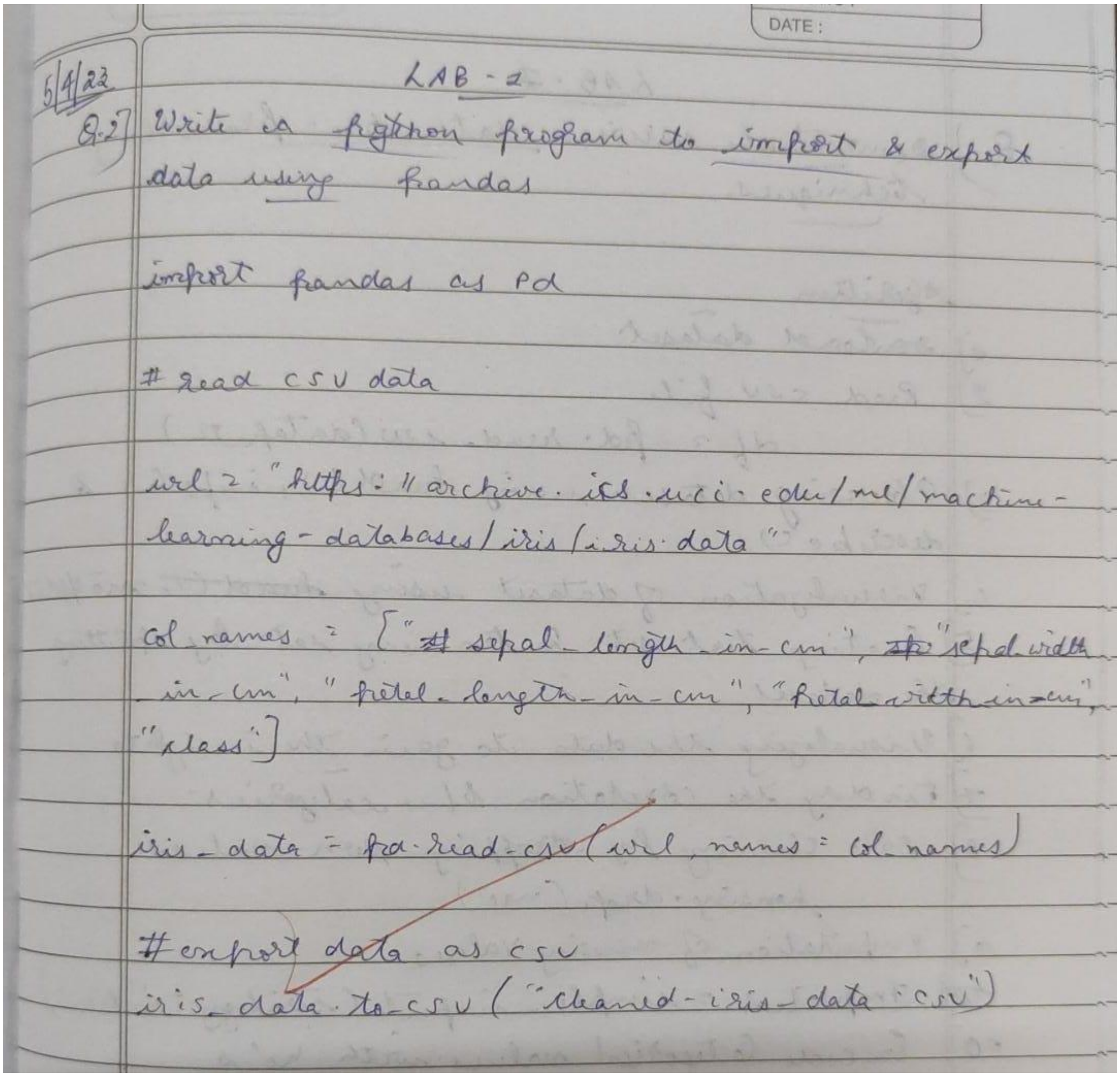
1) Write a python program to import and export data using Pandas library functions.

Importing data

Algorithm(Observation book)



DATE :

5/4/22

LAB - 1

Q.2] Write a python program to import & export data using pandas

import pandas as pd

read csv data

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

col_names = ["~~#~~ sepal_length_in_cm", ~~sto~~ "sepal_width_in_cm", "petal_length_in_cm", "petal_width_in_cm", "class"]

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

export data as csv

iris_data.to_csv("cleaned-iris_data.csv")

Code

```
import pandas as pd
df=pd.read_csv("/content/austinHousingData.csv")
df.head(5)
```

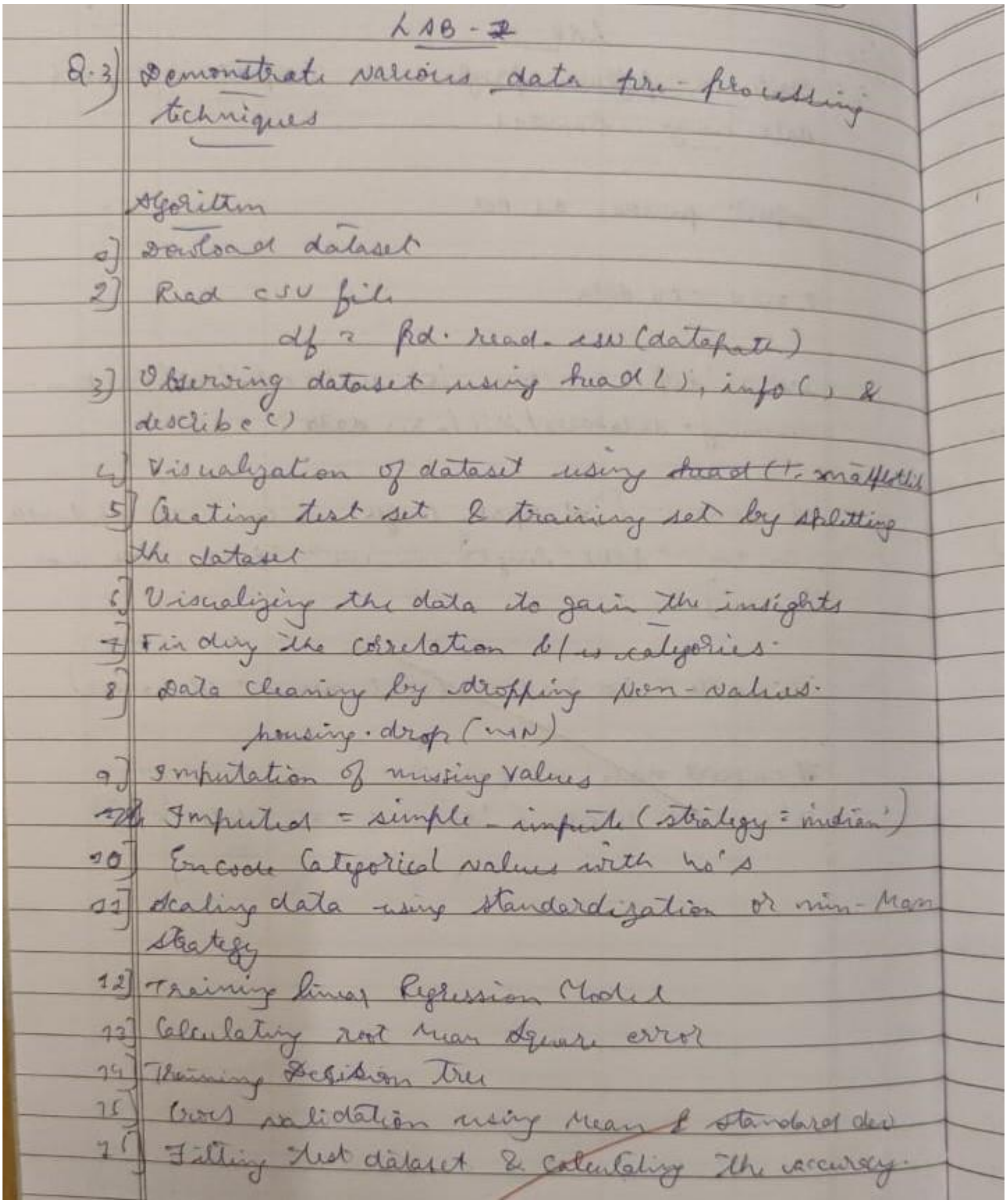
Output

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

5 rows × 47 columns

2) Demonstrate various data pre-processing techniques for a given dataset.

Algorithm(Observation book)



LAB - 2

Q.3) Demonstrate various data pre-processing techniques

Algorithm

1) Download dataset
2) Read csv file
   df = pd.read_csv(datapath)
3) Observing dataset using head(), info() & describe()
4) Visualization of dataset using ~~head()~~ matplotlib
5) Creating test set & training set by splitting the dataset
6) Visualizing the data to gain the insights
7) Finding the correlation b/w categories.
8) Data cleaning by dropping Non-values.
   housing.drop(NaN)
9) Imputation of missing values
   Imputed = simple_imputer(strategy='median')
10) Encode Categorical values with no's
11) Scaling data using standardization or min-Max strategy
12) Training linear Regression Model
13) Calculating root mean square error
14) Training Decision Tree
15) Cross validation using Mean & standard dev
16) Fitting test dataset & calculating the accuracy.

Code

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

```
# Define the column names
col_names = ["sepal_length_in_cm",
             "sepal_width_in_cm",
             "petal_length_in_cm",
             "petal_width_in_cm",
             "class"]
# Read data from URL
iris_data = pd.read_csv(url, names=col_names)
iris_data.head(5)
iris_data.to_csv("/content/exported_irisData.csv")
```

Output:

	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

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
df1=pd.read_csv("/content/Data.csv")
df1.head(5)
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

```
#Identifying and handling the missing values
df1.isnull().sum()
```

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

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

Algorithm(Observation book)

Lab 2
12/4/24

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

Algorithm for decision tree

```
def ID3(D, A):
    if D is pure or A is empty:
        return a leaf node with majority class
in D
    else:
        A_best = argmax(Information Gain(D, A))
        root = Node(A_best)
        for v in values(A_best):
            D_v = subset(D, A_best, v)
            child = ID3(D_v, A-{A_best})
            root.add_child(v, child)
        return root.
```

Entropy = $-P_i \log_2 (P_i)$

P_i -> fraction of sample within a particular node

$IG(A, D) = H(S) - \sum_v \frac{|S_v|}{|S|} * H(S_v)$

S -> Total instances

S_v -> no' of instances for which attribute D has value v.

Code

```
# Importing the required libraries
import pandas as pd
import numpy as np
import math
data = pd.read_csv('/content/PlayTennis.csv')
```

```
def highlight(cell_value):
    '''
    Highlight yes / no values in the dataframe
    '''
    color_1 = 'background-color: pink;'
    color_2 = 'background-color: lightgreen;'

    if cell_value == 'no':
        return color_1
    elif cell_value == 'yes':
        return color_2

data.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-
color', 'lightgray'), ('border', '1px solid gray'),
                                                    ('font-weight',
'bold')]},
     {'selector': 'tr:hover', 'props': [('background-color', 'white'),
('border', '1.5px solid black')]}])
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

```
def find_entropy(data):
    """
    Returns the entropy of the class or features
    formula: - ∑ P(X)logP(X)
    """
    entropy = 0
    for i in range(data.nunique()):
        x = data.value_counts()[i]/data.shape[0]
        entropy += (- x * math.log(x,2))
    return round(entropy,3)

def information_gain(data, data_):
    """
    Returns the information gain of the features
    """
    info = 0
    for i in range(data_.nunique()):
        df = data[data_ == data_.unique()[i]]
        w_avg = df.shape[0]/data.shape[0]
        entropy = find_entropy(df.play)
        x = w_avg * entropy
        info += x
    ig = find_entropy(data.play) - info
    return round(ig, 3)

def entropy_and_infogain(datax, feature):
    """
    Grouping features with the same class and computing their
    entropy and information gain for splitting
    """
    for i in range(data[feature].nunique()):
        df = datax[datax[feature]==data[feature].unique()[i]]
        if df.shape[0] < 1:
            continue

        display(df[[feature, 'play']].style.applymap(highlight)\
                .set_properties(subset=[feature, 'play'], **{'width':
'80px'})\
                .set_table_styles([{'selector': 'th', 'props':
[('background-color', 'lightgray'),
                                                                ('borde
r', '1px solid gray'),
                                                                ('font-
weight', 'bold')]},
                                  {'selector': 'td', 'props':
[('border', '1px solid gray')]},
                                  {'selector': 'tr:hover', 'props':
[('background-color', 'white'),
```

```
                                                                                (
'border', '1.5px solid black')]}]))

        print(f'Entropy of {feature} - {data[feature].unique()[i]} =
{find_entropy(df.play)}')
    print(f'Information Gain for {feature} = {information_gain(datax,
datax[feature])}')
```

```
#Computing entropy for the entire dataset
print(f'Entropy of the entire dataset: {find_entropy(data.play)}')
```

```
Entropy of the entire dataset: 0.94
```

```
#Calculate the Information Gain for each feature.
#Outlook
entropy_and_infogain(data, 'outlook')
```

	outlook	play
0	sunny	no
1	sunny	no
7	sunny	no
8	sunny	yes
10	sunny	yes

```
Entropy of outlook - sunny = 0.971
```

	outlook	play
2	overcast	yes
6	overcast	yes
11	overcast	yes
12	overcast	yes

```
Entropy of outlook - overcast = 0.0
```

	outlook	play
3	rainy	yes
4	rainy	yes
5	rainy	no
9	rainy	yes
13	rainy	no

```
Entropy of outlook - rainy = 0.971
Information Gain for outlook = 0.246
```

```python
#Temp
entropy_and_infogain(data, 'temp')
```

	temp	play
0	hot	no
1	hot	no
2	hot	yes
12	hot	yes

```
Entropy of temp - hot = 1.0
```

	temp	play
3	mild	yes
7	mild	no
9	mild	yes
10	mild	yes
11	mild	yes
13	mild	no

```
Entropy of temp - mild = 0.918
```

	temp	play
4	cool	yes
5	cool	no
6	cool	yes
8	cool	yes

```
Entropy of temp - cool = 0.811
Information Gain for temp = 0.029
```

```python
#Humidity
entropy_and_infogain(data, 'humidity')
```

	humidity	play
0	high	no
1	high	no
2	high	yes
3	high	yes
7	high	no
11	high	yes
13	high	no

```
Entropy of humidity - high = 0.985
```

	humidity	play
4	normal	yes
5	normal	no
6	normal	yes
8	normal	yes
9	normal	yes
10	normal	yes
12	normal	yes

```
Entropy of humidity - normal = 0.592
Information Gain for humidity = 0.151
```

```
#Windy
entropy_and_infogain(data, 'windy')
```

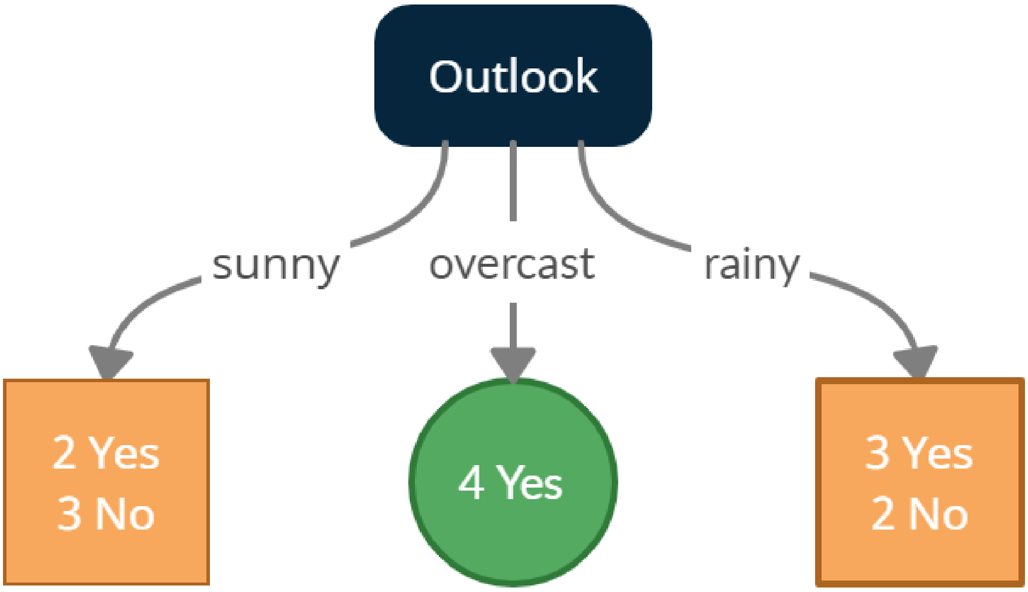
	windy	play
0	False	no
2	False	yes
3	False	yes
4	False	yes
7	False	no
8	False	yes
9	False	yes
12	False	yes

```
Entropy of windy - False = 0.811
```

	windy	play
1	True	no
5	True	no
6	True	yes
10	True	yes
11	True	yes
13	True	no

```
Entropy of windy - True = 1.0
Information Gain for windy = 0.048
```

#Make a decision tree node using the feature with the maximum Information Gain.



```python
sunny = data[data['outlook'] == 'sunny']
sunny.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                                     ('font-weight', 'bold')]},
     {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]}])
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
10	sunny	mild	normal	True	yes

```python
print(f'Entropy of the Sunny dataset: {find_entropy(sunny.play)}')
```

```
Entropy of the Sunny dataset: 0.971
```

```python
#temp
entropy_and_infogain(sunny, 'temp')
```

	temp	play
0	hot	no
1	hot	no

```
Entropy of temp - hot = 0.0
```

	temp	play
7	mild	no
10	mild	yes

```
Entropy of temp - mild = 1.0
```

	temp	play
8	cool	yes

```
Entropy of temp - cool = 0.0
Information Gain for temp = 0.571
```

```python
#Humidity
entropy_and_infogain(sunny, 'humidity')
```

	humidity	play
0	high	no
1	high	no
7	high	no

```
Entropy of humidity - high = 0.0
```

	humidity	play
8	normal	yes
10	normal	yes

```
Entropy of humidity - normal = 0.0
Information Gain for humidity = 0.971
```

```python
#Windy
entropy_and_infogain(sunny, 'windy')
```

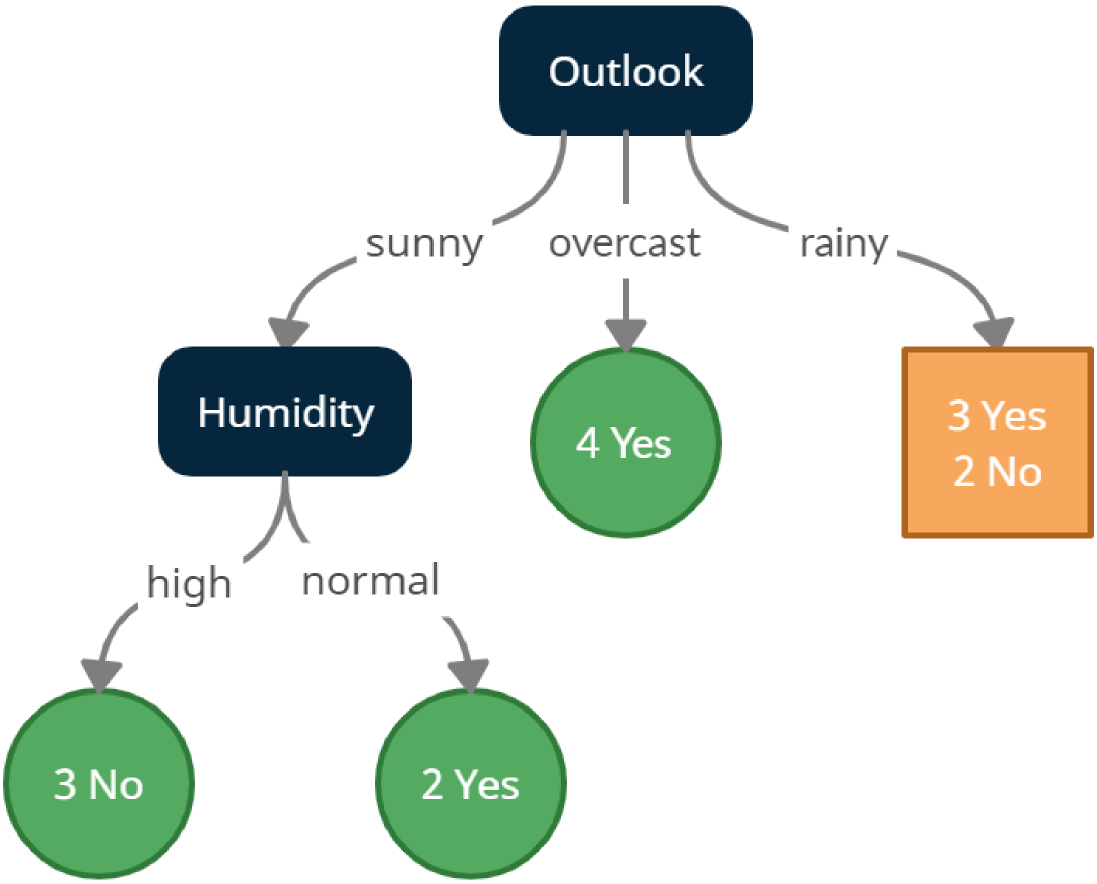
	windy	play
0	False	no
7	False	no
8	False	yes

```
Entropy of windy - False = 0.918
```

	windy	play
1	True	no
10	True	yes

```
Entropy of windy - True = 1.0
Information Gain for windy = 0.02
```

#Making a decision tree node using the feature which has the maximum Information Gain




```
#Outlook - Rainy
rainy = data[data['outlook'] == 'rainy']
rainy.style.applymap(highlight)\
    .set_properties(subset=data.columns, **{'width': '100px'})\
    .set_table_styles([{'selector': 'th', 'props': [('background-
color', 'lightgray'), ('border', '1px solid gray'),
                                                     ('font-weight',
'bold')]},
     {'selector': 'tr:hover', 'props': [('background-color', 'white'),
('border', '1.5px solid black')]}])
```

	outlook	temp	humidity	windy	play
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
9	rainy	mild	normal	False	yes
13	rainy	mild	high	True	no

```
print(f'Entropy of the Rainy dataset: {find_entropy(rainy.play)}')
```

```
Entropy of the Rainy dataset: 0.971
```

```
#temp
entropy_and_infogain(rainy, 'temp')
```

	temp	play
3	mild	yes
9	mild	yes
13	mild	no

```
Entropy of temp - mild = 0.918
```

	temp	play
4	cool	yes
5	cool	no

```
Entropy of temp - cool = 1.0
Information Gain for temp = 0.02
```

```
#Humidity
entropy_and_infogain(rainy, 'humidity')
```

	humidity	play
3	high	yes
13	high	no

```
Entropy of humidity - high = 1.0
```

	humidity	play
4	normal	yes
5	normal	no
9	normal	yes

```
Entropy of humidity - normal = 0.918
Information Gain for humidity = 0.02
```

```
#Windy
entropy_and_infogain(rainy, 'windy')
```

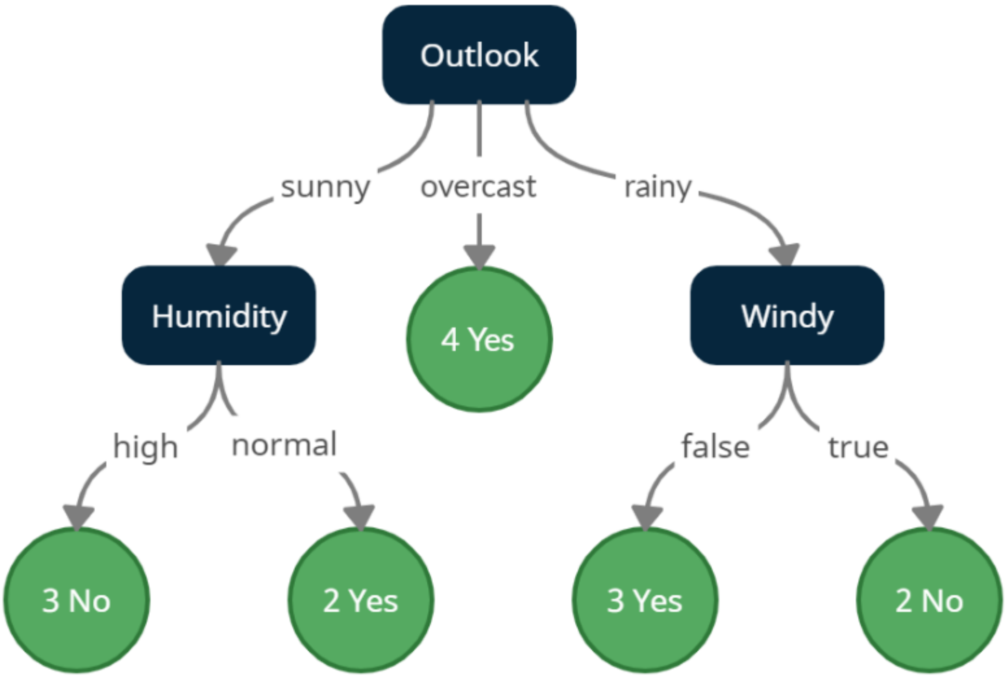
	windy	play
3	False	yes
4	False	yes
9	False	yes

```
Entropy of windy - False = 0.0
```

	windy	play
5	True	no
13	True	no

```
Entropy of windy - True = 0.0
Information Gain for windy = 0.971
```

#Making a decision tree node using the feature which has the maximum Information Gain.



4) Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Algorithm

DATE :

Lab 3
3/5/24

Implement Linear Regression algorithm using appropriate dataset (& Multi Linear)

Linear :-

i] Import necessary libraries

ii] Import Dataset

iii] Visualization of dataset using different plots like heatmap, distribution plot etc.

iv] Preprocess the data, convert or encode categorical data.

v] Split the dataset into training & testing set from sklearn. model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split (x, y, test_size = 0.3, random_state = 2.3)

vi] Build model

from sklearn. linear_model import LinearRegression
lin_reg = LinearRegression ()

vii] Fit dataset to model & train it lin_reg.fit (x_train, Ytrain)

viii] Calculate the accuracy using Mean Square error

Multi Linear

i - iii) same

iv] Encode Categorical Data

~~v]~~ ct = ColumnTransformer (transformers = [('encode', OneHotEncoder [ ] [3])], remainder = 'passthrough')

v] Split dataset into training & testing set.

vi] We can see multiple independent variables.

vii] Create Regression model

regressor = LinearRegression ( )

viii] Fit train set.

ix] Test model using test set.

Code:

```
#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
```

```python
# Reading the Dataset
df = pd.read_csv("data.csv")
```

```python
df.head()
```

	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

```python
df.shape
```

```
(36, 5)
```

```python
df.corr(numeric_only=True)
```

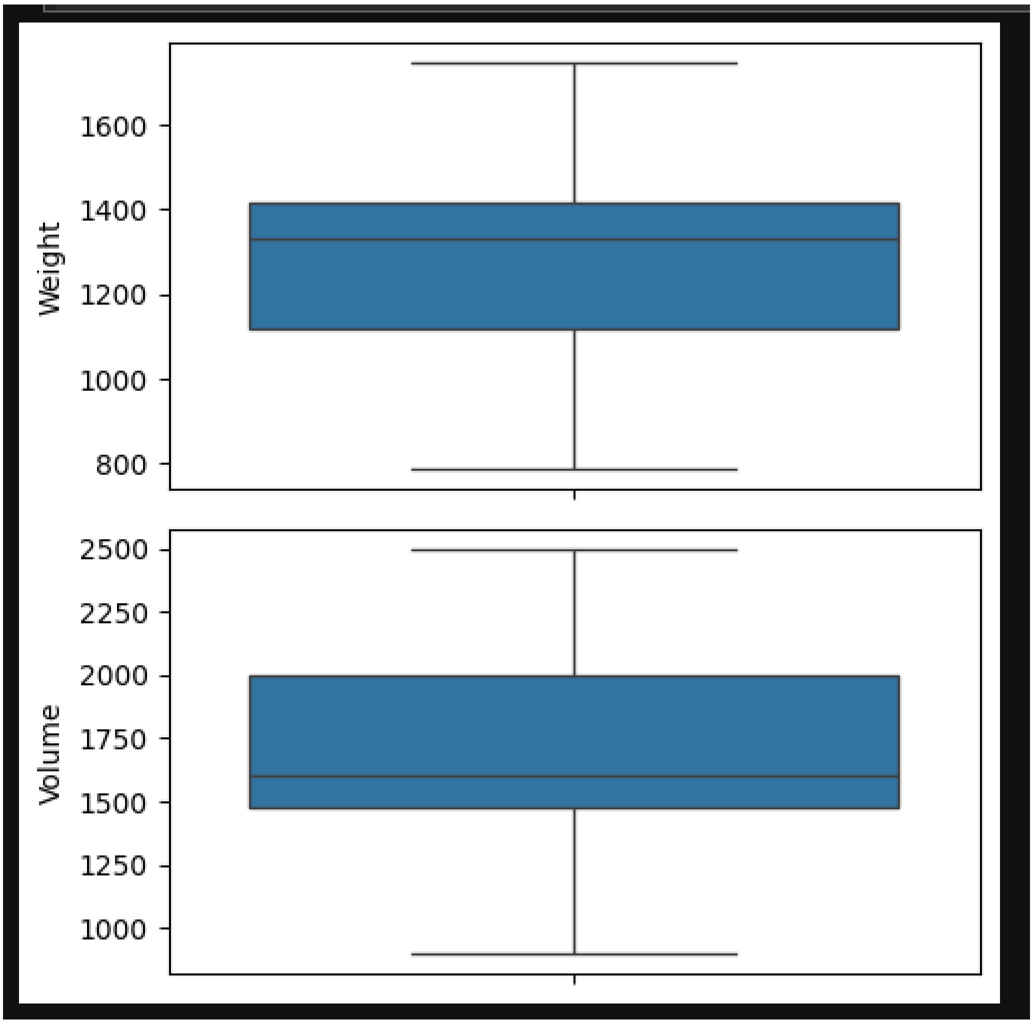
	Volume	Weight	CO2
Volume	1.000000	0.753537	0.592082
Weight	0.753537	1.000000	0.552150
CO2	0.592082	0.552150	1.000000

In [ ]:
```python
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
```

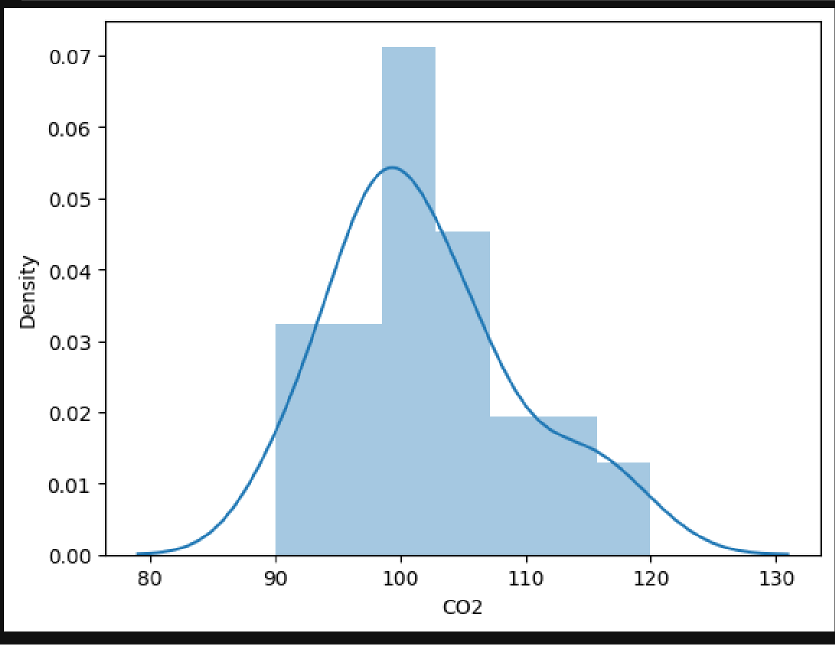
In [ ]:
```python
#Setting the value for X and Y
X = df[['Weight', 'Volume']]
y = df['CO2']
```

In [ ]:
```python
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()
```



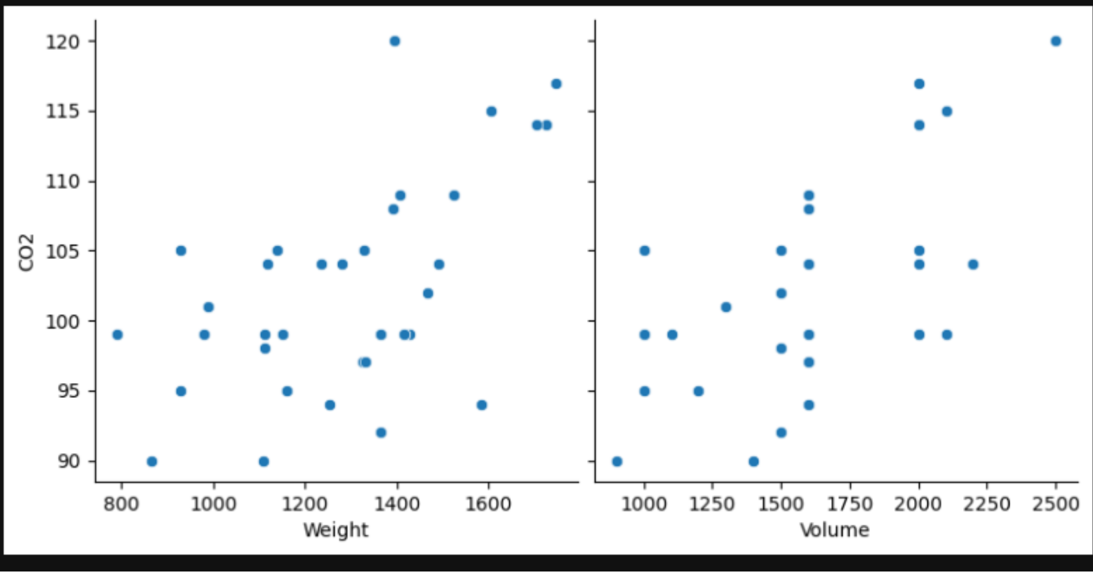
In [ ]:

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



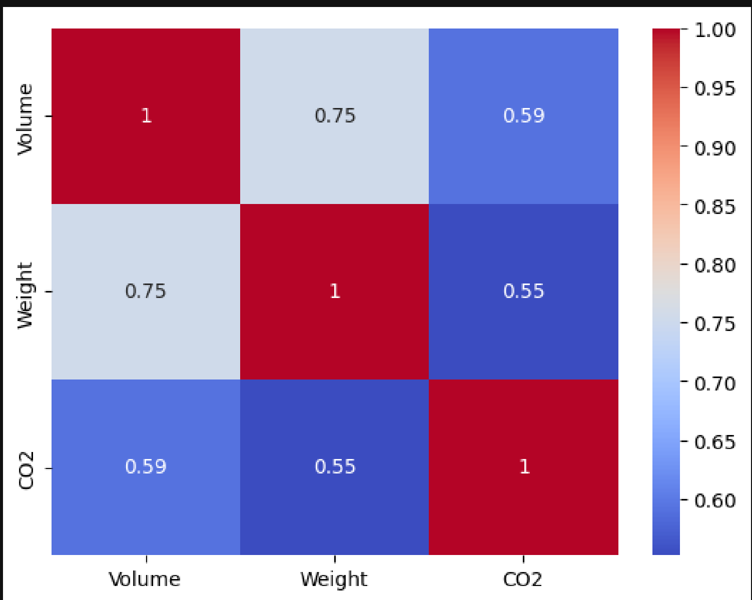
In [ ]:

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



In [ ]:

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



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

In [ ]: y_train.shape

Out[ ]: (25,)

In [ ]: y_test.shape

Out[ ]: (11,)

In [ ]: reg_model = linear_model.LinearRegression()

In [ ]: #Fitting the Multiple Linear Regression model
        reg_model = LinearRegression().fit(X_train, y_train)

In [ ]: #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[ ]: [('Weight', 0.0171800645996374), ('Volume', 0.0025046399866402976)]

In [ ]: #Predicting the Test and Train set result
        y_pred= reg_model.predict(X_test)
        x_pred= reg_model.predict(X_train)

In [ ]: 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 [ ]: #Actual value and the predicted value
        reg_model_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred})
        reg_model_diff
```

Out[ ]:

	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 [ ]:

```
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
```

Results

```
Mean Absolute Error: 6.901980901636316
Mean Square Error: 63.39765310998794
Root Mean Square Error: 7.96226432053018
```

5) Build KNN Classification model for a given dataset.

Algorithm

PAGE NO :
DATE :

3/5/24 Build KNN Classification model for a given dataset

Steps:

* Choose K: Determine No. of Neighbours & K
* Calculate Distance: Compute the distance b/w the new data & all points in dataset
* We can use some common distance metrics like Euclidean distance, Manhattan dist etc
* Find nearest neighbours: Select K points in dataset that are closest to the new data point based on calculated distances
* Majority vote: For classification, count the no. of data points in each category among the K nearest neighbours. Assign the new data point to the class that is most common among its K nearest neighbours.

Code:

```
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)
```

```
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))
```

Results:

```
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
[7.  3.2 4.7 1.4] - 1
[6.2 2.8 4.8 1.8] - 2
[5.7 2.8 4.5 1.3] - 1
[6.  3.4 4.5 1.6] - 1
[5.1 3.8 1.6 0.2] - 0
[5.5 2.5 4.  1.3] - 1
[4.8 3.1 1.6 0.2] - 0
[6.1 3.  4.9 1.8] - 2
[4.7 3.2 1.6 0.2] - 0
[5.6 2.9 3.6 1.3] - 1
[5.4 3.9 1.3 0.4] - 0
[5.  3.2 1.2 0.2] - 0
[6.1 2.9 4.7 1.4] - 1
[5.  3.4 1.5 0.2] - 0
[7.7 2.8 6.7 2.  ] - 2
[4.6 3.2 1.4 0.2] - 0
[5.7 2.9 4.2 1.3] - 1
[4.6 3.6 1.  0.2] - 0
[6.8 2.8 4.8 1.4] - 1
[6.8 3.2 5.9 2.3] - 2
```

6) Build Logistic Regression Model for a given dataset

Algorithm:

PAGE NO :
DATE :

3/5/24 Build Logistic Regression model for a given Dataset

i] Import all required libraries
ii] Import given dataset
iii] Preprocess dataset to standard scale
iv] Split dataset into test & train
v] Code Logistic Regression model
lr = LogisticRegression(C=0.01, solver='liblinear')
fit(x_train, y_train)
vi] Predict test set using model
yhat = lr.predict(x_test)
yhat_probab = lr.predict_probab(x_test)

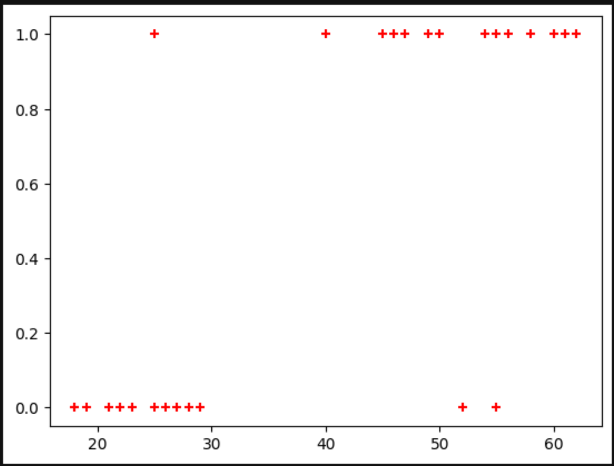
vii] Calculate the performance & accuracy of the model.

Code:

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

In [4]:
```
df = pd.read_csv("insurance_data.csv")
df.head()
plt.scatter(df.age,df.bought_insurance,marker ='+',color ='red')
```

Out[4]: <matplotlib.collections.PathCollection at 0x7e3b7e4c9cf0>



In [31]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.bought_insurance,train_size=0.7)
print(X_test)
```

```
    age
12   27
13   29
4    46
23   45
3    52
8    62
1    25
11   28
25   54
```

In [32]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
print(X_test)
y_predicted = model.predict(X_test)
probab = model.predict_proba(X_test)
score = model.score(X_test,y_test)
print(y_predicted)
print("\nprobability: ")
print(probab)
print("\naccuracy: ")
print(score)
```

```
    age
12   27
13   29
4    46
23   45
3    52
8    62
1    25
11   28
25   54
[0 0 1 1 1 1 0 0 1]
```

```
probability:
[[0.75147045 0.24852955]
 [0.69990447 0.30009553]
 [0.20424226 0.79575774]
 [0.2261509  0.7738491 ]
 [0.10537592 0.89462408]
 [0.03115876 0.96884124]
 [0.79674889 0.20325111]
 [0.7264443  0.2735557 ]
 [0.08328741 0.91671259]]

accuracy:
0.8888888888888888
```

In [33]:
```
print(X_test)
```

```
    age
12   27
13   29
4    46
23   45
3    52
8    62
1    25
11   28
25   54
```

In [38]:
```
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))
```

In [39]:
```
def prediction_function(age):
  z = 0.042 * age - 1.53 # 0.04150133 ~ 0.042 and -1.52726963 ~ -1.53
  y = sigmoid(z)
  return y
```

In [40]:
```
def insure(probability):
  if (probability > 0.5):
    print("The customer will get insurance.")
  else:
    print("The customer will not get insurance.")

age = 35
probability = prediction_function(age)
print("Probability of buying insurance for age 35:", probability)
insure(probability)
```

```
Probability of buying insurance for age 35: 0.4850044983805899
The customer will not get insurance.
```

In [41]:
```
age = 43
probability = prediction_function(age)
print("Probability of buying insurance for age 43:", probability)
insure(probability)
```

```
Probability of buying insurance for age 43: 0.568565299077705
The customer will get insurance.
```

Results:

```
Probability of buying insurance for age 35: 0.4850044983805899
The customer will not get insurance.

In [41]:  age = 43
          probability = prediction_function(age)
          print("Probability of buying insurance for age 43:", probability)
          insure(probability)

Probability of buying insurance for age 43: 0.568565299077705
The customer will get insurance.
```

7) Build Support vector machine model for a given dataset

Algorithm:

PAGE NO :
DATE :

24/5/24 Build Support vector machine model for a given dataset

Algorithm

1] ~~Crd~~ Load the dataset to be trained [Iris Dataset]

2] Convert the dataset into Dataframe (pandas). Add the target column to the Dataframe.

3] Print the first few rows of DataFrame to inspect the Data. Create a scatter plot to visualize the relationship b/w sepal length & sepal width

4] Split the dataset into training & testing sets. Use 70% of data for training & 30% for testing

5] Create and train the SVM classifier
- Initialize an SVM classifier ~~or~~ with a linear kernel
- Train the classifier using training data

6] Predict Test Labels.
- Use trained classifier to predict labels for the test set.

7] Evaluate the model
- Calculate accuracy of model by comparing predicted labels with actual test labels.

Code:

```
In [1]:
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

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

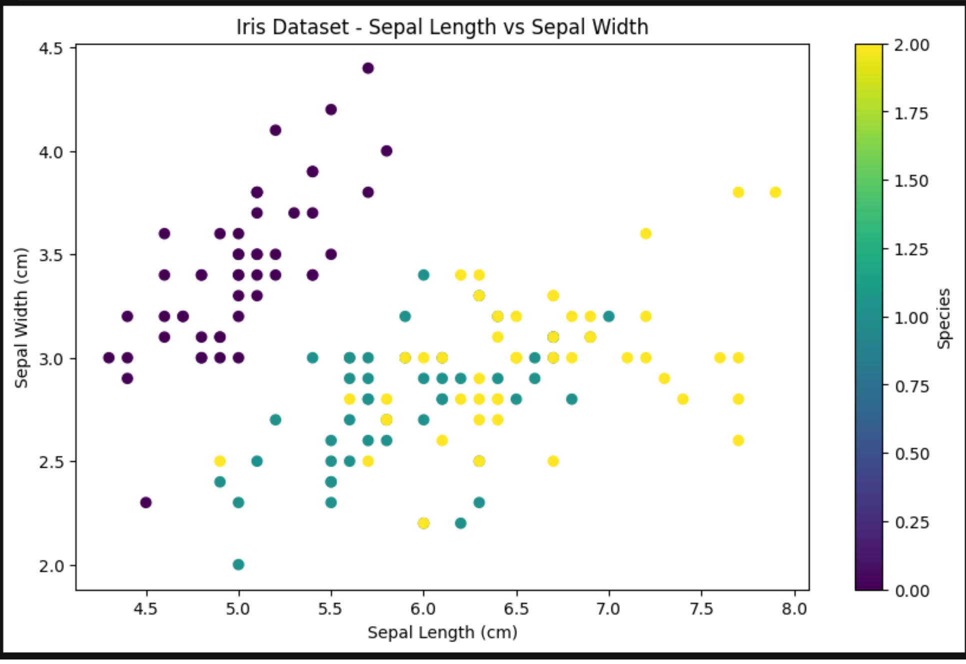
```
In [3]:
# Convert the dataset into a pandas DataFrame
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
```

```
In [4]:
# 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()
```



In [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)
```

In [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)
```

Results:

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

In [8]:

```python
y_pred
```

Out[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])
```

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

Algorithm:

PAGE NO :
DATE :

24/5/23 Build K-Means algorithm to cluster a set of data ~~stored~~ ~~in~~

Algorithm

1] Load the iris Dataset
- Import iris dataset
- Store dataset's features in a dataframe
- Store the dataset's target values as well

2] Initialize and fit K-Means model
- Create an instance of K-Means model with cluster = 3
- Fit model to feature data x)

3] Visualize the Clustering results.
- Create ~~instance~~ a new figure for plotting with size 14x14
- Define color map 'colormap' with values ['red', 'lime', 'black'] for visualization purposes.

4] Plot original classifications.
- Create subplot
- Plot original classifications using petal length & width as features

5] Plot K-Means Clustering results
- Create another subplot
- Plot the clustering results from K-Means model using petal length & width as features.

Code:

```python
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
```

```python
# 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']
```

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

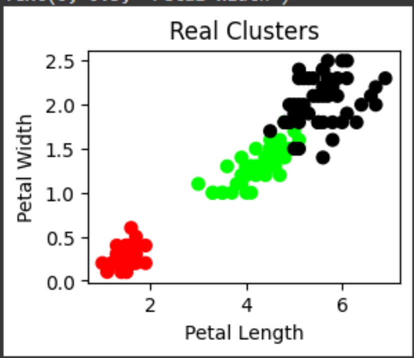
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
KMeans
KMeans(n_clusters=3)
```

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

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

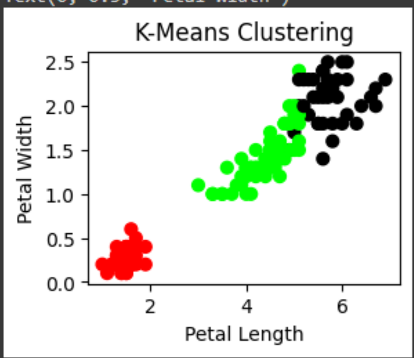
```python
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
Text(0, 0.5, 'Petal Width')
```




```python
# 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')
```

```
Text(0, 0.5, 'Petal Width')
```



9) Implement Dimensionality reduction using Principle Component Analysis (PCA)
method.

Algorithm

PAGE NO :
DATE :

24/5/24 Implement Dimensionality Reduction using Principle Component Analysis method

Algorithm

1] Import necessary libraries for data handling, standardization, PCA & plotting

2] Load iris Dataset
- Load the iris dataset.
- Store dataset's features in Dataframe X ~~ro~~ & store dataset's ~~ft~~ target values in dataframe y.

3] Standardize the data
- Initialize a 'Standard Scaler' to standardize the feature data.
- Fit ~~to~~ & transform feature data 'X' using scaler, storing result in 'X_scaled'

4] Apply PCA
- Initialize PCA model with n = 2 to reduce data to 2 dimensions.
- Fit & transform ~~as~~ standardized data 'X_scaled' using PCA model storing result in 'X_PCA'

5] Convert PCA result to Dataframe
- Convert the PCA transformed data 'X_PCA' into a DataFrame 'XPCA_df' with columns ['PCA1', 'PCA2']
- Add the target column from 'y' to 'X_PCA_df' for visualization purposes.

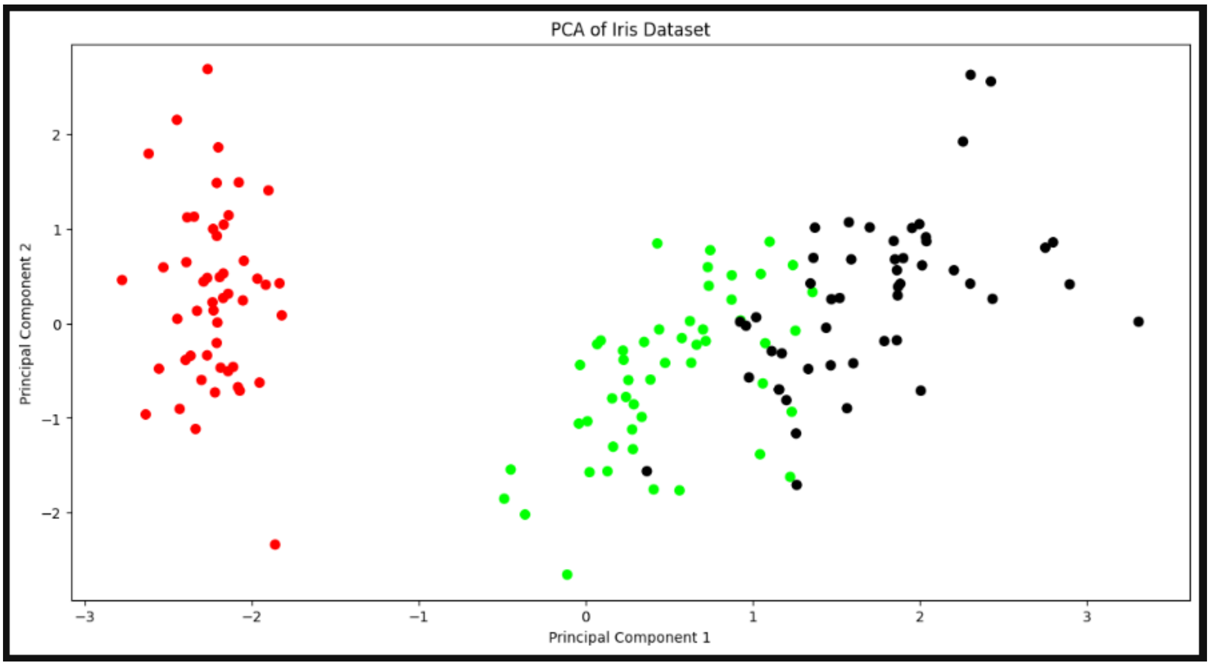
DATE :

6] Visualize the PCA result
- Create a new figure for plotting
- Define a color map array 'colormap' with values ['red', 'lime', 'black']
- Create scatter plot of PCA - transformed data using principal components.
- Color points based on original class labels from 'y'.

Code:

```
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()
```

Results:



10) Build Artificial Neural Network model with back propagation on a given dataset

Algorithm

PAGE NO :
DATE :

31/5/24 Build Artificial Neural Network model with back propagation

Algorithm

1] Initialize parameters
- Normalize i/p feature matrix 'x'
- Normalize the output 'y'
- Set hyperparameters: no' of epochs, lr, no' of neurons in i/p layer, hidden layer & o/p layer
- Initialize weights & biases for hidden layer & o/p layer with random values.

2] Define Activation Functions
- Sigmoid f^n $\sigma(x) = 1/1+e^{-x}$
- Derivative of sigmoid => $\sigma'(x) = \sigma(x) \cdot (1-\sigma(x))$

3] Training the Network

Forward Propagation
* Compute i/p to hidden layer: hinp1 = x · wh
* Add bias to hidden layer i/p => hinp = hinp1 + bh
* Apply activation function: hlayer_act = σ(hinp)
* Compute ~~activation function~~ i/p to o/p layer: outinp1 = hlayer_act · wout
* Add bias to output layer input: outinp = outinp1 + bout
* Apply activation f^n => o/p = σ(outinp)

Backpropagation
* Compute error at o/p layer: EO = y - output
* Compute gradient at o/p layer: outgrad = σ'(output)
* Compute delta for o/P layer: d_output = EO · outgrad
* compute error at hidden layer: EH = d_output · $wout^T$
* Compute gradient at hidden layer: hiddengrad = σ'(hlayer_act)
* Compute delta for hidden layer: d_hiddenlayer = EH · hiddengrad

31/5

Update wts & biases
* Update wts for o/p layer: wout += $hlayer\_act^T$ · doutput · lr
* Update wts for hidden layer: wh += x^T · d_hiddenlayer · lr

Code:

```
In [10]:
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
```

```
In [11]:
#Varialble Initialization
epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

```
In [12]:
# 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))
```

```
In [13]:
#sigmoid function

def sigmoid(x):
  return 1/(1+np.exp(-x))

# Derivative of Sigmoid

def der_sigmoid(x):
  return x*(1-x)
```

```
In [14]:
# 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)
```

```
In [15]:
# 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)
```

Results

```
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]]
```

11) Implement Random forest ensemble method on a given dataset.

Algorithm

21/5/24 Implement Random Forest Ensemble method on

Algorithm

1] Import necessary Libraries

2] Load and inspect Data

3] Preprocess the data as in seperating features & target

4] Split the data to training and test samples. Use 0.4 to allocate 40% of data to testing & use rest for training.

5] Initialize Random Forest Classifier and train it using 'fit' method

6] Make predictions on test sample using method 'predict'

7] Evaluate the model.

Code:

```python
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))
```

Results:

```
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]]
```

12) Implement Boosting ensemble method on a given dataset.

Algorithm

DATE :

31/5/24 Implement Boosting ensemble method

1] Import libraries

2] Load the dataset

3] Data preprocessing involving seperation of features & target

4] Split the dataset to train test ~~&~~ samples.

5] Initialize the adaboost classifier with specified no. of estimators & base estimator (Logistic Regression)

6] Train the model using the training data

7] Make predictions for test sample using trained model.

8] Evaluate the model.

Code:

```
In [4]:
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
from sklearn import datasets

In [5]:
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

In [6]:
# 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'])

In [7]:
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=42)

In [9]:
mylogregmodel = LogisticRegression()

In [10]:
adabc = AdaBoostClassifier(n_estimators = 150, base_estimator = mylogregmodel, learning_rate = 1)

In [11]:
model = adabc.fit(X_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estima
tor` in version 1.2 and will be removed in 1.4.
  warnings.warn(

In [12]:
y_pred = model.predict(X_test)

In [13]:
metrics.accuracy_score(y_test, y_pred)
```

Results:

```
Out[13]:  0.9833333333333333
```