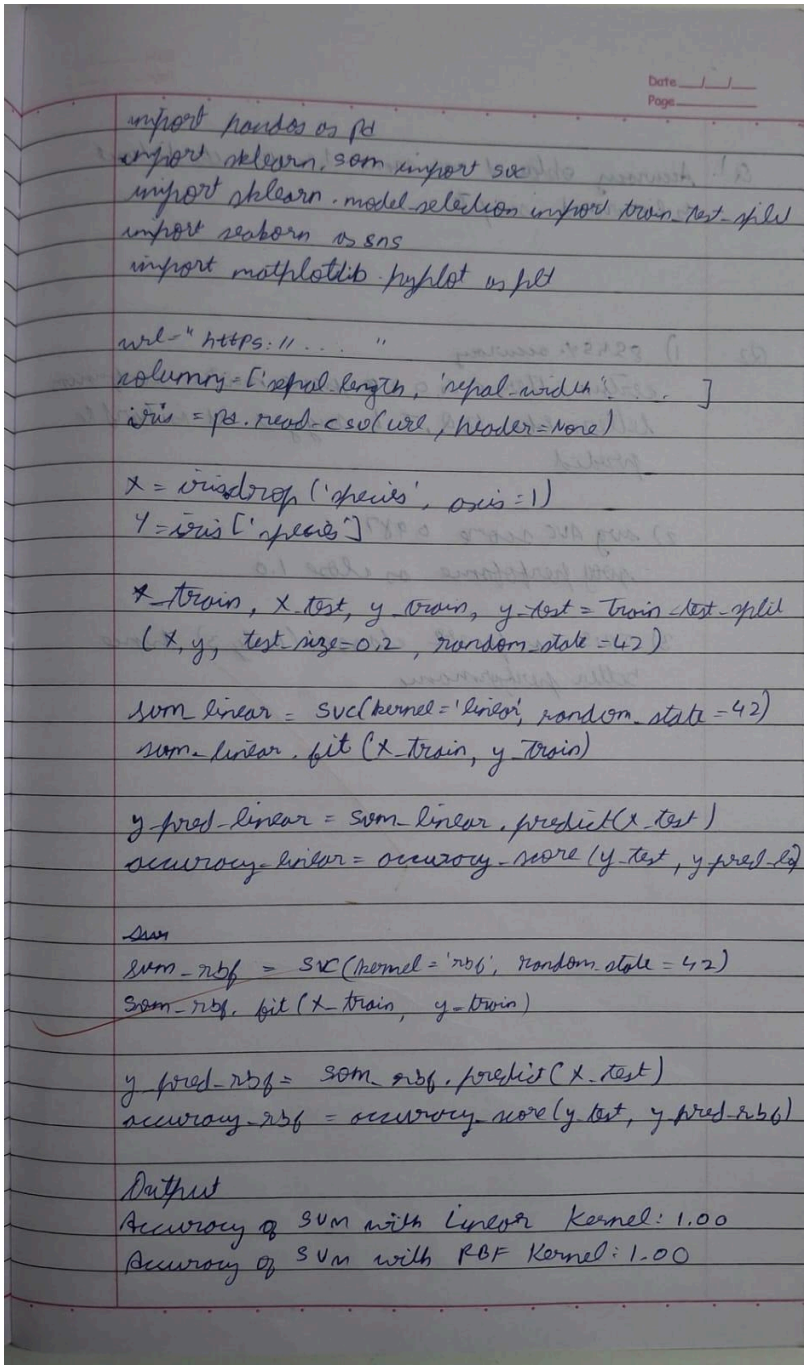
```
  sepal_length  sepal_width  petal_length  petal_width species
0          5.1          3.5           1.4          0.2  setosa
1          4.9          3.0           1.4          0.2  setosa
2          4.7          3.2           1.3          0.2  setosa
3          4.6          3.1           1.5          0.2  setosa
4          5.0          3.6           1.4          0.2  setosa
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Program 7

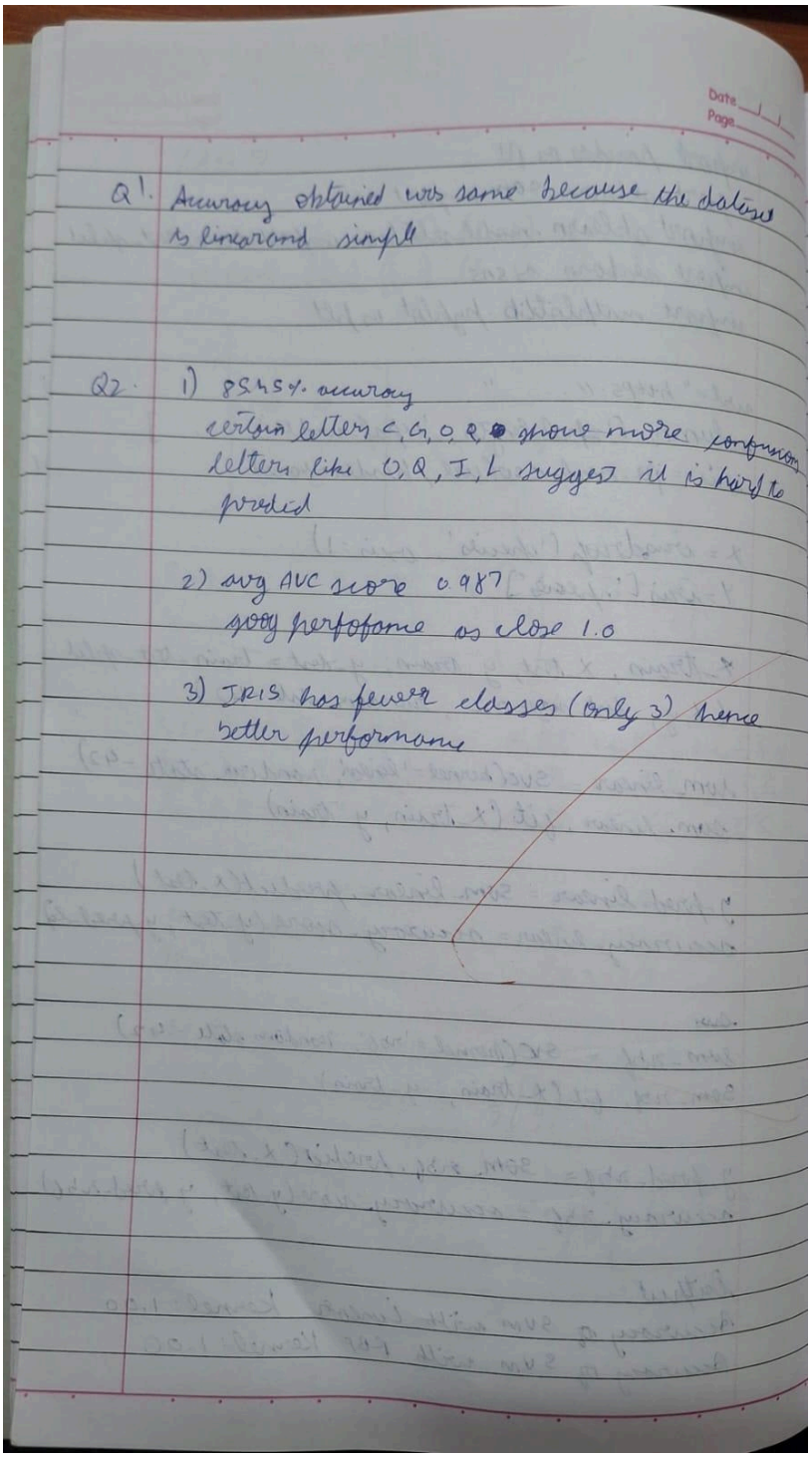
Build Support vector machine model for a given dataset

Screenshot:





```
import pandas as pd
import sklearn.svm import svc
import sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt

url = "https://..."
columns = ['sepal_length', 'sepal_width', ...]
iris = pd.read_csv(url, header=None)

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

X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.2, random_state=42)

svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)

y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_lin)

svm
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)

y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

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



Q1. Accuracy obtained was same because the dataset is linearly and simple

Q2. 1) 85.45% accuracy
certain letters C, G, O, Q, ● shows more confusion
letters like O, Q, I, L suggest it is hard to predict

2) avg AUC score 0.987
good performance as close 1.0

3) IRIS has fewer classes (only 3) hence better performance

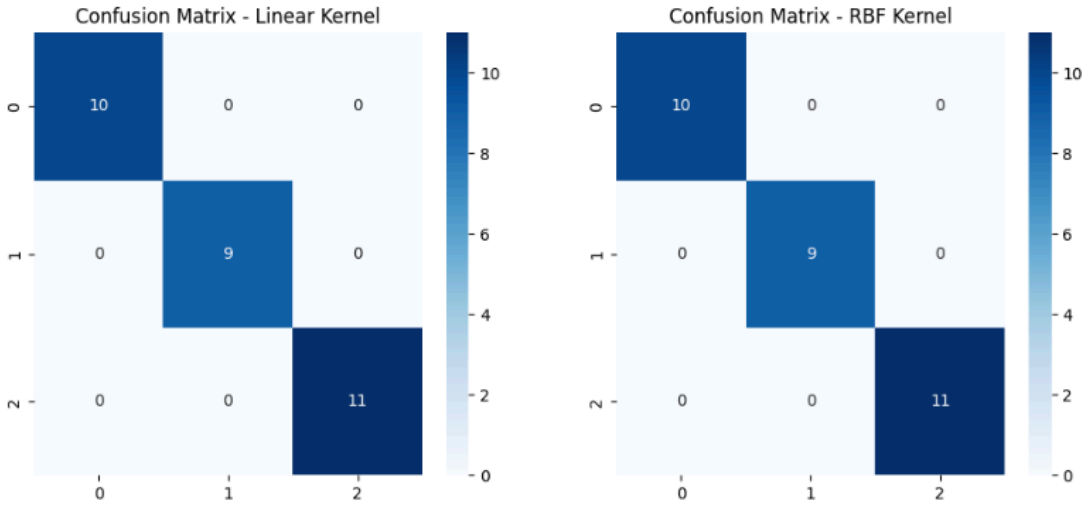
Code:

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

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

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

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

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

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

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

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

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

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

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

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

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

# Linear Kernel Confusion Matrix
```

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

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

plt.show()
```

Output:

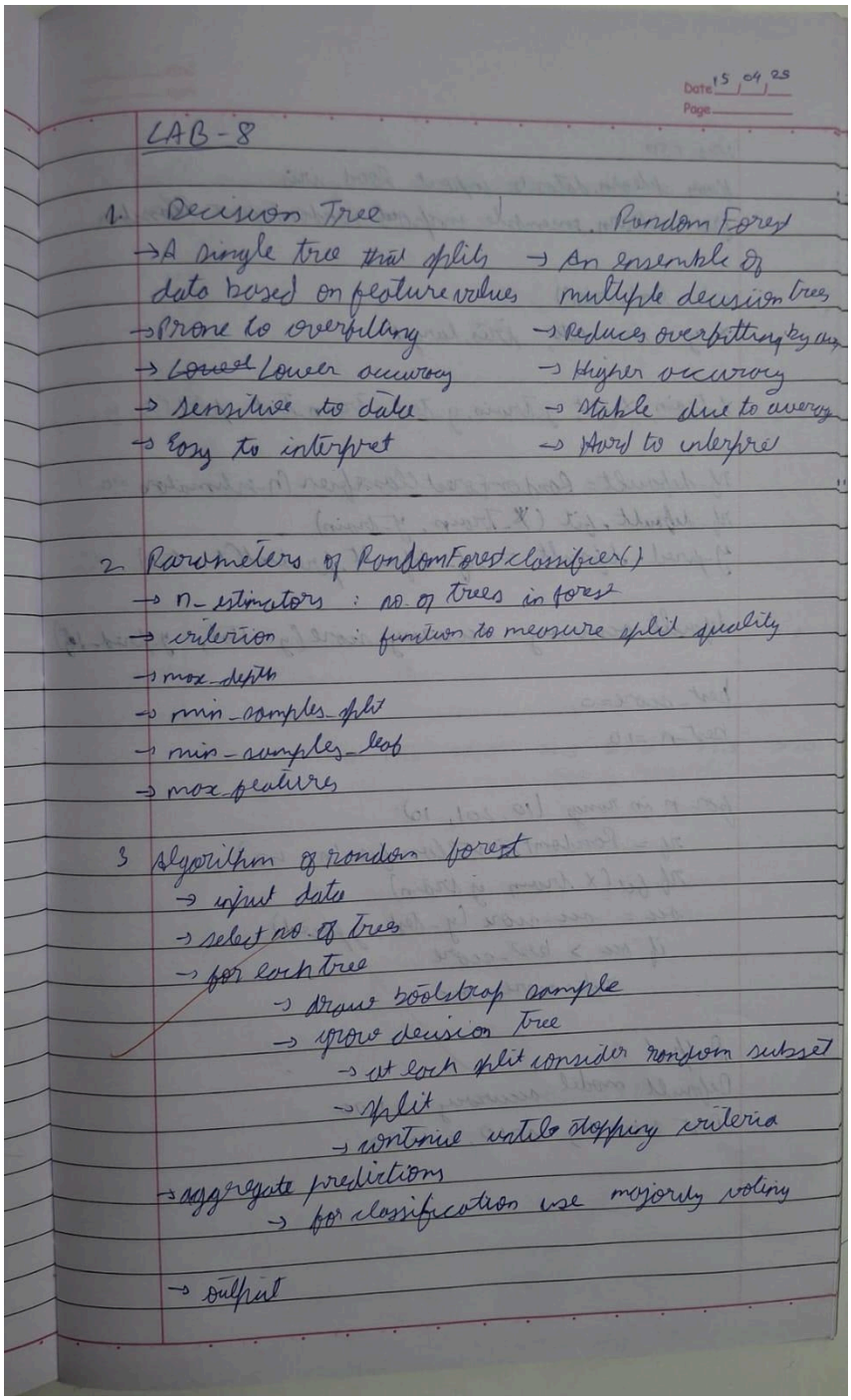
```
Accuracy of SVM with Linear Kernel: 1.00
Accuracy of SVM with RBF Kernel: 1.00
```

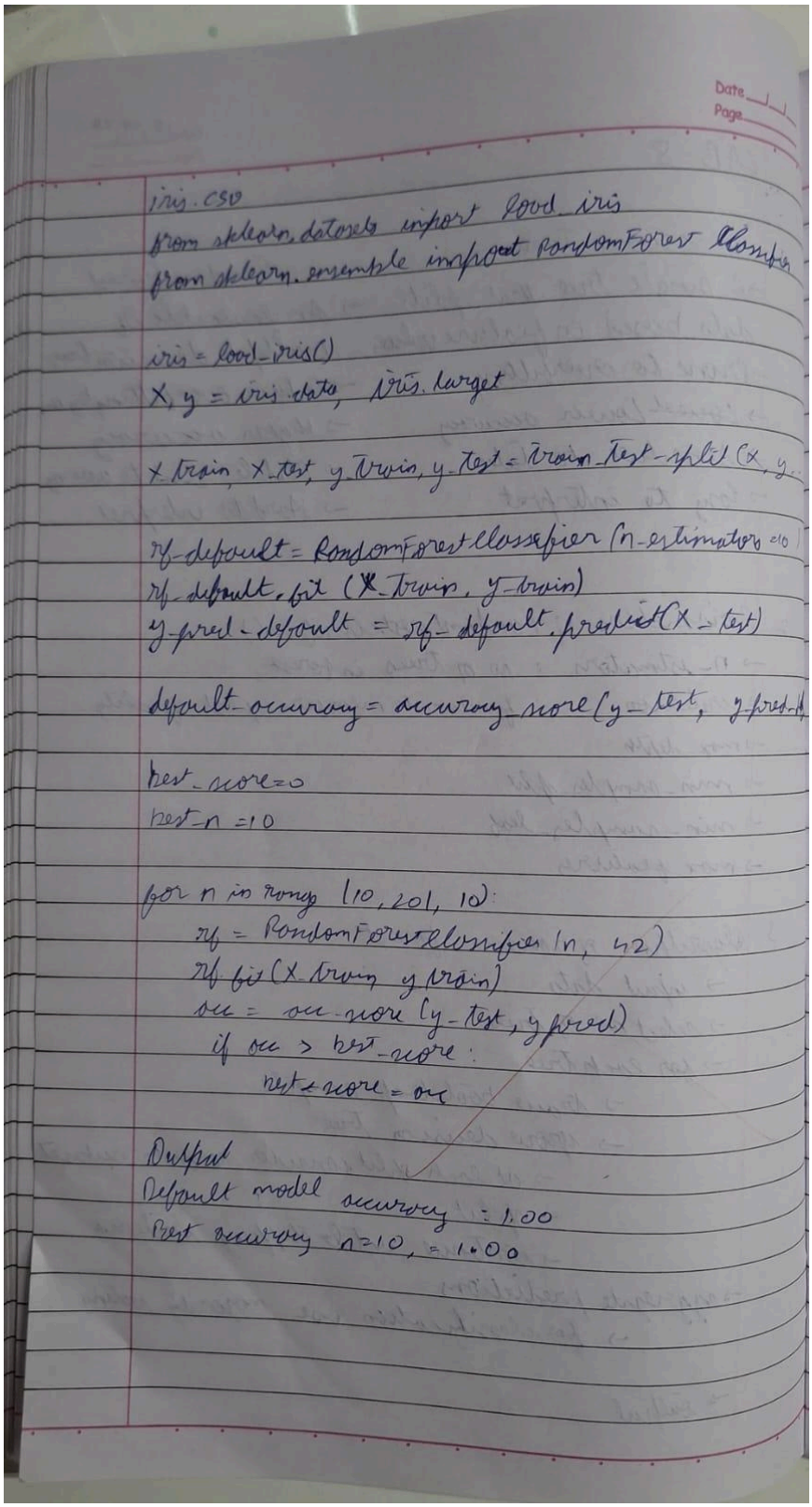


Program 8

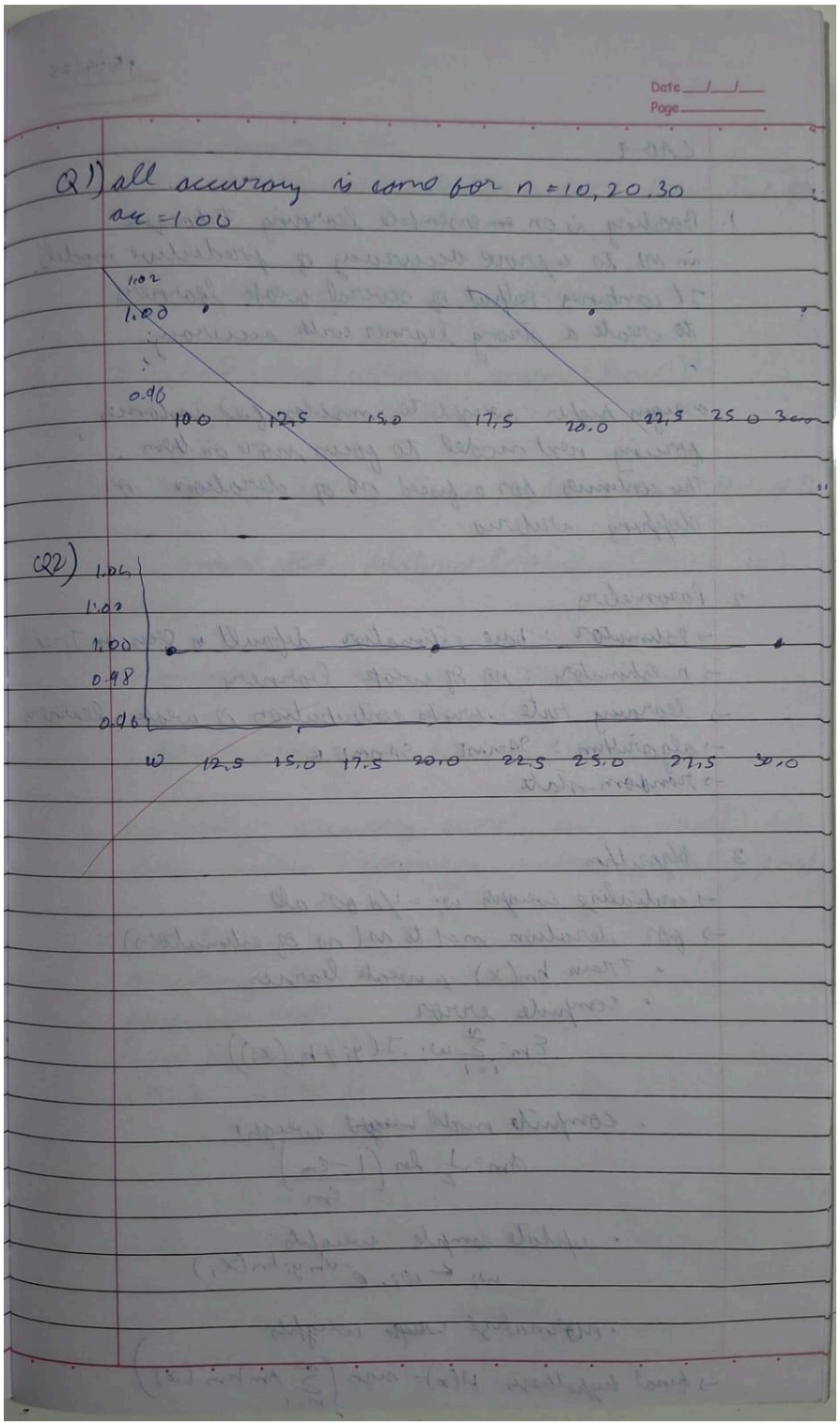
Implement Random forest ensemble method on a given dataset.

Screenshot:





```
iris.csv
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,

rf_default = RandomForestClassifier(n_estimators=10)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)

default_accuracy = accuracy_score(y_test, y_pred_d

best_score=0
best_n =10

for n in range(10, 201, 10):
    rf = RandomForestClassifier(n, 42)
    rf.fit(X_train, y_train)
    acc = accuracy_score(y_test, y_pred)
    if acc > best_score:
        best_score = acc

Output
Default model accuracy : 1.00
Best accuracy n=10, = 1.00
```


Date / /
Page
Q1) all accuracy is same for n = 10, 20, 30
Acc = 1.00
1.02
1.00
0.98
10.0 12.5 15.0 17.5 20.0 22.5 25.0 30.0
Q2)
1.04
1.02
1.00
0.98
0.96
10 12.5 15.0 17.5 20.0 22.5 25.0 27.5 30.0

Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

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

# Train default Random Forest model
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)

# Evaluate default model
default_accuracy = accuracy_score(y_test, y_pred_default)
print(f"Default model accuracy (n_estimators=10): {default_accuracy:.4f}")

# Fine-tune n_estimators to find best model
best_score = 0
best_n = 10

for n in range(10, 201, 10):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    if acc > best_score:
        best_score = acc
        best_n = n

print(f"Best accuracy: {best_score:.4f} with n_estimators = {best_n}")
```

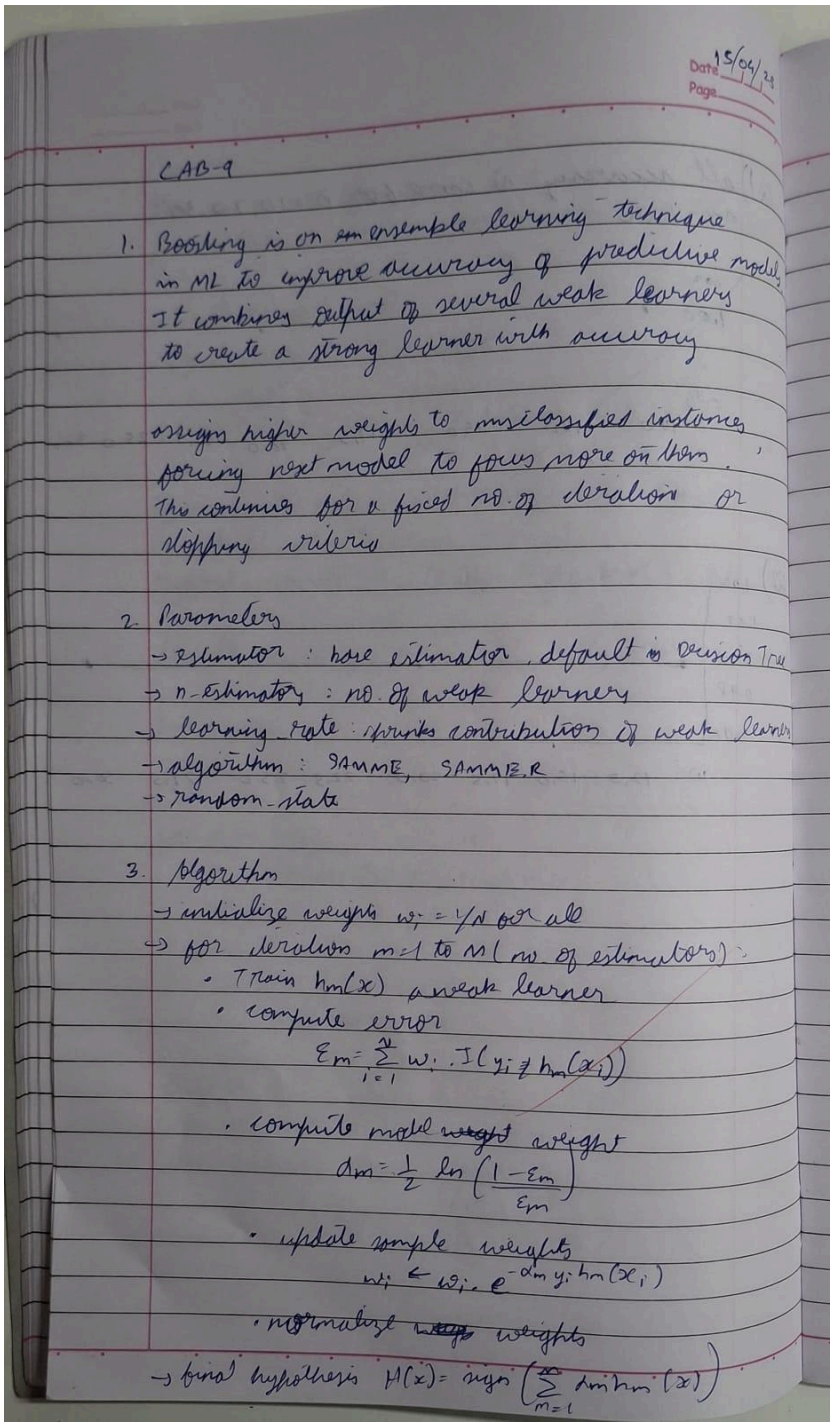
Output:

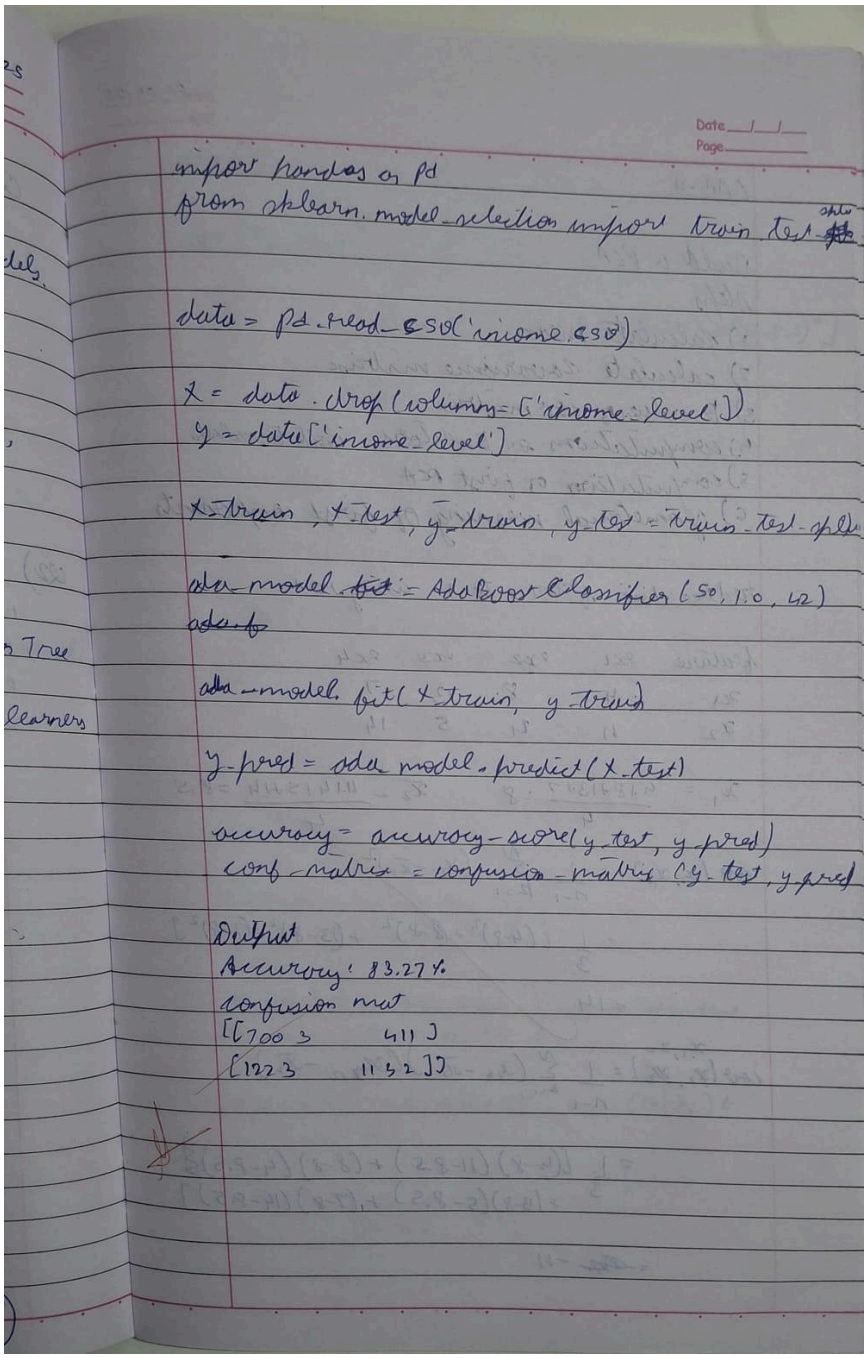
Default model accuracy (n_estimators=10): 1.0000
Best accuracy: 1.0000 with n_estimators = 10

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:





```
impor pandas as pd
from sklearn.model_selection impor train_test_split

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

X = data.drop(columns=['income_level'])
y = data['income_level']

X_train, X_test, y_train, y_test = train_test_split(

ada_model = AdaBoostClassifier(50, 1.0, 42)

ada_model.fit(X_train, y_train)

y_pred = ada_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred

Output
Accuracy: 83.27%
confusion mat
[[7003    411]
 [1223    1132]]
```

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
data = pd.read_csv('income.csv')

# Define features and target variable
X = data.drop(columns=['income_level'])
y = data['income_level']

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

# Build AdaBoost model (default base estimator is a DecisionTreeClassifier
with max_depth=1)
ada_model = AdaBoostClassifier(n_estimators=50, learning_rate=1.0,
random_state=42)

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

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

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

# Print results
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Confusion Matrix:")
print(conf_matrix)
```

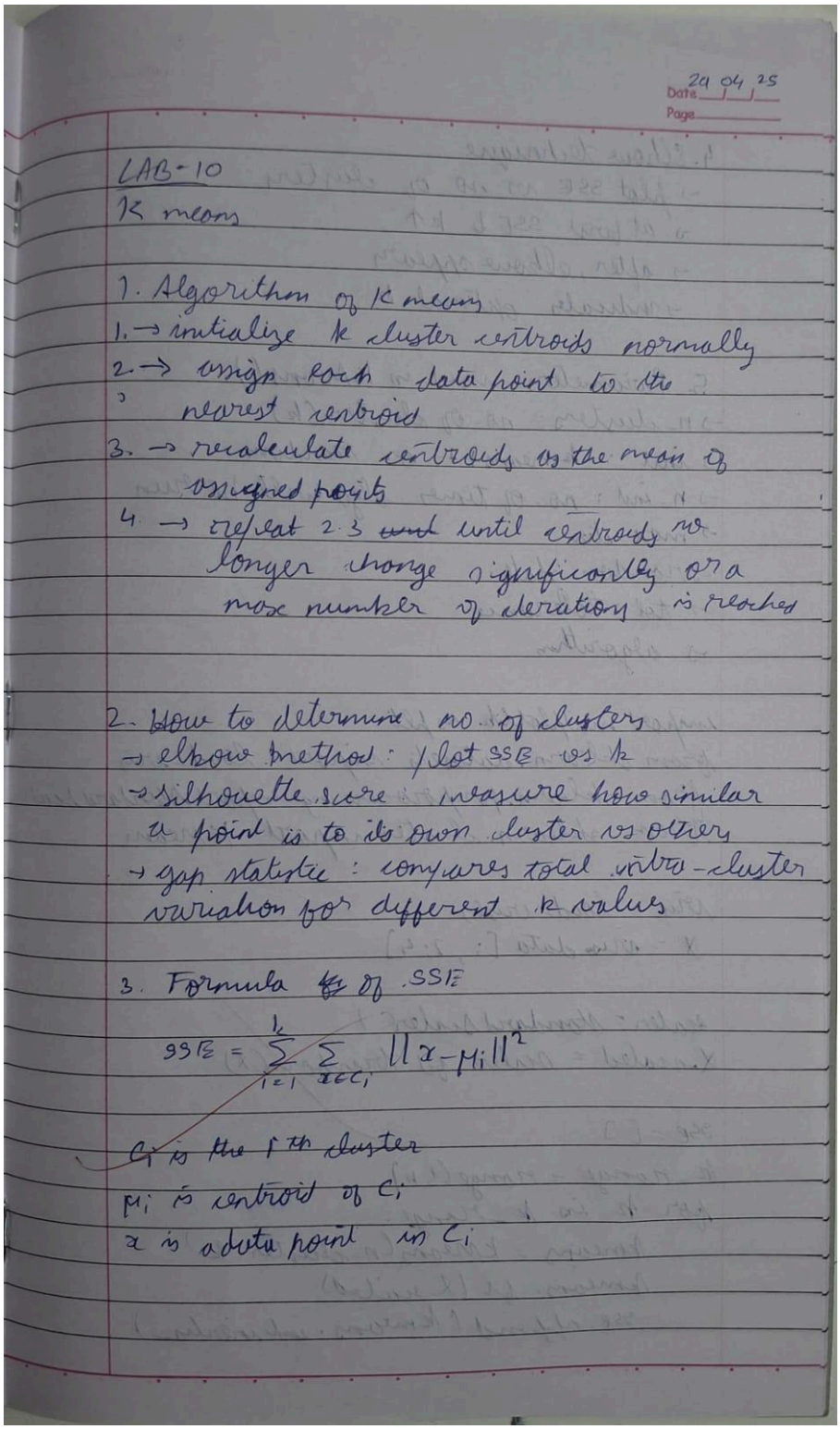
Output:

Accuracy: 83.27%
Confusion Matrix:
[[7003 411]
[1223 1132]]

Program 10

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

Screenshot:



Date 24/04/25
Page

LAB-10
K means

1. Algorithm of K means
1. → initialize k cluster centroids normally
2. → assign each data point to the nearest centroid
3. → recalculate centroids as the mean of assigned points
4. → repeat 2-3 until centroids no longer change significantly or a max number of iteration is reached

2. How to determine no. of clusters
→ elbow method : plot SSE vs k
→ silhouette score : measure how similar a point is to its own cluster vs other
→ gap statistic : compares total intra-cluster variation for different k values

3. Formula of SSE

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

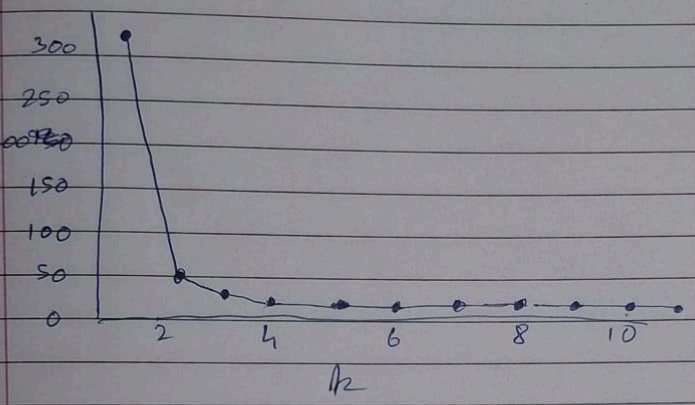
C_i is the i^{th} cluster
μ_i is centroid of C_i
x is a data point in C_i

4. Elbow technique
→ plot SSE vs no. of clusters
→ at first SSE ↓ k ↑
→ after, elbow appears
→ indicates optimal k

5. Parameters used in KMeans()
→ n_clusters: no. of cluster(k)
→ init : kmeans++ or random
→ n_init : no. of times algo will be run
→ max_iter
→ random_state
→ tol : tolerance
→ algorithm

```
import matplotlib as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

iris = load_iris()
X = iris.data[:, 2:4]

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

sse = []
k_range = range(1, n)
for k in k_range:
    kmeans = KMeans(n_cluster=k)
    kmeans.fit(X_scaled)
    sse.append(kmeans.inertia_)
```

```
plt.plot(k_range, sse, marker='o')
plt.title('elbow method')
plt.xlabel('k')
plt.ylabel('SSE')
```

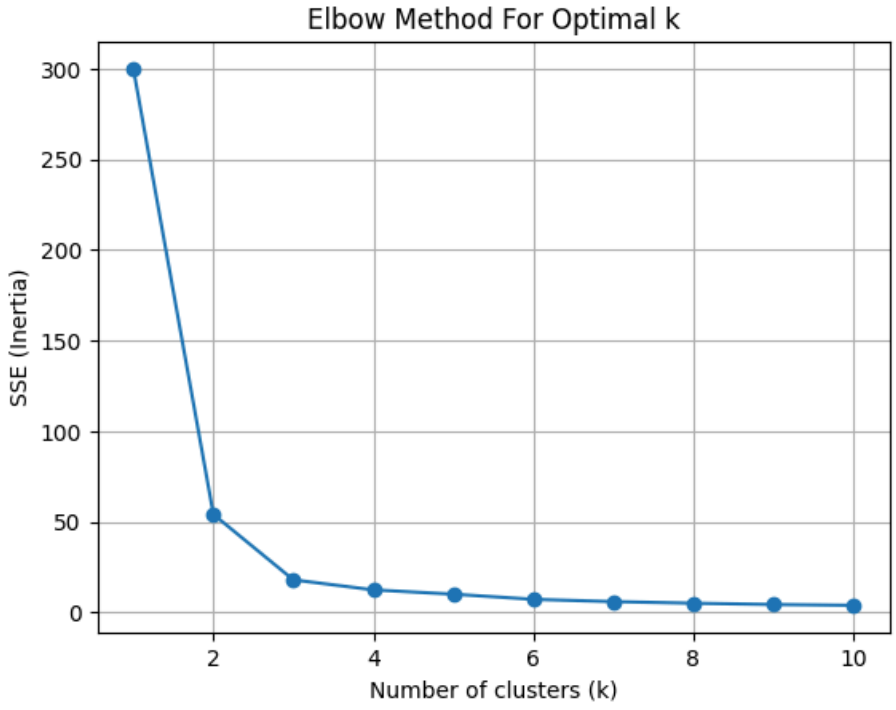


optimal k value = 4

Code:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Step 1: Load dataset and select petal length and width
iris = load_iris()
X = iris.data[:, 2:4]  # Petal length and petal width

# Step 2: Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Elbow Method
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse.append(kmeans.inertia_)

# Plotting SSE vs. Number of Clusters
plt.plot(k_range, sse, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('SSE (Inertia)')
plt.grid(True)
plt.show()
```

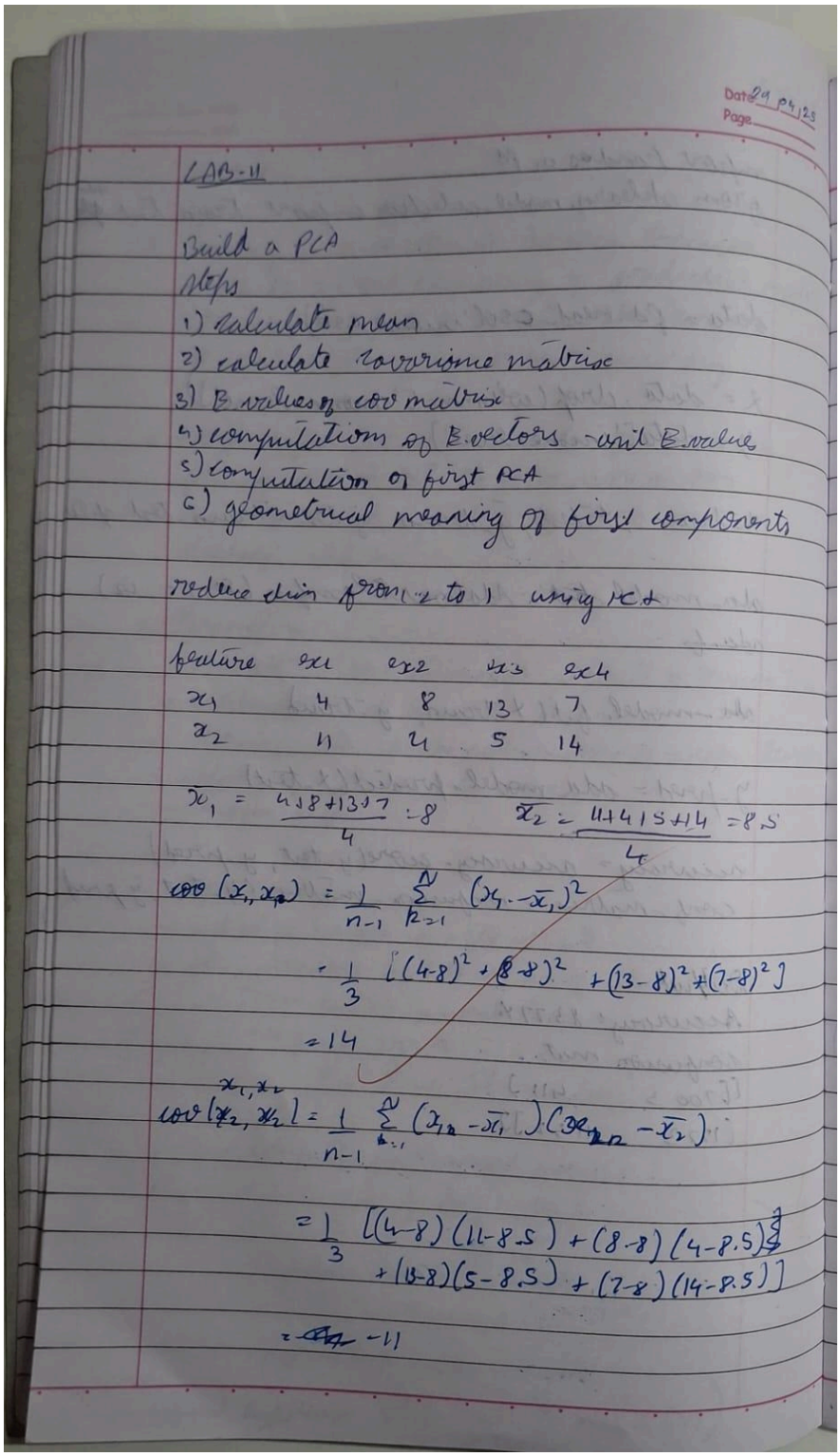
Output:



Program 11

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

Screenshot:



$cov(x_2, x_1) = -11$

$$cov(x_2, x_2) = \frac{1}{N-1} \sum_{n=1}^{N} (x_{2n} - \bar{x}_2)^2$$

$$= \frac{1}{3}\left[(4-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2\right]$$

$$= 23$$

cov matrix $= C = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$

E. values and E. vectors

$$C - \lambda I = \begin{bmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{bmatrix} = 0$$

$$(14-\lambda)(23-\lambda) - (121) = 0$$

$$\lambda^2 - 37\lambda + 201 = 0$$

$$\lambda = \frac{37 \pm \sqrt{565}}{2} \quad \lambda_1 = 30.849 \quad \lambda_2 = 6.0151$$

select $\lambda_1 \rightarrow$ most λ

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = 0$$

① $(14-\lambda_1)u_1 - 11u_2 = 0$
② $-11u_1 + (23-\lambda_1)u_2 = 0$ } not independent

$$\Rightarrow \frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t \Rightarrow u_1 = 11t, \quad u_2 = (14-\lambda_1)t$$

$$\Rightarrow u_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$$

$\|u_1\| = \sqrt{u^2 + (14-\lambda_1)^2}$

$= \sqrt{u^2 + (14 - 30385)^2}$

$= 19.7348$

unit E. vector corresponding to λ_1

$$e_1 = \begin{bmatrix} u/\|u_1\| \\ (14-\lambda_1)/\|u_1\| \end{bmatrix} = \begin{bmatrix} u/19.7348 \\ (14-\lambda_1)/19.7348 \end{bmatrix}$$

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

e_2 from eqn ②

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

let $\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix}$ be k^{th} sample

first PCA component

$$e_1^T \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$$

$= (0.5574)(x_{1k} - \bar{x}_1) - (0.8303)(x_{2k} - \bar{x}_2)$

Date __/__/__
Page _____

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA

digits = load_digits()
X, y = digits.data, digits.target

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

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

X_train, X_test, y_train, y_test = train_test_split
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Output

```
accuracy : 0.5361
```

Code:

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

# Step 1: Load the dataset
digits = load_digits()
X, y = digits.data, digits.target

# Step 2: Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA with n_components=2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Split the data into 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42, stratify=y)

# Step 5: Train a logistic regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Step 6: Make predictions and calculate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy with PCA (2 components): {accuracy:.4f}")
```

Output:

Accuracy with PCA (2 components): 0.5361