VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB

REPORTON

MACHINE LEARNING

Submitted by

Girish Kumar S K(1BM21CS068)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019(March 2024 to June 2024)

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**" is carried out by **Girish Kumar S K (1BM21CS068)** 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 Visveswaraya Technological University, Belgaumduring the year 2023-2024. The lab report has been approved as it satisfies the academic requirements in respect of **Machine Learning Lab - (22CS3PCMAL)** work prescribed for thesaid degree.

Dr. K. Panimozhi
Assistant Professor
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

Index Sheet

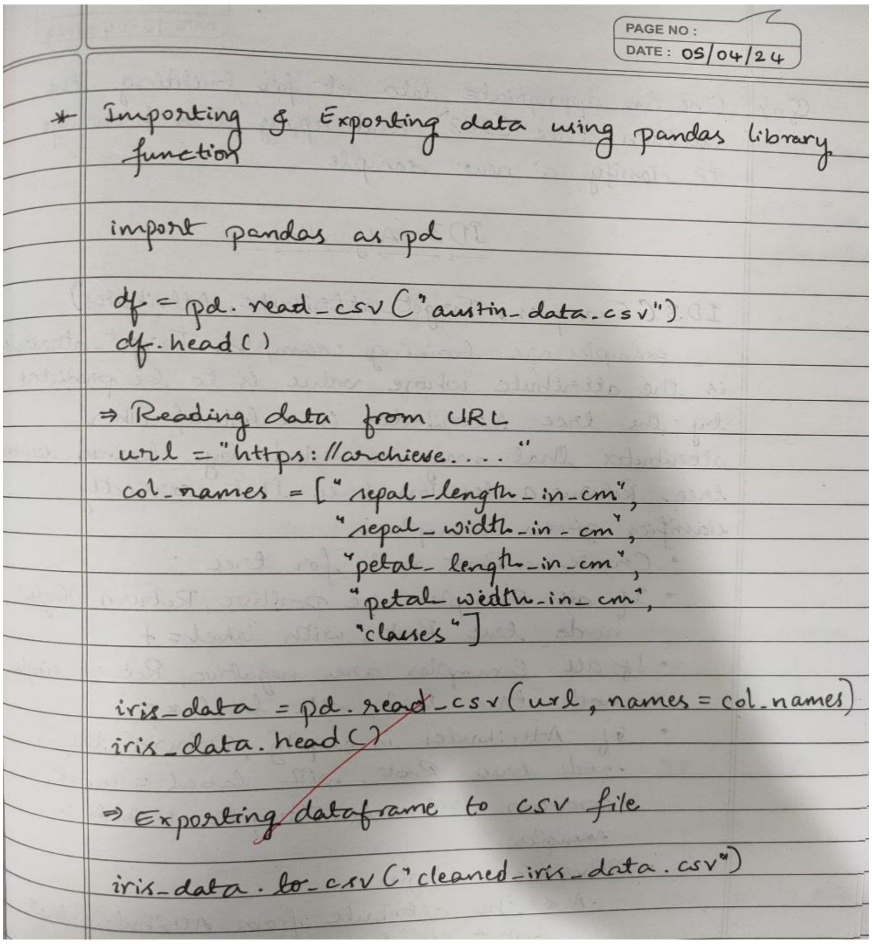
Course outcomes:

CO1	Apply machine learning techniques in computing systems
CO2	Evaluate the model using metrics
CO3	Design a model using machine learning to solve a problem
CO4	Conduct experiments to solve real-world problems using appropriate machine learning techniques

Lab1

Date: 05/04/2024

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


PAGE NO :
DATE : 05/04/24

* Importing & Exporting data using pandas library function

import pandas as pd

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

⇒ Reading data from URL
url = "https://archieve....."
col_names = ["sepal_length_in_cm",
"sepal_width_in_cm",
"petal_length_in_cm",
"petal_width_in_cm",
"classes"]

iris_data = pd.read_csv(url, names = col_names)
iris_data.head()

⇒ Exporting dataframe to csv file

iris_data.to_csv("cleaned_iris_data.csv")

CODE:

```
import pandas as pd

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

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

col_names = ["sepal_length_in_cm",
          "sepal_width_in_cm",
          "petal_length_in_cm",
          "petal_width_in_cm",
          "class"]

iris_data = pd.read_csv(url, names=col_names)
iris_data.head()
```

iris_data.to_csv("cleaned_iris_data.csv")

OUTPUT :

1. Dataset -

	zpid	city	streetAddress	zipcode	description	latitude	longitude	propertyTaxRate	garageSpaces	hasAssociation	...	numOfMiddleSchools
0	111373431	pflugerville	14424 Lake Victor Dr	78660	14424 Lake Victor Dr, Pflugerville, TX 78660 i...	30.430632	-97.663078	1.98	2	True	...	1
1	120900430	pflugerville	1104 Strickling Dr	78660	Absolutely GORGEOUS 4 Bedroom home with 2 full...	30.432673	-97.661697	1.98	2	True	...	1
2	2084491383	pflugerville	1408 Fort Dessau Rd	78660	Under construction - estimated completion in A...	30.409748	-97.639771	1.98	0	True	...	1
3	120901374	pflugerville	1025 Strickling Dr	78660	Absolutely darling one story home in charming ...	30.432112	-97.661659	1.98	2	True	...	1
4	60134862	pflugerville	15005 Donna Jane Loop	78660	Brimming with appeal & warm livability! Sleek ...	30.437368	-97.656860	1.98	0	True	...	1

2. After reading dataset from URL –

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

3. CSV file after exporting –

exported_listings - Excel

POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1		id	name	host_id	host_nam	neighbou	neighbou	latitude	longitude	room_typ	price	minimum	number_c	last_revie	reviews_p	calculated	availabilit	number_c
2	0	329172	Hillside de	1680871	Janet		78746	30.30085	-97.8079	Entire hor	495	3	7	########	0.05	1	363	1
3	1	329306	Urban Hor	880571	Angel		78702	30.27232	-97.7258	Private ro	63	2	570	########	4.36	5	55	45
4	2	331549	One Room	1690383	Sandra		78725	30.23911	-97.5863	Private ro	100	2	0			1	0	0
5	3	333815	Solar Sanc	372962	Kim		78704	30.25381	-97.7526	Private ro	102	2	164	########	1.26	1	36	18
6	4	333442	Rare Seclu	1698318	Virginia		78703	30.31267	-97.7664	Entire hor	286	3	163	########	1.32	1	271	15
7	5	335885	4Bed/2Bat	1707903	Stephen		78749	30.20621	-97.8558	Entire hor	300	28	43	########	0.33	1	90	0
8	6	334616	Great SXS	608539	D		78741	30.24481	-97.7238	Entire hor	250	1	2	########	0.02	1	0	0
9	7	337125	1800 Sq ft,	261883	Carolyn		78759	30.42035	-97.7669	Private ro	80	32	33	########	0.25	1	83	0
10	8	340045	Greenbelt	1725752	Shanti		78746	30.25937	-97.7925	Entire hor	400	1	1	########	0.09	1	0	1
11	9	335945	The Sheric	1555683	Patrick		78741	30.24026	-97.7334	Entire hor	221	3	34	########	0.26	1	27	2
12	10	340630	2 Bedroon	531267	Shelley		78757	30.33061	-97.7253	Entire hor	50	30	23	########	0.18	3	25	3
13	11	341596	South Aus	1733004	Julie		78704	30.24623	-97.759	Private ro	120	2	90	########	0.69	1	122	15
14	12	341382	SXSW | 4 I	1205884	Luke And Rachel		78702	30.26543	-97.7134	Entire hor	750	4	124	########	0.95	1	0	0
15	13	342243	Spacious F	1736662	Teresa		78759	30.41633	-97.7523	Private ro	45	31	3	########	0.06	1	317	0
16	14	342039	Garden Gu	1735494	Steph And Joey		78702	30.25158	-97.7315	Entire hor	116	1	342	########	2.62	2	120	30
17	15	343889	Close-in C	1744639	Darcy		78702	30.27293	-97.7229	Entire hor	47	30	32	########	0.24	1	214	4
18	16	343462	Charming	1742984	Rachel		78721	30.26895	-97.6895	Entire hor	99	1	255	########	1.95	1	143	12
19	17	345473	Exquisite	1751224	Whitney		78704	30.24619	-97.7457	Entire hor	308	2	164	########	1.38	1	86	44
20	18	345221	Austin His	1644218	Cecily		78701	30.27247	-97.748	Entire hor	299	3	8	########	0.06	2	0	0
21	19	5456	Walk to 6t	8028	Sylvia		78702	30.26057	-97.7344	Entire hor	96	2	623	########	3.71	1	318	44
22	20	347572	East Austi	1761506	Savanna		78702	30.26448	-97.7172	Entire hor	125	2	29	########	0.22	1	0	0
23	21	5769	NW Austi	8186	Elizabeth		78729	30.45697	-97.7842	Private ro	41	1	273	########	1.77	1	14	9
24	22	345536	Artist's Hn	1752493	Gigi		78704	30.2466	-97.7616	Entire hor	156	3	284	########	2.19	2	319	29
25	23	347935	for formul	1763742	Amber		78704	30.25152	-97.7621	Entire hor	643	5	1	########	0.01	1	0	0

Lab – 2

Date : 12-04-2024

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

Algorithm :

PAGE NO :
DATE : 12-04-2024

Qn 4 Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

ID3 Algorithm

ID3 (Examples, Target_attribute, Attributes)

examples are training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by learned decision tree. Returns a decision tree that correctly classifies given examples.

- Create a Root node for tree
- If all Examples are positive, Return single-node tree Root, with label = +
- If all Examples are negative, Return single-node tree Root, with label = −
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in examples
- Otherwise Begin
  - A ← the attribute from Attributes that best classifies examples
  - The decision attribute for Root ← A
  - For each possible value, v_i, of A,
    - Add new tree branch below Root, corresponding to test A = v_i
    - Let examples v_i, be the subset of examples that have value v_i for A
    - If Examples v_i, is empty

DATE :

- Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
- Else below this new branch add the subtree ID3 (Examples vi, Target_attribute, Attributes {A}))
- End
- Return Root

* The best attribute is one with highest information gain

CODE :

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import math

df = pd.read_csv('/content/diabetes.csv')
df

def calculate_entropy(data, target_column):
    total_rows = len(data)
    target_values = data[target_column].unique()

    entropy = 0
    for value in target_values:
        # Calculate the proportion of instances with the current value
        value_count = len(data[data[target_column] == value])
        proportion = value_count / total_rows
        entropy -= proportion * math.log2(proportion)

    return entropy

entropy_outcome = calculate_entropy(df, 'Outcome')
print(f"Entropy of the dataset: {entropy_outcome}")

def calculate_entropy(data, target_column): # for each categorical variable
    total_rows = len(data)
    target_values = data[target_column].unique()
```

```
    entropy = 0
    for value in target_values:
        # Calculate the proportion of instances with the current value
        value_count = len(data[data[target_column] == value])
        proportion = value_count / total_rows
        entropy -= proportion * math.log2(proportion) if proportion != 0 else 0

    return entropy

def calculate_information_gain(data, feature, target_column):

    # Calculate weighted average entropy for the feature
    unique_values = data[feature].unique()
    weighted_entropy = 0

    for value in unique_values:
        subset = data[data[feature] == value]
        proportion = len(subset) / len(data)
        weighted_entropy += proportion * calculate_entropy(subset, target_column)

    # Calculate information gain
    information_gain = entropy_outcome - weighted_entropy

    return information_gain

for column in df.columns[:-1]:
    entropy = calculate_entropy(df, column)
    information_gain = calculate_information_gain(df, column, 'Outcome')
    print(f"{column} - Entropy: {entropy:.3f}, Information Gain: {information_gain:.3f}")

# Feature selection for the first step in making decision tree
selected_feature = 'DiabetesPedigreeFunction'

# Create a decision tree
clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)
X = df[[selected_feature]]
y = df['Outcome']
clf.fit(X, y)

plt.figure(figsize=(8, 6))
plot_tree(clf, feature_names=[selected_feature], class_names=['0', '1'], filled=True,
rounded=True)
plt.show()

def id3(data, target_column, features):
    if len(data[target_column].unique()) == 1:
        return data[target_column].iloc[0]
```

```
    if len(features) == 0:
        return data[target_column].mode().iloc[0]

    best_feature = max(features, key=lambda x: calculate_information_gain(data, x,
target_column))

    tree = {best_feature: {}}

    features = [f for f in features if f != best_feature]

    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        tree[best_feature][value] = id3(subset, target_column, features)

    return tree

id3(df, 'Outcome', ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'] )
```

OUTPUT :

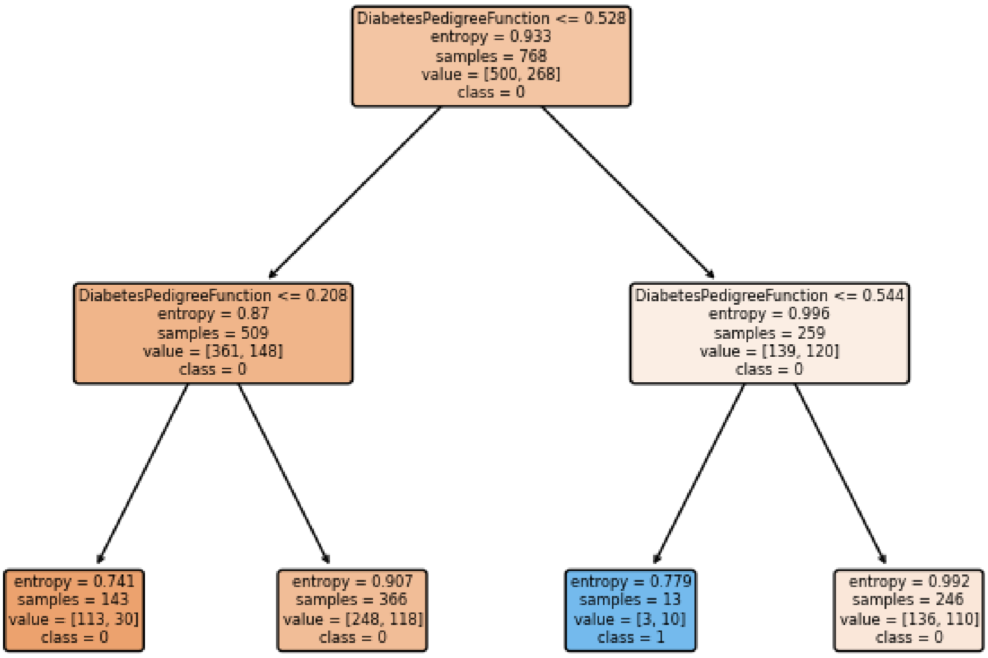
1. Entropy of Dataset :

```
Entropy of the dataset: 0.9331343166407831
```

2. Entropy and Information Gain of each feature

```
Pregnancies - Entropy: 3.482, Information Gain: 0.062
Glucose - Entropy: 6.751, Information Gain: 0.304
BloodPressure - Entropy: 4.792, Information Gain: 0.059
SkinThickness - Entropy: 4.586, Information Gain: 0.082
Insulin - Entropy: 4.682, Information Gain: 0.277
BMI - Entropy: 7.594, Information Gain: 0.344
DiabetesPedigreeFunction - Entropy: 8.829, Information Gain: 0.651
Age - Entropy: 5.029, Information Gain: 0.141
```

3. Decision Tree :


DiabetesPedigreeFunction <= 0.528
entropy = 0.933
samples = 768
value = [500, 268]
class = 0
DiabetesPedigreeFunction <= 0.208
entropy = 0.87
samples = 509
value = [361, 148]
class = 0
DiabetesPedigreeFunction <= 0.544
entropy = 0.996
samples = 259
value = [139, 120]
class = 0
entropy = 0.741
samples = 143
value = [113, 30]
class = 0
entropy = 0.907
samples = 366
value = [248, 118]
class = 0
entropy = 0.779
samples = 13
value = [3, 10]
class = 1
entropy = 0.992
samples = 246
value = [136, 110]
class = 0

Lab 3

Date : 03/05/2024

1.Linear Regression

Observation Screenshot:

DATE :

Q. Implement Linear & Multiple Linear Regression algorithm using appropriate dataset

* Linear Regression
1. Import dataset (2 features)
2. store independent feature to 'x' & dependent feature to 'y'
3. estimate b0 & b1 coefficient using Linear ~~two~~ Regression model
4. plot regression line
5. print b0, b1 and predict value of y for new value of x.

Code and Output :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import LinearRegression
```

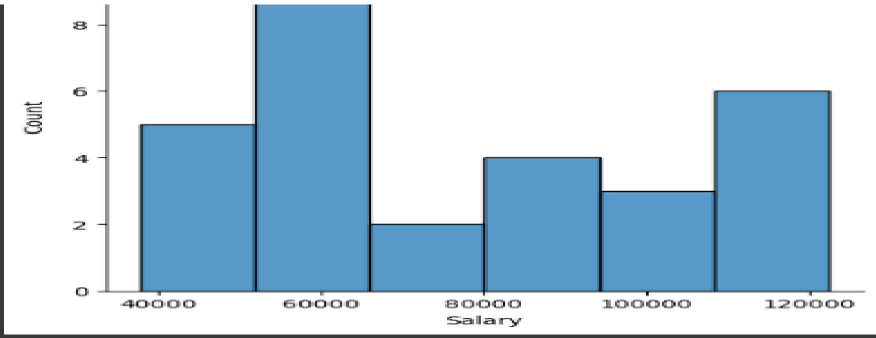
```
# Get dataset
df_sal = pd.read_csv('/content/Salary_Data.csv')
df_sal.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
# Describe data
df_sal.describe()
```

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

```
# Data distribution
plt.title('Salary Distribution Plot')
sns.displot(df_sal['Salary'])
plt.show()
```




```
# Relationship between Salary and Experience
plt.scatter(df_sal['YearsExperience'], df_sal['Salary'], color = 'lightcoral')
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
```


```
# Splitting variables
X = df_sal.iloc[:, :1]  # independent
y = df_sal.iloc[:, 1:]  # dependent
```

```
# Splitting dataset into test/train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
# Regressor model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

LinearRegression

```
# Prediction result
y_pred_test = regressor.predict(X_test)     # predicted value of y_test
y_pred_train = regressor.predict(X_train)   # predicted value of y_train
```

```
# Prediction on training set
plt.scatter(X_train, y_train, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()
```

```python
# Prediction on test set
plt.scatter(X_test, y_test, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()
```


```python
# Regressor coefficients and intercept
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
```

```
Coefficient: [[9312.57512673]]
Intercept: [26780.09915063]
```

2.Multiple Linear Regression

Observation Screenshot

Multiple Linear
* ~~Logistic~~ Regression
1. split dataset into training & testing set
2. we can see multiple independent variables
3. create regression model
   regression = LinearRegression()
4. Fit train set
5. Test model using test set
6. compare actual value & predicted value

In [34]:
```
#Importing the Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# import warnings
import warnings
warnings.filterwarnings("ignore")

# We will use some methods from the sklearn module
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score
```

In [25]:
```
# Reading the Dataset
df = pd.read_csv("data.csv")
```

In [26]:
```
df.head()
```

Out[26]:

	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105

In [27]:
```
df.shape
```

Out[27]: (36, 5)

In [28]:
```
df.corr(numeric_only=True)
```

Out[28]:

	Volume	Weight	CO2
Volume	1.000000	0.753537	0.592082
Weight	0.753537	1.000000	0.552150
CO2	0.592082	0.552150	1.000000

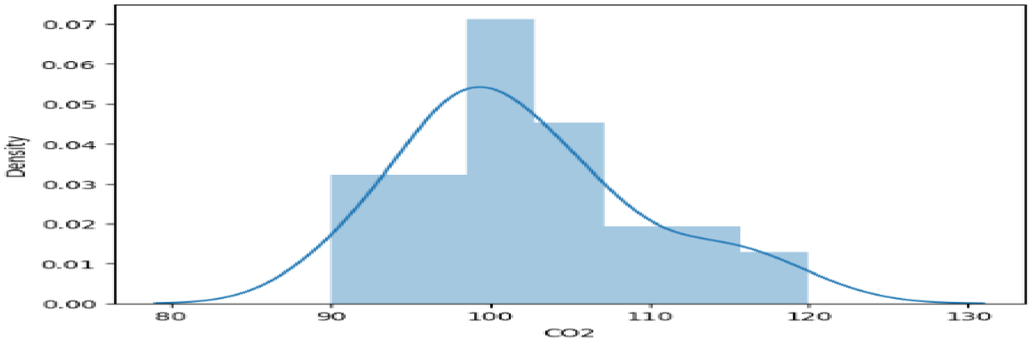
In [29]:
```
print(df.describe())
```

```
            Volume       Weight         CO2
count    36.000000    36.000000   36.000000
mean   1611.111111  1292.277778  102.027778
std     388.975047   242.123889    7.454571
min     900.000000   790.000000   90.000000
25%    1475.000000  1117.250000   97.750000
50%    1600.000000  1329.000000   99.000000
75%    2000.000000  1418.250000  105.000000
max    2500.000000  1746.000000  120.000000
```

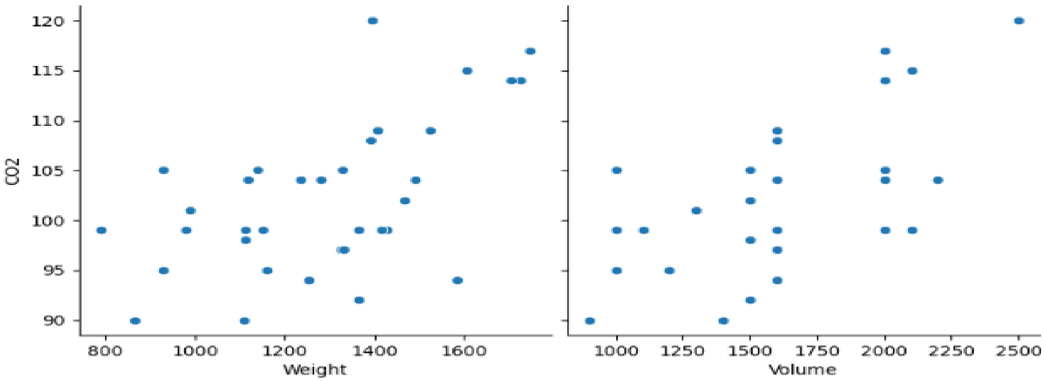
In [30]:
```
#Setting the value for X and Y
x = df[['Weight', 'Volume']]
y = df['CO2']
```

In [31]:
```
fig, axs = plt.subplots(2, figsize = (5,5))
plt1 = sns.boxplot(df['Weight'], ax = axs[0])
plt2 = sns.boxplot(df['Volume'], ax = axs[1])
plt.tight_layout()
```

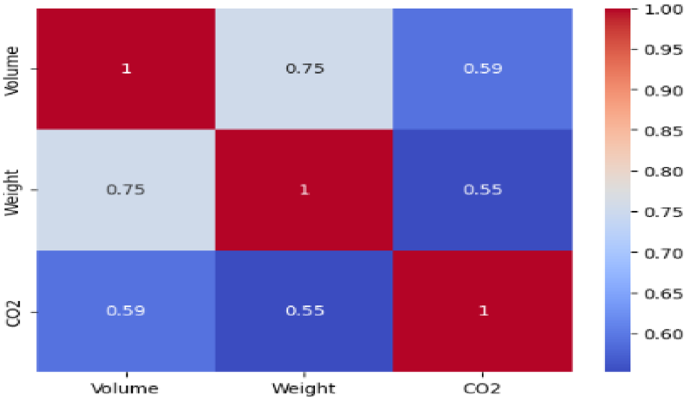

```
2]: sns.distplot(df['CO2']);
```




```
In [33]: sns.pairplot(df, x_vars=['Weight', 'Volume'], y_vars='CO2', height=4, aspect=1, kind='scatter')
         plt.show()
```




```
In [35]: # Create the correlation matrix and represent it as a heatmap.
         sns.heatmap(df.corr(numeric_only=True), annot = True, cmap = 'coolwarm')
         plt.show()
```




```
In [36]: X_train,X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 100)
```

In [36]:
```
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 100)
```
In [37]:
```
y_train.shape
```
Out[37]: (25,)

In [38]:
```
y_test.shape
```
Out[38]: (11,)

In [39]:
```
reg_model = linear_model.LinearRegression()
```
In [40]:
```
#Fitting the Multiple Linear Regression model
reg_model = LinearRegression().fit(X_train, y_train)
```
In [41]:
```
#Printing the model coefficients
print('Intercept: ',reg_model.intercept_)
# pair the feature names with the coefficients
list(zip(X, reg_model.coef_))
```
```
Intercept:  74.33882836589245
```
Out[41]: [('Weight', 0.0171800645996374), ('Volume', 0.0025046399866402976)]

In [42]:
```
#Predicting the Test and Train set result
y_pred= reg_model.predict(X_test)
x_pred= reg_model.predict(X_train)
```
In [43]:
```
print("Prediction for test set: {}".format(y_pred))
```
```
Prediction for test set: [ 90.41571939 102.16323413  99.56363213 104.56661845 101.54657652
  95.94770019 108.64011848 102.22654214  92.80374837  97.27327129
  97.57074463]
```
In [44]:
```
#Actual value and the predicted value
reg_model_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred})
reg_model_diff
```
Out[44]:

	Actual value	Predicted value
0	99	90.415719
19	105	102.163234
32	104	99.563632
35	120	104.566618
7	92	101.546577
12	99	95.947700
29	114	108.640118
33	108	102.226542
5	105	92.803748
1	95	97.273271
18	104	97.570745

In [45]:
```
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

print('Mean Absolute Error:', mae)
print('Mean Square Error:', mse)
print('Root Mean Square Error:', r2)
```
```
Mean Absolute Error: 6.901980901636316
Mean Square Error: 63.39765310998794
Root Mean Square Error: 7.96226432053018
```

3.KNN Algorithm:

Observation Screenshot

* Build KNN classification model for given dataset

Algorithm :

1> Define value of K and a distance metre

2> for given point, calculate distance between given point & every other point in dataset

3> choose k, closest points

4> The class/value of given point is majority of that k points.

If a euclidian distance is used as distance metre :

then $d = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$

Code and Output :

In [1]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris = datasets.load_iris()

x = iris.data
y = iris.target

print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0 - Iris-Setosa, 1 - Iris-Versicolour, 2 - Iris-Virginica')
print(y)
```

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

In [5]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#to make predictions on our test data
y_pred=classifier.predict(x_test)

print('Prediction -')

for i,test in enumerate(x_test) :
  print(f'{test} - {y_pred[i]}')

# print('Confusion Matrix')
# print(confusion_matrix(y_test,y_pred))
# print('Accuracy Metrics')
# print(classification_report(y_test,y_pred))
```

```
Prediction -
[5.2 4.1 1.5 0.1] - 0
[5.5 2.3 4.  1.3] - 1
[6.7 3.1 4.7 1.5] - 1
```

Lab 4
Date : 17/05/2024
Logistic Regression Algorithm
Observation Screenshot :

03/05/2024

* Implement Logistic Regression using appropriate dataset.

```
func logistR (x, y, learning rate, num_its):
    Initialize random values for weights (w) & bias (b)
    for i=1 to num_its :
        logits = x * w + b
        pred = sigmoid (logits)
        loss = compute_loss (y, pred)
        update weights & bias using gradients
    return w, b

func sigmoid (x)
    return 1 / (1 + exp(-x))

func compute_loss (y_true, y_pred):
    loss = -mean (y_true * log (y_pred)
            + (1 + y_true) * log (1 - y_pred))
    return loss
```

Code and Output :

In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly as py
import plotly.graph_objs as go
import time

init_notebook_mode(connected=True)
```

In [ ]:
```python
def sigmoid(X, weight):
    z = np.dot(X, weight)
    return 1 / (1 + np.exp(-z))
```

In [ ]:
```python
def loss(h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

In [ ]:
```python
def gradient_descent(X, h, y):
    return np.dot(X.T, (h - y)) / y.shape[0]
def update_weight_loss(weight, learning_rate, gradient):
    return weight - learning_rate * gradient
```

In [ ]:
```python
def log_likelihood(x, y, weights):
    z = np.dot(x, weights)
    ll = np.sum( y*z - np.log(1 + np.exp(z)) )
    return ll
```

In [ ]:

```
def gradient_ascent(X, h, y):
    return np.dot(X.T, y - h)
def update_weight_mle(weight, learning_rate, gradient):
    return weight + learning_rate * gradient
```

In [ ]:

```
data = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn.csv")
print("Dataset size")
print("Rows {} Columns {}".format(data.shape[0], data.shape[1]))
print("Columns and data types")
pd.DataFrame(data.dtypes).rename(columns = {0:'dtype'})
```

```
Dataset size
Rows 7043 Columns 21
Columns and data types
```

Out[ ]:

	dtype
customerID	object
gender	object
SeniorCitizen	int64

In [ ]:

```
df = data.copy()
```

In [ ]:

```
churns = ["Yes", "No"]
fig = {
    'data': [
        {
            'x': df.loc[(df['Churn']==churn), 'MonthlyCharges'] ,
            'y': df.loc[(df['Churn']==churn),'tenure'],
            'name': churn, 'mode': 'markers',
        } for churn in churns
    ],
    'layout': {
        'title': 'Tenure vs Monthly Charges',
        'xaxis': {'title': 'Monthly Charges'},
        'yaxis': {'title': "Tenure"}
    }
}

py.offline.iplot(fig)
```

In [ ]:

```
figs = []
for churn in churns:
    figs.append(
        go.Box(
            y = df.loc[(df['Churn']==churn),'tenure'],
            name = churn
        )
    )
layout = go.Layout(
    title = "Tenure",
    xaxis = {"title" : "Churn?"},
    yaxis = {"title" : "Tenure"},
    width=800,
    height=500
)

fig = go.Figure(data=figs, layout=layout)
py.offline.iplot(fig)
```

```
In [ ]: figs = []

        for churn in churns:
            figs.append(
                go.Box(
                    y = df.loc[(df['Churn']==churn),'MonthlyCharges'],
                    name = churn
                )
            )
        layout = go.Layout(
            title = "MonthlyCharges",
            xaxis = {"title" : "Churn?"},
            yaxis = {"title" : "MonthlyCharges"},
            width=800,
            height=500
        )

        fig = go.Figure(data=figs, layout=layout)
        py.offline.iplot(fig)
```

```
In [ ]: _ = df.groupby('Churn').size().reset_index()
        # .sort_values(by='tenure', ascending=True)

        data = [go.Bar(
            x = _['Churn'].tolist(),
            y = _[0].tolist(),
            marker=dict(
                color=['rgba(255,190,134,1)', 'rgba(142,186,217,1)'])
        )]
        layout = go.Layout(
            title = "Churn distribution",
            xaxis = {"title" : "Churn?"},
            width=800,
            height=500
        )
        fig = go.Figure(data=data, layout=layout)
        py.offline.iplot(fig)
```

```
In [ ]: df['class'] = df['Churn'].apply(lambda x : 1 if x == "Yes" else 0)
        # features will be saved as X and our target will be saved as y
        X = df[['tenure','MonthlyCharges']].copy()
        X2 = df[['tenure','MonthlyCharges']].copy()
        y = df['class'].copy()
```

In [ ]:
```
start_time = time.time()

num_iter = 100000

intercept = np.ones((X.shape[0], 1))
X = np.concatenate((intercept, X), axis=1)
theta = np.zeros(X.shape[1])

for i in range(num_iter):
    h = sigmoid(X, theta)
    gradient = gradient_descent(X, h, y)
    theta = update_weight_loss(theta, 0.1, gradient)

print("Training time (Log Reg using Gradient descent):" + str(time.time() - start_time) + " seconds")
print("Learning rate: {}\nIteration: {}".format(0.1, num_iter))
```

```
Training time (Log Reg using Gradient descent):70.8485119342804 seconds
Learning rate: 0.1
Iteration: 100000
```

In [ ]:
```
result = sigmoid(X, theta)
```

In [ ]:
```
f = pd.DataFrame(np.around(result, decimals=6)).join(y)
f['pred'] = f[0].apply(lambda x : 0 if x < 0.5 else 1)
print("Accuracy (Loss minimization):")
f.loc[f['pred']==f['class']].shape[0] / f.shape[0] * 100
```

```
Accuracy (Loss minimization):
```

Out[ ]:
```
53.301150078091716
```

In [ ]:
```
start_time = time.time()
num_iter = 100000

intercept2 = np.ones((X2.shape[0], 1))
X2 = np.concatenate((intercept2, X2), axis=1)
theta2 = np.zeros(X2.shape[1])

for i in range(num_iter):
    h2 = sigmoid(X2, theta2)
    gradient2 = gradient_ascent(X2, h2, y) #np.dot(X.T, (h - y)) / y.size
    theta2 = update_weight_mle(theta2, 0.1, gradient2)

print("Training time (Log Reg using MLE):" + str(time.time() - start_time) + "seconds")
print("Learning rate: {}\nIteration: {}".format(0.1, num_iter))
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(fit_intercept=True, max_iter=100000)
clf.fit(df[['tenure','MonthlyCharges']], y)
print("Training time (sklearn's LogisticRegression module):" + str(time.time() - start_time) + " seconds")
print("Learning rate: {}\nIteration: {}".format(0.1, num_iter))
```

```
Training time (sklearn's LogisticRegression module):83.02515387535095 seconds
Learning rate: 0.1
Iteration: 100000
```

In [ ]:
```
result3 = clf.predict(df[['tenure','MonthlyCharges']])
```

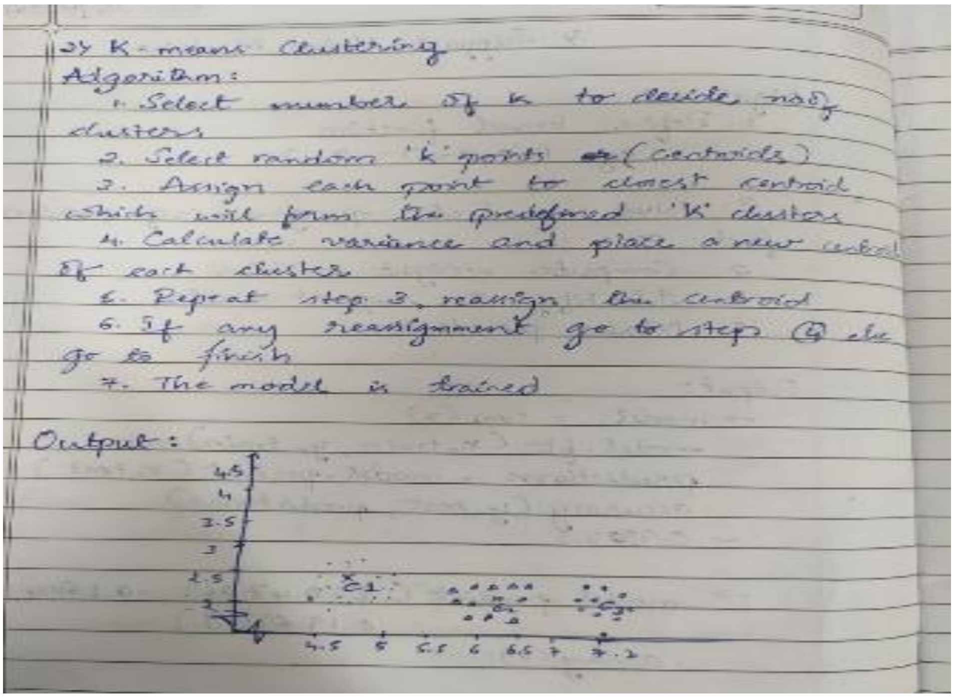
In [ ]:
```
print("Accuracy (sklearn's Logistic Regression):")
f3 = pd.DataFrame(result3).join(y)
f3.loc[f3[0]==f3['class']].shape[0] / f3.shape[0] * 100
```

```
Accuracy (sklearn's Logistic Regression):
```

Out[ ]:
```
78.44668465142695
```

Lab 5
Date : 24.05.2024
1. K Means Clustering Algorithm

Observation Screenshot :


2) K-means Clustering
Algorithm:
1. Select number of k to decide no. of clusters
2. Select random 'k' points (centroids)
3. Assign each point to closest centroid which will form the predefined 'K' clusters
4. Calculate variance and place a new centroid of each cluster
5. Repeat step 3, reassign the centroid
6. If any reassignment go to step (4) else go to finish
7. The model is trained

Output:

Code and Output:

In [ ]:
```
# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
```

In [ ]:
```
# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels _: Gives cluster no for which samples belongs
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will cha
nge from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

Out[ ]: KMeans(n_clusters=3)

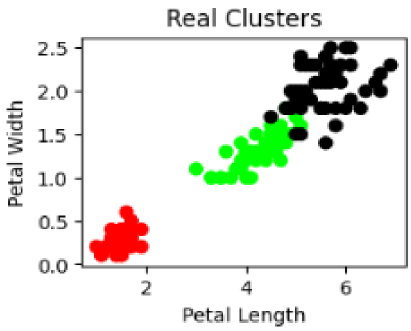
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
```

```
<Figure size 1400x1400 with 0 Axes>
```

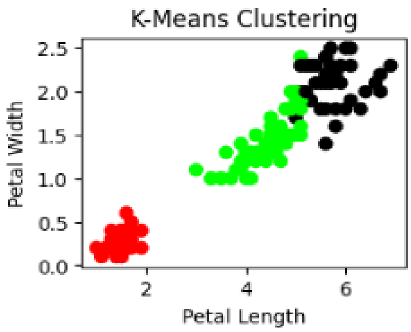
In [ ]:
```
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
Out[ ]: Text(0, 0.5, 'Petal Width')
```



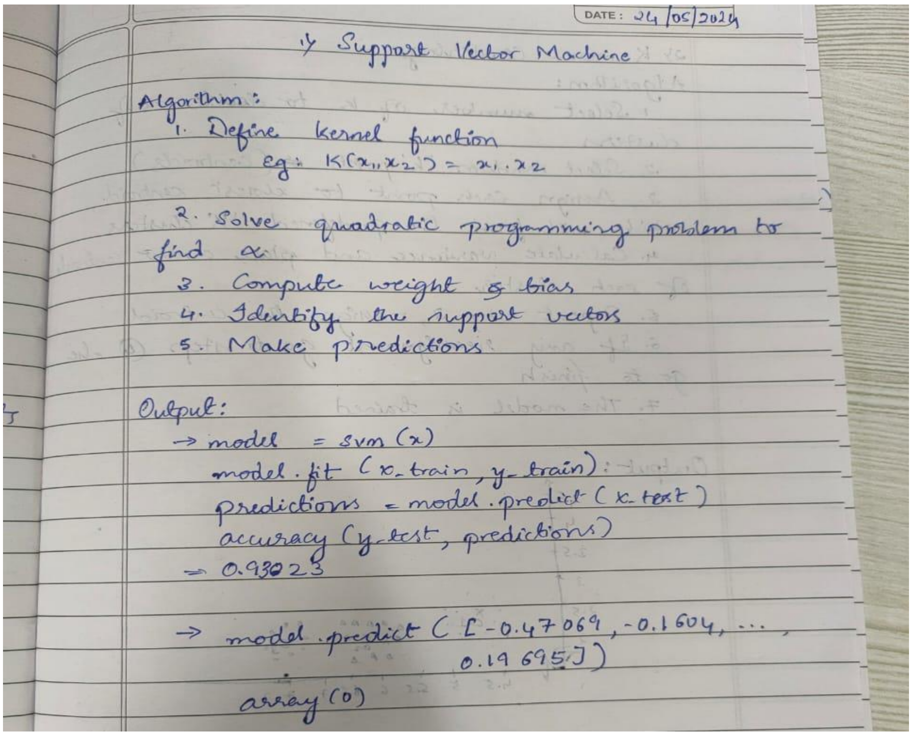

```
In [ ]:
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
Out[ ]: Text(0, 0.5, 'Petal Width')
```



2. Support Vector Machine

Observation Screenshot :


DATE : 24/05/2024

4) Support Vector Machine

Algorithm :
1. Define kernel function
eg: $K(x_1, x_2) = x_1 \cdot x_2$
2. Solve quadratic programming problem to find α
3. Compute weight & bias
4. Identify the support vectors
5. Make predictions

Output:
→ model = svm (x)
model.fit (x_train, y_train)
predictions = model.predict (x_test)
accuracy (y_test, predictions)
→ 0.93023

→ model.predict ([-0.47069, -0.1604, ..., 0.19695])
array (0)

Code and Output :

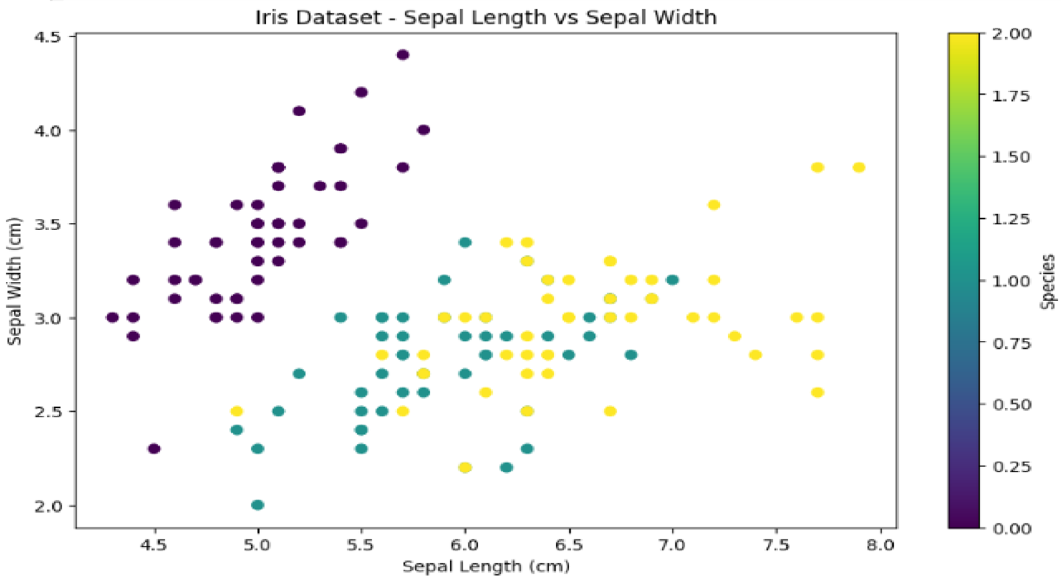
In [2]:
```python
# Load the Iris dataset
iris = load_iris()
```

In [3]:
```python
# Convert the dataset into a pandas DataFrame
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
```

In [4]:
```python
# Display the first few rows of the DataFrame
print(iris_df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

In [5]:
```python
# Plotting to visualize the data
plt.figure(figsize=(10, 6))
plt.scatter(iris_df['sepal length (cm)'], iris_df['sepal width (cm)'],
            c=iris_df['target'], cmap='viridis')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Iris Dataset - Sepal Length vs Sepal Width')
plt.colorbar(label='Species')
plt.show()
```



n [6]:
```python
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)
```

n [7]:
```python
# Creating and training the SVM classifier
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Predicting the labels for the test set
y_pred = svm_classifier.predict(X_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of SVM Classifier:", accuracy)
```

```
Accuracy of SVM Classifier: 1.0
```

n [8]:
```python
y_pred
```

ut[8]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0])
```

3.Principal Component Analysis

Observation Screenshot:

3) Principal Component Analysis:

Algorithm:

1. Calculate mean
2. Calculation of covariance matrix
3. Eigen values of covariance matrix
4. Computation of eigen-vector (unit eigen vector)
5. Computation of first principal components
6. Geometric meaning of first principal component.

Output:

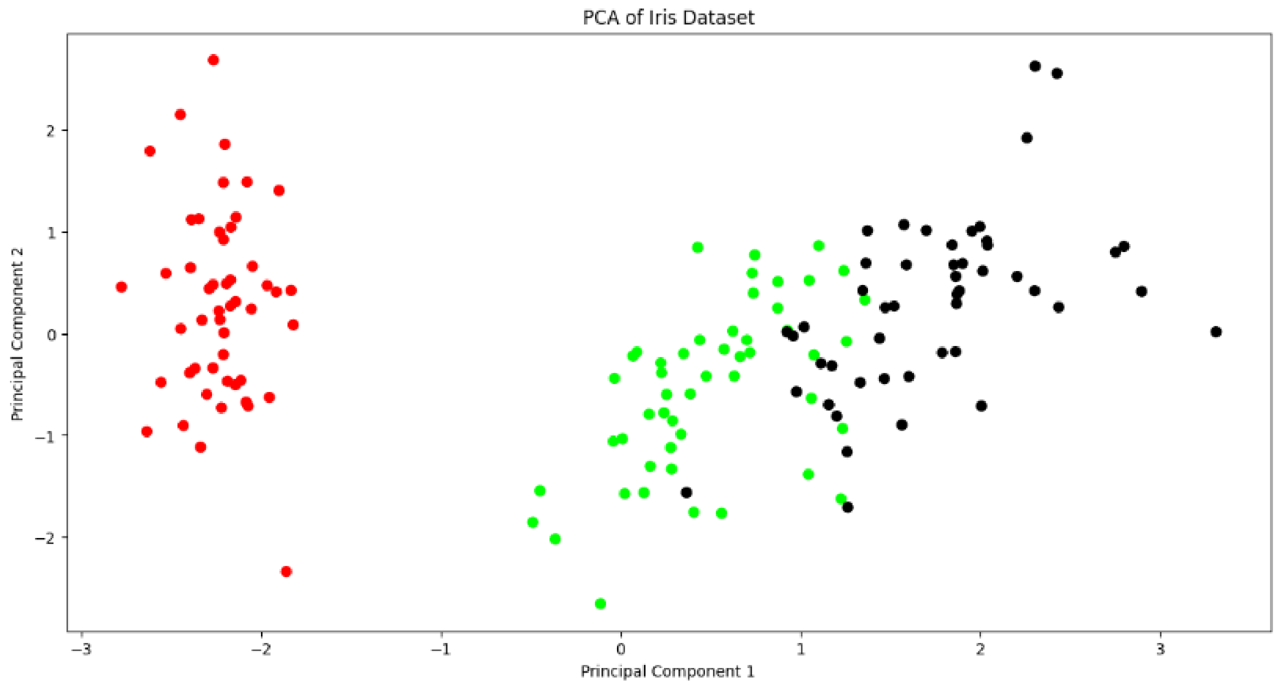
pca.explained_variance_ratio

array([0.98379448, 0.01620458])

Code and Output :

In [1]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Load the iris dataset
iris = datasets.load_iris()
X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(iris.target, columns=['Targets'])

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Convert PCA result to a DataFrame
X_pca_df = pd.DataFrame(X_pca, columns=['PCA1', 'PCA2'])

# Add the target column for visualization
X_pca_df['Targets'] = y.Targets

# Visualize the PCA result
plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])

# Plot the PCA transformed data
plt.scatter(X_pca_df.PCA1, X_pca_df.PCA2, c=colormap[X_pca_df.Targets], s=40)
plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



Lab – 6
Date : 31/05/2024
1.Build Artificial Neural Network model with back propagation on a given dataset.

Observation Screenshot :

* Artificial Neural Network with back propogation

Algorithm

1. Initialize parameters
   - normalise i/p features matrix 'x'
   - normalise o/p 'y'
   - set hyper parameters : 'no.of epochs', no.of neurons

2. Define activation func
   - Sigmoid function adjustments

3. Training network
   - forward propogation
     - compute i/p for hidden layer
     - Add bias
     - apply activation func

4. Backward propogation
   - comput error
   - compute gradient
   - compute delta

5. Update weights & biases

Output: I/p [[0.6667 1]
[0.333 0.556]
[0.1 0.667]]

Actual o/p → [[0.92] [0.86] [0.89]]

predicted o/p → [[0.80056875]
[0.79393831]
[0.8011234]]

Code and Output :

```python
import numpy as np
x = np.array(([2,9],[1,5],[3,6]),dtype = float)
y = np.array(([92],[86],[89]),dtype = float)
x = x/np.amax(x,axis=0)
y = y/100
```

```python
#Varialble Initialization
epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

```python
# weight and bias Initialization
wh = np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh = np.random.uniform(size=(1,hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout = np.random.uniform(size=(1,output_neurons))
```

```python
#sigmoid function
def sigmoid(x):
    return 1/(1+np.exp(-x))

# Derivative of Sigmoid
def der_sigmoid(x):
    return x*(1-x)
```

```python
# Draws a random range of numbers uniformly of dim x*y

for i in range(epoch):

    # forward propagation
    hinp1 = np.dot(x,wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act,wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output
    outgrad = der_sigmoid(output)
    d_output = EO*outgrad
    EH = d_output.dot(wout.T)
```

```python
    # how much hidden layer weights contributed to error
    hiddengrad = der_sigmoid(hlayer_act)
    d_hiddenlayer = EH*hiddengrad

    #dotproduct of nextlayererror and current layer op
    wout += hlayer_act.T.dot(d_output)*lr
    wh += x.T.dot(d_hiddenlayer)*lr

    print("Input: \n" + str(x))
    print("Actual output: \n" + str(y))
    print("Predicted Output: \n",output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.80056875]
 [0.79393831]
 [0.80112347]]
```

2. Implement Random forest ensemble method on a given dataset.

Observation Screenshot :

PAGE NO :
DATE :

* Random forest ensemble method

Algorithm
1. Import necessary libraries
2. Load & inspect data
3. preprocess data for training & testing samples. Use 0.4 to allocate 40% of data to testing & use 60% for training
4. Initialise Random forest classifier & train it using fit method
5. Make predictions on test sample using method predict
7. Evaluate the model

Output → Accuracy : 0.98
confusion matrix → [[23 0 0
0 19 0
0 1 17]]

Code and Output :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import datasets

# Load the data
iris_data = datasets.load_iris()

X = pd.DataFrame(iris_data.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(iris_data.target, columns=['Targets'])


# Check the info of the modified data
# print(iris_data.info())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to the training data
rf_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

✓ 0.7s

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       0.95      1.00      0.97        19
           2       1.00      0.94      0.97        18

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60

Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]
```

3. Implement Boosting ensemble method on a given dataset

Observation Screenshot :

DATE :

* Boosting ensemble method

Algorithm :

1. Import libraries
2. Load the dataset
3. Data preprocessing involves separation of features & dataset
4. Split dataset to train & test samples
5. Initialize adaboost classifier with specified no. of estimator & base estimator
6. Train model using training data
7. Make predictions for test sample using trainedg model
8. Evaluate model.

Output → matrix accuracy score : 0.9833

Code and Output :

```
[10] from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import AdaBoostClassifier
     from sklearn import metrics
     from sklearn import datasets

[11] import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split

[12] # Load the iris dataset
     iris = datasets.load_iris()
     X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
     y = pd.DataFrame(iris.target, columns=['Targets'])

[13] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=42)

[14] mylogregmodel = LogisticRegression()

[15] adabc = AdaBoostClassifier(n_estimators = 150, estimator = mylogregmodel, learning_rate = 1)

     model = adabc.fit(X_train, y_train)

     /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A
       y = column_or_1d(y, warn=True)

[17] y_pred = model.predict(X_test)

[18] metrics.accuracy_score(y_test, y_pred)

     0.9833333333333333
```