AIM:1- Data Pre-processing and Exploration

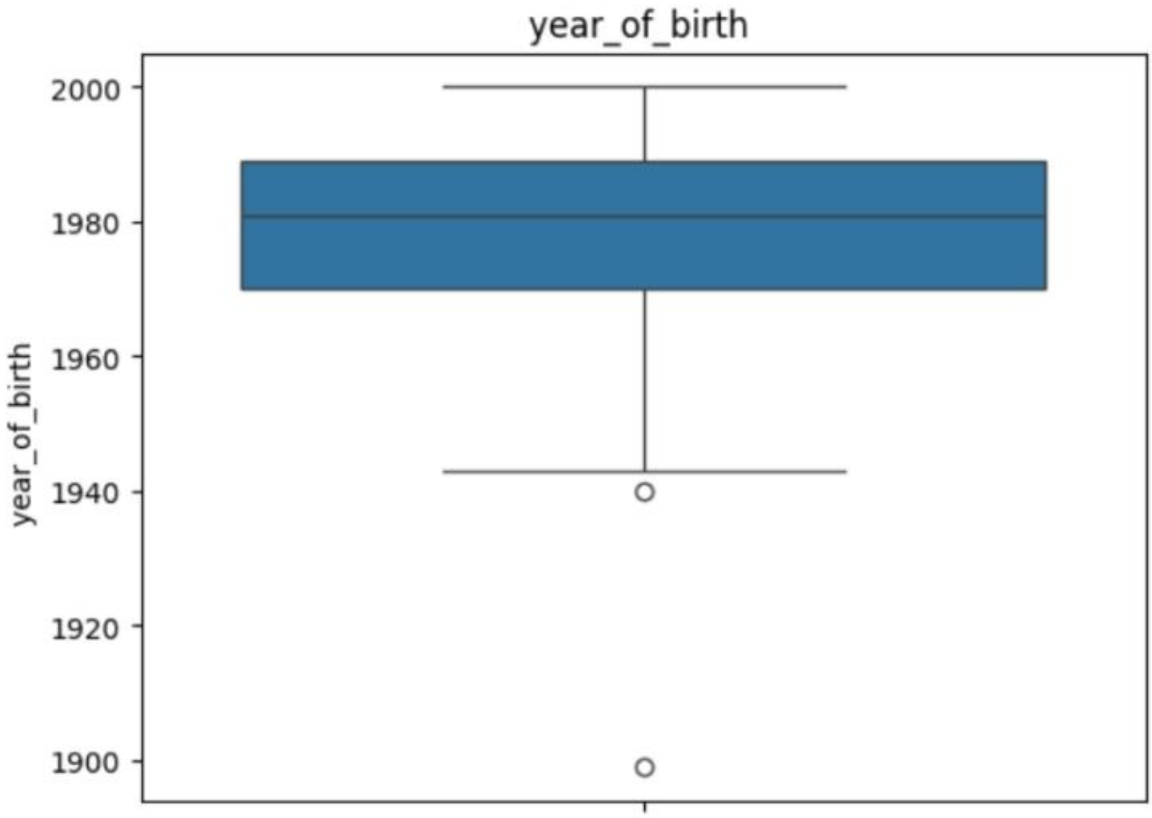
a.
Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.
b.
Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables

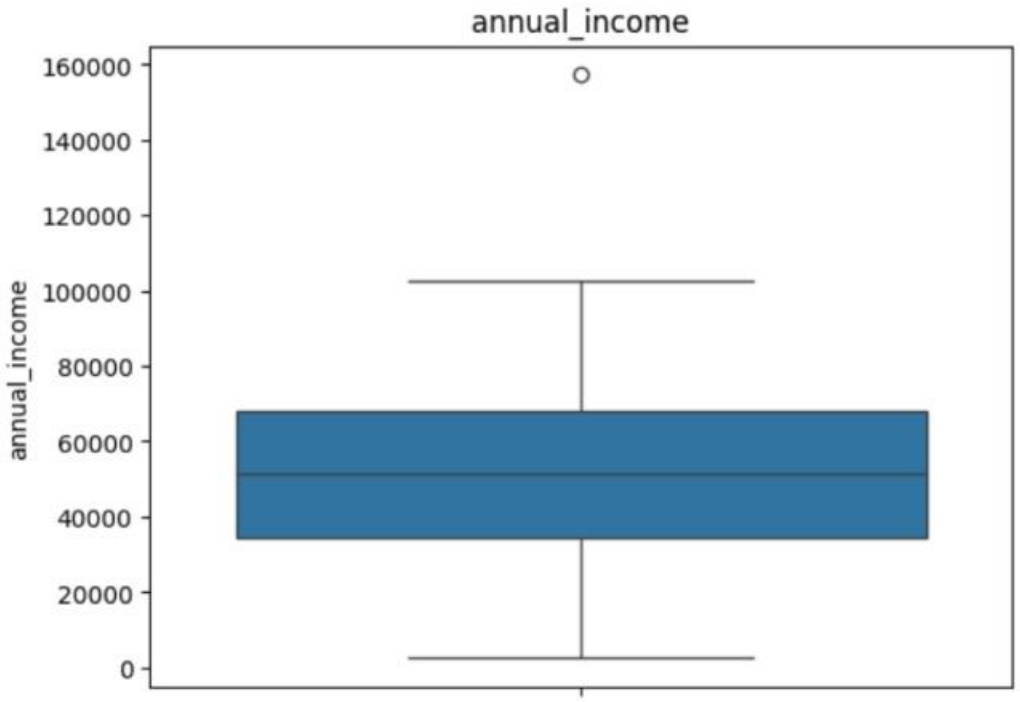
```
visualization.import pandas as pd

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

df.shape

df.columns

 df.head(3)

 df.tail(2)

 df.sample(5)



df.info()

df['annual_income']=df['annual_income'].fillna(df['annual_income'].mean())

df.info()

import matplotlib.pyplot as plt

import seaborn as sns

columns_to_check=[ 'year_of_birth ',

      'annual_income', 'online_purchases', 'complaints', 'calls', 'intercoms']

for i,column in enumerate(columns_to_check):

   plt.figure(i)
```

```
    sns.boxplot(df[column])

    plt.title(column)


#inconsistency handling

df.columns


df['educational_level'].unique()


df['marital_status'].unique()


df['marital_status']=df['marital_status'].replace('Widow','Widowed')

df['marital_status'].unique()


df['intercoms'].unique()




q1=df['year_of_birth '].quantile(0.25)

q3=df['year_of_birth '].quantile(0.75)

iqr=q3-q1

lower_bound=q1-1.5*iqr

upper_bound=q3+1.5*iqr

print(lower_bound)
```

```
print(upper_bound)

df.loc[df['year_of_birth '] <lower_bound ]

df.loc[df['year_of_birth '] >upper_bound ]

df=df.drop(df.loc[df['year_of_birth '] <lower_bound ].index)

df.loc[df['year_of_birth '] <lower_bound ]
```

OUTPUT





customer_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms

AIM:1-C Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization

```
from sklearn.datasets import fetch_california_housing

california=fetch_california_housing()

import pandas as pd

df=pd.DataFrame(california.data)

df.sample(3)

from sklearn.preprocessing import MinMaxScaler

mn=MinMaxScaler(feature_range=(0,1))

scaled_df=mn.fit_transform(df)

scaled_df=pd.DataFrame(scaled_df)

scaled_df.sample(3)
```

```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

scaled_df2=sc.fit_transform(df)

scaled_df2=pd.DataFrame(scaled_df2)

scaled_df2.sample(3)
```

OUTPUT:

	0	1	2	3	4	5	6	7
18489	0.237938	-1.004309	0.114265	-0.187656	1.898143	0.000572	0.645226	-1.008392
12494	-1.403624	1.697265	-0.633114	0.126264	-0.178798	-0.154970	1.375597	-0.948496
7460	-0.943359	0.584852	-0.367404	-0.105526	0.038433	0.175818	-0.801471	0.688653

AIM:2 Testing Hypothesis

a.
Implement and demonstrate the FIND-S algorithm for finding the most specific
hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)

```
import pandas as pd

import numpy as np

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

df

a=np.array(df)[:,:-1]

print('The attributes are:',a)

t=np.array(df)[:,-1]


print('The target is:',t)
```

```
def train(c,t):
 for i,val in enumerate(t):
  if val=='yes':
   specific_hypothesis=c[i].copy()
   break
 for i,val in enumerate(c):
  if t[i]=='yes':
   for x in range(len(specific_hypothesis)):
    if val[x]!=specific_hypothesis[x]:
     specific_hypothesis[x]='?'
    else:
     pass
 return specific_hypothesis


print("Final hypothesis is",train(a,t))


#Second code down


def find_s(examples):
  # Initialize hypothesis to the most specific hypothesis
  hypothesis =  ['ϕ', 'ϕ', 'ϕ', 'ϕ', 'ϕ', 'ϕ']
```

```
    # For each positive example in the data
    for example in examples:
        if example[-1] == 'Yes':  # Positive example
            for i in range(len(hypothesis)):
                # Update hypothesis if attribute value is different
                if hypothesis[i] == 'ϕ':
                    hypothesis[i] = example[i]
                elif hypothesis[i] != example[i]:
                    hypothesis[i] = '?'

    return hypothesis
# Example usage:
data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]
hypothesis = find_s(data)
print("Final hypothesis:", hypothesis)
```

OUTPUT

```
Final hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

AIM:3 Linear Models

a-Simple Linear Regression Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE

```
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2.
9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil>=2.8.2->p
andas) (1.16.0)
```

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from scikit-learn) (2.1.
3)
Requirement already satisfied: scipy>=1.6.0 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from scikit-learn) (1.14.
1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from scikit-learn) (1.4.
2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from scikit-learn)
(3.5.0)
```

```
!pip install matplotlib
```

```
Collecting matplotlib
  Downloading matplotlib-3.10.0-cp313-cp313-win_amd64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.1-cp313-cp313-win_amd64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.55.3-cp313-cp313-win_amd64.whl.metadata (168 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.8-cp313-cp313-win_amd64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (2.1.3)
Requirement already satisfied: packaging>=20.0 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from matplotlib) (24.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.1.0-cp313-cp313-win_amd64.whl.metadata (9.3 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.2.1-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from matplotlib)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\ravi maurya\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil>=2.7->mat
plotlib) (1.16.0)
```

```
   ---------------------------------------- 0.0/8.0 MB ? eta -:--:--
   ----- ---------------------------------- 1.0/8.0 MB 5.6 MB/s eta 0:00:02
   ------------- -------------------------- 2.6/8.0 MB 6.8 MB/s eta 0:00:01
   ----------------------- ---------------- 4.7/8.0 MB 7.7 MB/s eta 0:00:01
   -------------------------------- ------- 6.6/8.0 MB 8.2 MB/s eta 0:00:01
   ---------------------------------------- 8.0/8.0 MB 8.2 MB/s eta 0:00:00
Downloading contourpy-1.3.1-cp313-cp313-win_amd64.whl (220 kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.55.3-cp313-cp313-win_amd64.whl (2.2 MB)
   ---------------------------------------- 0.0/2.2 MB ? eta -:--:--
   ---------------------------- ----------- 1.6/2.2 MB 8.1 MB/s eta 0:00:01
   ---------------------------------------- 2.2/2.2 MB 8.0 MB/s eta 0:00:00
Downloading kiwisolver-1.4.8-cp313-cp313-win_amd64.whl (71 kB)
Downloading pillow-11.1.0-cp313-cp313-win_amd64.whl (2.6 MB)
   ---------------------------------------- 0.0/2.6 MB ? eta -:--:--
   ----------------------- ---------------- 1.6/2.6 MB 8.3 MB/s eta 0:00:01
   ---------------------------------------- 2.6/2.6 MB 9.2 MB/s eta 0:00:00
Downloading pyparsing-3.2.1-py3-none-any.whl (107 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler, contourpy, matplotlib
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.55.3 kiwisolver-1.4.8 matplotlib-3.10.0 pillow-11.1.0 pyparsing-3.2.1
```

```python
import pandas as pd
```

```python
df=pd.read_csv('Salary_Data.csv')
```

```python
df
```

	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
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0

15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

]:
```
df.shape
```

]:
```
(30, 2)
```

]:
```
X=df.iloc[:,:-1]
```

]:
```
X
```

]:

	YearsExperience
0	1.1
1	1.3
2	1.5
3	2.0
4	2.2
5	2.9
6	3.0
7	3.2
8	3.2
9	3.7

10	3.9
11	4.0
12	4.0
13	4.1
14	4.5
15	4.9
16	5.1
17	5.3
18	5.9
19	6.0
20	6.8
21	7.1
22	7.9
23	8.2
24	8.7
25	9.0

26	9.5
27	9.6
28	10.3
29	10.5

0]:
```
y=df.iloc[:,-1]
```

1]:
```
y
```

1]:
```
0      39343.0
1      46205.0
2      37731.0
3      43525.0
4      39891.0
5      56642.0
6      60150.0
7      54445.0
8      64445.0
9      57189.0
10     63218.0
11     55794.0
12     56957.0
13     57081.0
14     61111.0
```

```
15     67938.0
16     66029.0
17     83088.0
18     81363.0
19     93940.0
20     91738.0
21     98273.0
22    101302.0
23    113812.0
24    109431.0
25    105582.0
26    116969.0
27    112635.0
28    122391.0
29    121872.0
Name: Salary, dtype: float64
```

```python
from sklearn.linear_model import LinearRegression
```

```python
from sklearn.model_selection import train_test_split
```

```python
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.15,random_state=1)
```

```python
lr=LinearRegression()
```

```python
lr.fit(xtrain,ytrain)
```

```
LinearRegression
LinearRegression()
```

```python
predictions=lr.predict(xtest)
```

```python
from sklearn.metrics import r2_score
```

```python
r2_score(ytest,predictions)
```

```
0.7486825407561983
```

OUTPUT

```
0.7486825407561983
```

AIM:3B Multiple Linear Regression

Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.

```
import pandas as pd

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

df.shape

df.columns

df.drop(['Unnamed: 0'],axis=1,inplace=True)

df.columns

X=df.iloc[:,:-1]

X.columns

y=df.iloc[:,-1]

y
```

```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=4)

from sklearn.linear_model import LinearRegression

lr=LinearRegression()

lr.fit(xtrain,ytrain)

predictions=lr.predict(xtest)

from sklearn.metrics import mean_absolute_error,r2_score

mean_absolute_error(ytest,predictions)

r2_score(ytest,predictions)
```

OUTPUT

```
0.9157191830180074
```

AIM:3C Regularized Linear Models (Ridge, Lasso, ElasticNet)

Implement regression variants like LASSO and Ridge on any generated dataset.

```
[2]: import pandas as pd

[3]: df=pd.read_csv('Advertising.csv')

[4]: df.shape

[4]: (200, 5)

[6]: df.columns

[6]: Index(['Unnamed: 0', 'TV', 'radio', 'newspaper', 'sales'], dtype='object')

[7]: df.drop(['Unnamed: 0'],axis=1,inplace=True)

[8]: df.columns

[8]: Index(['TV', 'radio', 'newspaper', 'sales'], dtype='object')

[10]: from sklearn.linear_model import Ridge,Lasso

[11]: rr=Ridge(alpha=0.2)

[12]: lr=Lasso(alpha=0.2)

[15]: X=df.iloc[:,:-1]

[16]: y=df.iloc[:,-1]

[17]: from sklearn.model_selection import train_test_split

[18]: xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=4)

[19]: rr.fit(xtrain,ytrain)

[19]: Ridge
      Ridge(alpha=0.2)

[20]: lr.fit(xtrain,ytrain)

[20]: Lasso
      Lasso(alpha=0.2)

[21]: pred1=rr.predict(xtest)

[22]: pred2=lr.predict(xtest)
```

```
[23]: from sklearn.metrics import mean_absolute_error,r2_score

[24]: mean_absolute_error(ytest,pred1)

[24]: np.float64(1.153244394468935)

[26]: r2_score(ytest,pred1)

[26]: 0.9157183530461694

[27]: mean_absolute_error(ytest,pred2)

[27]: np.float64(1.1564703521717854)

[28]: r2_score(ytest,pred2)

[28]: 0.9152342518213656
```

OUTPUT

```
[28]: 0.9152342518213656
```

Aim4.a Discriminative Models

a Logistic Regression Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve.

```
from sklearn.datasets import load_breast_cancer
```

```
ds=load_breast_cancer()
```

```
X=ds.data
```

```
y=ds.target
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=4)
```

```
from sklearn.linear_model import LogisticRegression
```

```
lr=LogisticRegression()
```

```
lr.fit(xtrain,ytrain)
```

```
LogisticRegression
LogisticRegression()
```

```
preds=lr.predict(xtest)
```

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,roc_auc_score
```

```
accuracy_score(ytest,preds)
```

```
0.8811188811188811
```

```
precision_score(ytest,preds)
```

```
np.float64(0.9148936170212766)
```

```
recall_score(ytest,preds)
```

```
np.float64(0.9052631578947369)
```

```
roc_auc_score(ytest,preds)
```

```
np.float64(0.8692982456140352)
```

OUTPUT

```
np.float64(0.8692982456140352)
```

Aim4.

B Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions

```
import pandas as pd
```

```
df=pd.read_csv('Iris.csv')
```

```
df.shape
```

```
(150, 5)
```

```
df.columns
```

```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
X=df.iloc[:,:-1]
```

```
y=ddf.iloc[:,:-1]
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=26)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn=KNeighborsClassifier(n_neighbors=6)
```

```
knn.fit(xtrain,ytrain)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

```
preds=knn.predict(xtest)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(ytest,preds)
```

```
0.9473684210526315
```

OUTPUT

```
]:  0.9473684210526315
```

Aim4.CBuild a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

```
import pandas as pd
```

```
df=pd.read_csv('Iris.csv')
```

```
df.columns
```

```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
x=df.iloc[:,:-1]
```

```
y=df.iloc[:,-1]
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=4)
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt=DecisionTreeClassifier(max_depth=3)
```

```
dt.fit(xtrain,ytrain)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

```
preds=dt.predict(xtest)
```

```
from sklearn.metrics import accuracy_score
```

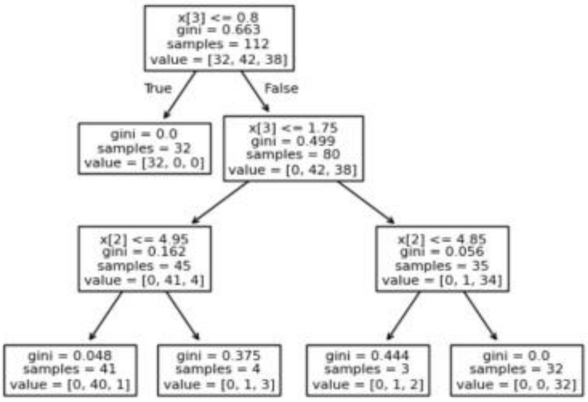
```
accuracy_score(ytest,preds)
```

```
0.9736842105263158
```

```
from sklearn import tree
```

```
tree.plot_tree(dt)
```

```
[Text(0.375, 0.875, 'x[3] <= 0.8\ngini = 0.663\nsamples = 112\nvalue = [32, 42, 38]'),
 Text(0.25, 0.625, 'gini = 0.0\nsamples = 32\nvalue = [32, 0, 0]'),
 Text(0.3125, 0.75, 'True  '),
 Text(0.5, 0.625, 'x[3] <= 1.75\ngini = 0.499\nsamples = 80\nvalue = [0, 42, 38]'),
 Text(0.4375, 0.75, '  False'),
 Text(0.25, 0.375, 'x[2] <= 4.95\ngini = 0.162\nsamples = 45\nvalue = [0, 41, 4]'),
 Text(0.125, 0.125, 'gini = 0.048\nsamples = 41\nvalue = [0, 40, 1]'),
 Text(0.375, 0.125, 'gini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.75, 0.375, 'x[2] <= 4.85\ngini = 0.056\nsamples = 35\nvalue = [0, 1, 34]'),
 Text(0.625, 0.125, 'gini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.875, 0.125, 'gini = 0.0\nsamples = 32\nvalue = [0, 0, 32]')]
```




```
[21]: #Below code for e
      from sklearn.ensemble import RandomForestClassifier
```

```
]: rf=RandomForestClassifier()
```

```
]: rf.fit(xtrain,ytrain)
```

```
]: ▾ RandomForestClassifier
   RandomForestClassifier()
```

```
]: preds2=rf.predict(xtest)
```

```
]: accuracy_score(ytest,preds2)
```

```
]: 0.9736842105263158
```

OUTPUT

```
0.9736842105263158
```

Aim4.D Implement a Support Vector Machine for any relevant dataset

```
from sklearn.datasets import load_breast_cancer
```

```
ds=load_breast_cancer()
```

```
x=ds.data
```

```
y=ds.target
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=4)
```

```
from sklearn.svm import SVC
```

```
sv=SVC(kernel='linear')
```

```
sv.fit(xtrain,ytrain)
```

```
SVC
SVC(kernel='linear')
```

```
preds=sv.predict(xtest)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(ytest,preds)
```

```
0.9370629370629371
```

OUTPUT

```
0.9370629370629371
```

Aim4.E Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree.

```
import pandas as pd
```

```
df=pd.read_csv('Iris.csv')
```

```
df.columns
```

```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
x=df.iloc[:,:-1]
```

```
y=df.iloc[:,-1]
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=4)
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt=DecisionTreeClassifier(max_depth=3)
```

```
dt.fit(xtrain,ytrain)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

```
preds=dt.predict(xtest)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(ytest,preds)
```

```
0.9736842105263158
```

```
from sklearn import tree
```

```
tree.plot_tree(dt)
```

```
[Text(0.375, 0.875, 'x[3] <= 0.8\ngini = 0.663\nsamples = 112\nvalue = [32, 42, 38]'),
 Text(0.25, 0.625, 'gini = 0.0\nsamples = 32\nvalue = [32, 0, 0]'),
 Text(0.3125, 0.75, 'True  '),
 Text(0.5, 0.625, 'x[3] <= 1.75\ngini = 0.499\nsamples = 80\nvalue = [0, 42, 38]'),
 Text(0.4375, 0.75, '  False'),
 Text(0.25, 0.375, 'x[2] <= 4.95\ngini = 0.162\nsamples = 45\nvalue = [0, 41, 4]'),
 Text(0.125, 0.125, 'gini = 0.048\nsamples = 41\nvalue = [0, 40, 1]'),
 Text(0.375, 0.125, 'gini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.75, 0.375, 'x[2] <= 4.85\ngini = 0.056\nsamples = 35\nvalue = [0, 1, 34]'),
 Text(0.625, 0.125, 'gini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.875, 0.125, 'gini = 0.0\nsamples = 32\nvalue = [0, 0, 32]')]
```

```
#Below code for e
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier()
```

```
rf.fit(xtrain,ytrain)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
preds2=rf.predict(xtest)
```

```
accuracy_score(ytest,preds2)
```

```
0.9736842105263158
```

OUTPUT

```
0.9736842105263158
```

AIM:5- Generative Models

a.

Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.

```
from sklearn.datasets import load_breast_cancer
```

```
ds=load_breast_cancer()
```

```
X=ds.data
```

```
y=ds.target
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=4)
```

```
from sklearn.naive_bayes import GaussianNB
```

```
nb=GaussianNB()
```

```
nb.fit(xtrain,ytrain)
```

```
GaussianNB
GaussianNB()
```

```
preds=nb.predict(xtest)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(ytest,preds)
```

```
0.9300699300699301
```

OUTPUT

```
0.9300699300699301
```

AIM:6-B Implement Bayesian Linear Regression to explore prior and posterior distribution

```
import pandas as pd
```

```
from sklearn.mixture import GaussianMixture
```

```
df=pd.read_csv('Clustering_gmm.csv')
```

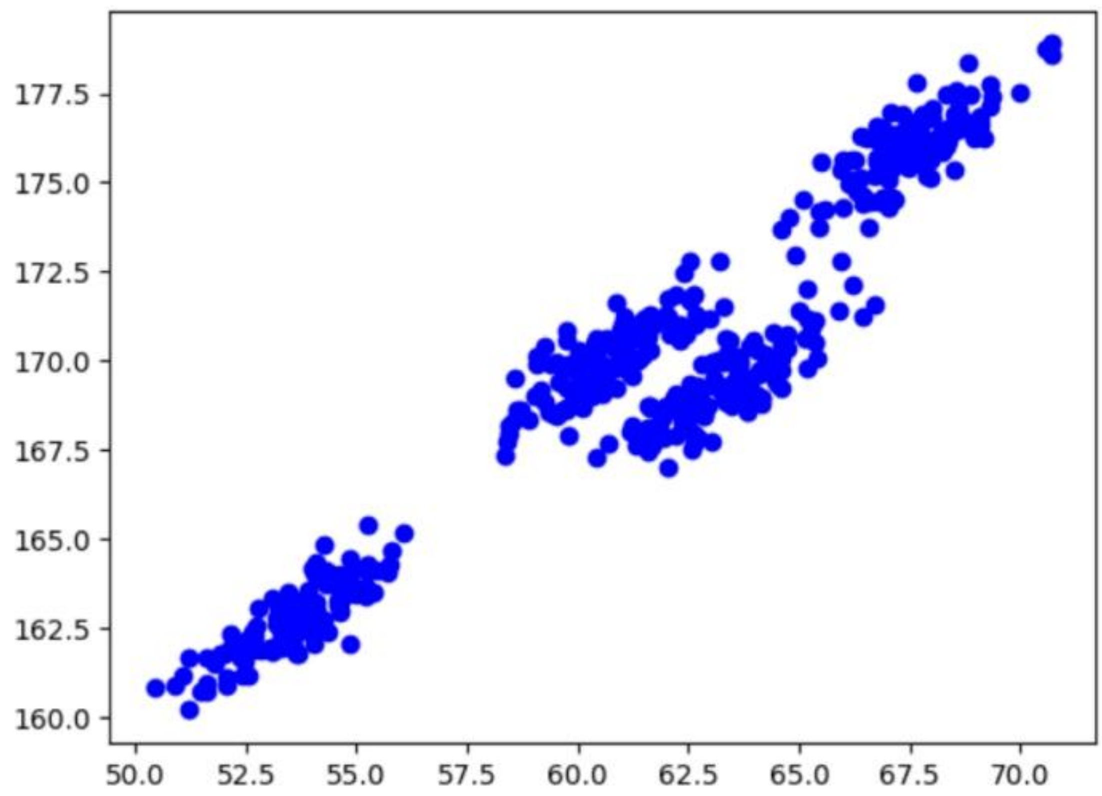
```
df
```

	Weight	Height
0	67.062924	176.086355
1	68.804094	178.388669
2	60.930863	170.284496
3	59.733843	168.691992
4	65.431230	173.763679
...	...	...
495	59.976983	169.679741
496	66.423814	174.625574
497	53.604698	161.919208
498	50.433644	160.794875
499	60.224392	169.689709

500 rows × 2 columns

```
:5]: import matplotlib.pyplot as plt
     plt.scatter(df['Weight'],df['Height'], color='blue')


     #df.plot(kind='scatter', x='x', y='y', color='blue', alpha=0.5, title="Scatter Plot using Pandas")
```

```
:5]: <matplotlib.collections.PathCollection at 0x14162a9cb90>
```




```
gmm=GaussianMixture(n_components=4)
```

```
gmm.fit(df)
```

```
GaussianMixture
GaussianMixture(n_components=4)
```

```
predictions=gmm.predict(df)
```

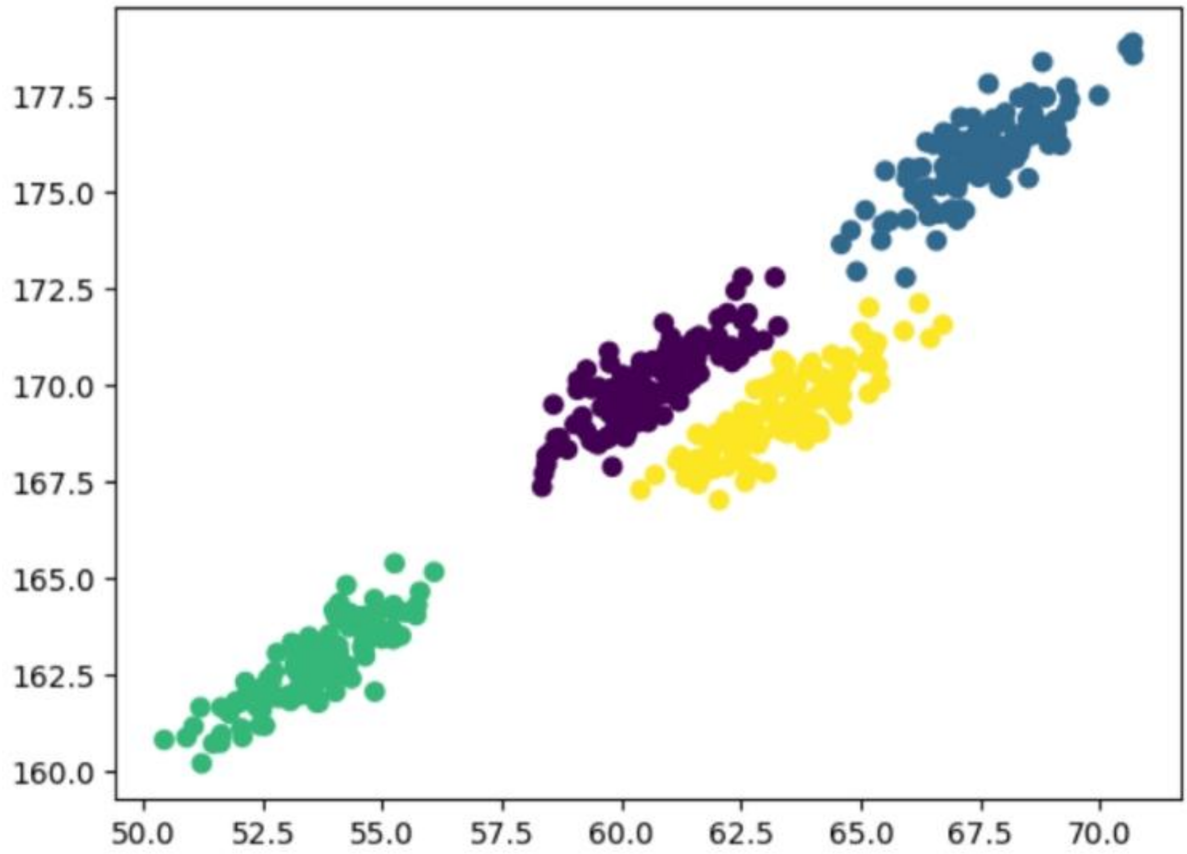
```
predictions
```

```
array([1, 1, 0, 0, 1, 3, 3, 0, 0, 2, 3, 1, 3, 2, 0, 0, 1, 0, 2, 0, 0, 1,
       3, 2, 0, 2, 1, 3, 2, 3, 3, 3, 1, 3, 3, 1, 2, 0, 2, 1, 2, 0, 0, 1,
       3, 1, 0, 0, 1, 1, 2, 2, 3, 0, 0, 3, 3, 2, 2, 1, 1, 2, 1, 1, 1, 3,
       0, 1, 0, 3, 2, 0, 2, 1, 2, 1, 3, 3, 0, 0, 2, 0, 2, 0, 0, 3, 2, 2,
       1, 2, 2, 2, 1, 2, 0, 0, 1, 1, 1, 0, 2, 2, 1, 2, 0, 0, 3, 1, 0, 2,
       2, 3, 2, 2, 1, 2, 1, 0, 3, 1, 0, 2, 1, 1, 2, 1, 1, 1, 1, 2, 0, 2,
       3, 2, 2, 2, 0, 1, 0, 3, 1, 1, 1, 1, 3, 2, 0, 0, 3, 2, 2, 1, 0, 0,
       0, 3, 3, 1, 3, 0, 1, 0, 1, 1, 3, 3, 1, 0, 2, 3, 1, 2, 1, 3, 0, 2,
       0, 2, 2, 2, 1, 1, 1, 0, 3, 2, 3, 1, 3, 2, 1, 3, 2, 3, 3, 2, 0, 3,
       0, 2, 2, 0, 3, 3, 0, 3, 3, 0, 3, 3, 3, 1, 1, 1, 0, 1, 3, 2, 2, 2,
       0, 1, 3, 2, 3, 1, 2, 0, 2, 3, 3, 3, 3, 3, 0, 0, 0, 3, 2, 1, 1, 3,
       3, 1, 2, 2, 0, 1, 0, 0, 0, 3, 1, 2, 0, 2, 2, 3, 2, 3, 0, 0, 1, 2,
       1, 1, 0, 0, 2, 1, 0, 2, 1, 3, 1, 2, 3, 3, 0, 1, 1, 0, 1, 3, 1, 3,
       3, 1, 0, 0, 0, 1, 2, 2, 1, 2, 2, 2, 0, 1, 1, 2, 0, 2, 1, 1, 3, 2,
       2, 0, 2, 0, 1, 0, 2, 3, 2, 3, 3, 0, 0, 0, 3, 3, 2, 1, 0, 3, 2, 1,
       0, 0, 1, 1, 3, 1, 3, 3, 0, 3, 2, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 2, 1, 0, 2, 2, 3, 0, 1, 1, 3, 3, 1, 3, 2, 0, 0, 0, 2,
       0, 3, 0, 2, 2, 0, 2, 2, 3, 2, 2, 3, 3, 3, 3, 2, 2, 2, 2, 1, 2, 1,
       0, 1, 2, 0, 1, 0, 1, 0, 3, 3, 2, 0, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3,
       3, 0, 1, 0, 1, 3, 1, 0, 1, 2, 3, 1, 2, 1, 0, 3, 0, 3, 1, 2, 0, 0,
       3, 1, 1, 0, 0, 3, 0, 3, 2, 1, 3, 1, 2, 3, 3, 3, 2, 3, 0, 2, 3, 3,
       3, 3, 3, 0, 2, 1, 3, 1, 0, 1, 3, 2, 0, 2, 3, 2, 0, 2, 0, 3, 3, 2,
       3, 3, 3, 2, 1, 3, 3, 2, 0, 3, 3, 0, 1, 2, 2, 0])
```

```
plt.scatter(df['Weight'],df['Height'],c=predictions)
```

```
<matplotlib.collections.PathCollection at 0x14162ce8f50>
```

OUTPUT



177.5
175.0
172.5
170.0
167.5
165.0
162.5
160.0
50.0
52.5
55.0
57.5
60.0
62.5
65.0
67.5
70.0

AIM:7-A Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
iris=load_iris()
X = iris.data
Y = iris.target
Ir = LogisticRegression()
```

```
k_fold = KFold(n_splits = 5)
score = cross_val_score(Ir, X, Y , cv = k_fold)
print ("Cross Validation Score : {}", format(score))
```

```
Cross Validation Score : {} [1.         1.         0.86666667 0.93333333 0.83333333]
```

holdout cross validation

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
iris = load_iris()
X = iris.data
Y = iris.target
Ir = LogisticRegression()
xtrain,xtest,ytrain,ytest = train_test_split(X , Y , test_size = 0.25 , random_state = 1)
Ir.fit(xtrain, ytrain)
```

```
LogisticRegression
LogisticRegression()
```

```
prediction = Ir.predict(xtest)
print("Testing Accurcy : {}".format(accuracy_score(ytest,prediction)))
```

```
Testing Accurcy : 0.9736842105263158
```

Stratified K-Fold cross Validation

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
```

```
import pandas as pd
df = pd.read_csv('pima-indians-diabetes.data.csv')
```

```
X=df.iloc[:,:-1]
```

```
y=df.iloc[:,-1]
```

```
lr=LogisticRegression()
```

```
skf=StratifiedKFold(n_splits=3)
```

```
score = cross_val_score(lr, X, y , cv = skf)
```

```
print('Scores',score)
```

```
Scores [0.76953125 0.73046875 0.8       ]
```

OUTPUT


```
Scores [0.76953125 0.73046875 0.8       ]
```

AIM:8 Implement Bayesian Learning using inferences

```
from sklearn.datasets import load_breast_cancer
```

```
ds=load_breast_cancer()
```

```
X=ds.data
```

```
y=ds.target
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=4)
```

```
from sklearn.naive_bayes import GaussianNB
```

```
nb=GaussianNB()
```

```
nb.fit(xtrain,ytrain)
```

```
▾ GaussianNB
GaussianNB()
```

```
preds=nb.predict(xtest)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(ytest,preds)
```

```
0.9300699300699301
```

OUTPUT

```
0.9300699300699301
```

Aim:9-A Set up a generator network to produce samples and a discriminator network to distinguish between real and generated data. (Use a simple small dataset)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.utils import shuffle
```

```
# Generate real dataset (points in a 2D Gaussian)
np.random.seed(0)
real_data = np.random.randn(500, 2)  # 500 points in 2D
real_data
```

```
def generate_fake_samples(n=500):
    return np.random.uniform(-3, 3, size=(n, 2))
```

```
y_real = np.ones(500)  # Real labels
y_fake = np.zeros(500)  # Fake labels
```

```
y_fake
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
```

```
X_train = np.vstack([real_data, generate_fake_samples()])
y_train = np.hstack([y_real, y_fake])
```

```
X_train.shape
```

```
(1000, 2)
```

```
y_train.shape
```

```
(1000,)
```

```
X_train, y_train = shuffle(X_train, y_train, random_state=0)
```

```
discriminator = LogisticRegression()
discriminator.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```python
accuracy = discriminator.score(X_train, y_train)
print(f"Discriminator Accuracy: {accuracy:.2f}")
```

```
Discriminator Accuracy: 0.53
```

```python
# Improve generator: Adjust distribution to fool discriminator
fake_samples = generate_fake_samples()
fake_probs = discriminator.predict_proba(fake_samples)[:, 1]  # Probability of being real
better_fake_samples = fake_samples[fake_probs > 0.5]  # Keep more realistic samples
```

```python
plt.figure(figsize=(6, 6))
plt.scatter(real_data[:, 0], real_data[:, 1], label="Real Data", alpha=0.6)
plt.scatter(better_fake_samples[:, 0], better_fake_samples[:, 1], label="Generated Data", alpha=0.6)
plt.legend()
plt.title("Real vs. Generated Data (Sklearn GAN)")
plt.show()
```