ASSIGNMENT – 4

]

Assignment Date	27 October 2022
Student Name	Abinaya.M
Student Roll Number	820319104002
Maximum Marks	2 Marks

Problem Statement: Customer Segmentation Analysis

Problem Statement :

Chronic Kidney Disease prediction is one of the most importants in healthcare analytics. The most interesting and challenging tasks in day-to-day life is prediction in the medical field. 10% of the world is affected by counc kidney disease (CHD), and millions die each year because they do not have access to affordable treatment. Chronic Kidney Dis can be cured, if treated in the early stages. The main aim of this project is to predict whether the patients have chronic kidney disease or not, in a more accurate and faster way based on certain diagnostic measurements like Blood Pressure(Bp), Albumin(Al).

Clustering the data and performing classification

Algorithms

1. Download the dataset: [Dataset]

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from google.colab import drive

drive.mount('/content/drive')
```

2. Load the dataset into the tool.

```
data.head()
```

```
   CustomerID Gender Age Annual Income (k$) Spending Score (1-100) 0 1
Male 19 15 39 1 2 Male 21 15 81 2 3 Female 20 16 6 3 4 Female 23 16 77
4 5 Female 31 17 40
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----
 0 CustomerID 200 non-null int64
 1 Gender 200 non-null object
 2 Age 200 non-null int64
 3 Annual Income (k$) 200 non-null int64
 4 Spending Score (1-100) 200 non-null int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
data.describe()
```

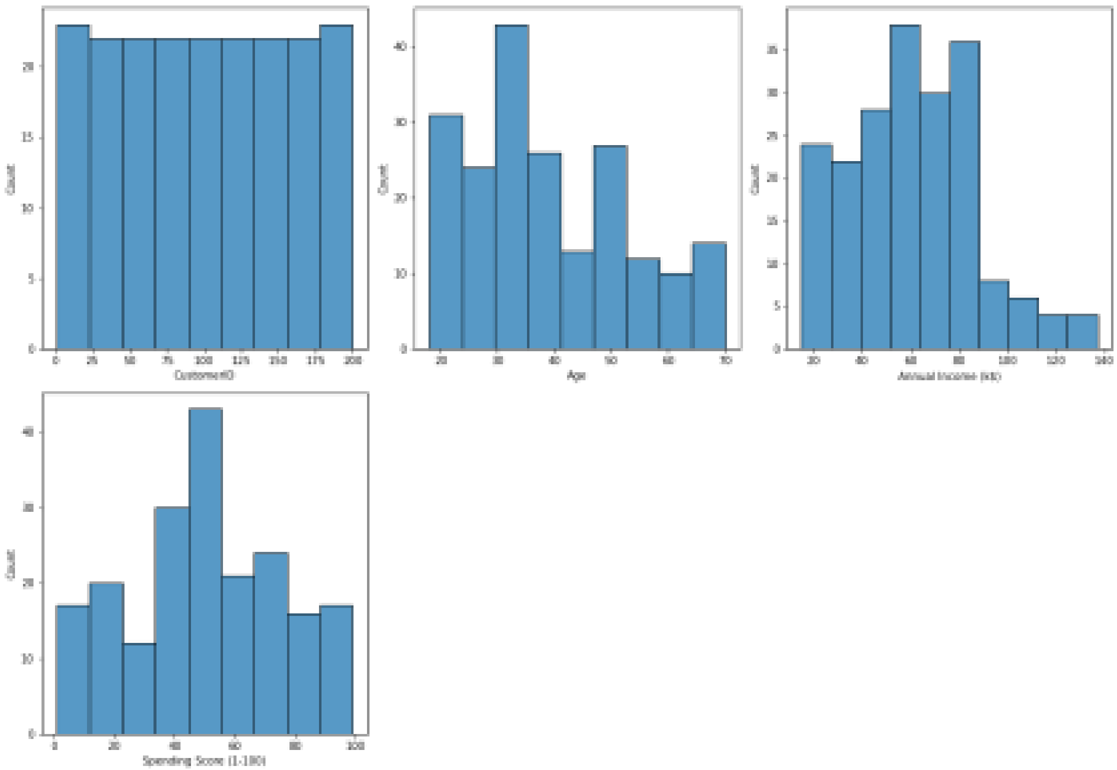
```
       CustomerID  Age  Annual  Income  (k$)  Spending  Score  (1-100)  count
200.000000  200.000000  200.000000  200.000000  mean  100.500000  38.850000
60.560000  50.200000  std  57.879185  13.969007  26.264721  25.823522  min
1.000000 18.000000 15.000000 1.000000 25% 50.750000 28.750000 41.500000
34.750000  50%  100.500000  36.000000  61.500000  50.000000  75%  150.250000
49.000000   78.000000   73.000000   max   200.000000   70.000000   137.000000
99.000000
```

3. Perform Below Visualizations.

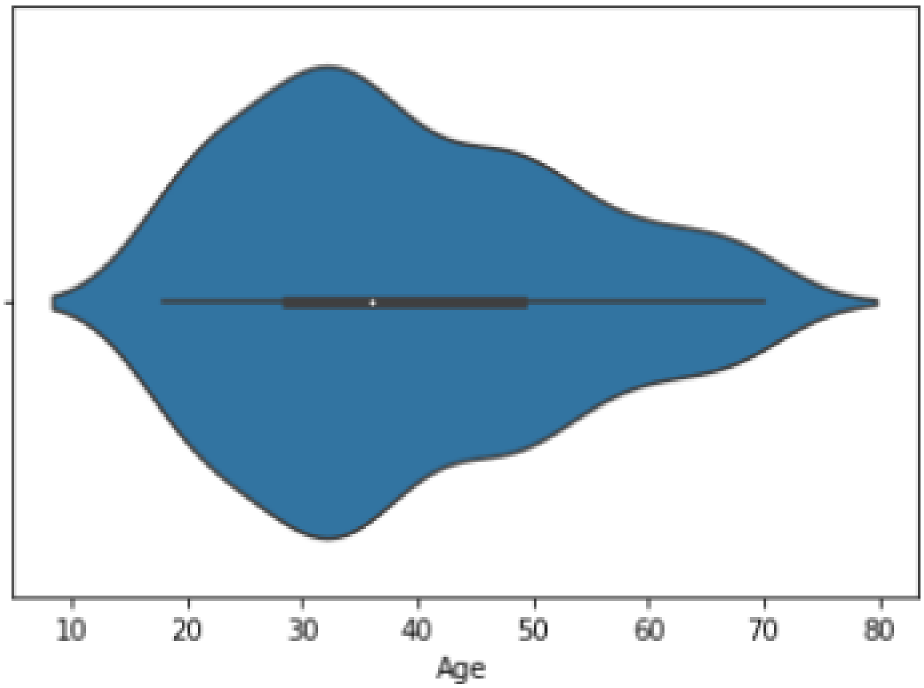
univariate analysis

```
#univariate analysis
cols = 3
rows = 3
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure( figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):
```

```
    ax=fig.add_subplot(rows,cols,i+1)

    sns.histplot(x = data[col], ax = ax)

fig.tight_layout()
plt.show()
```



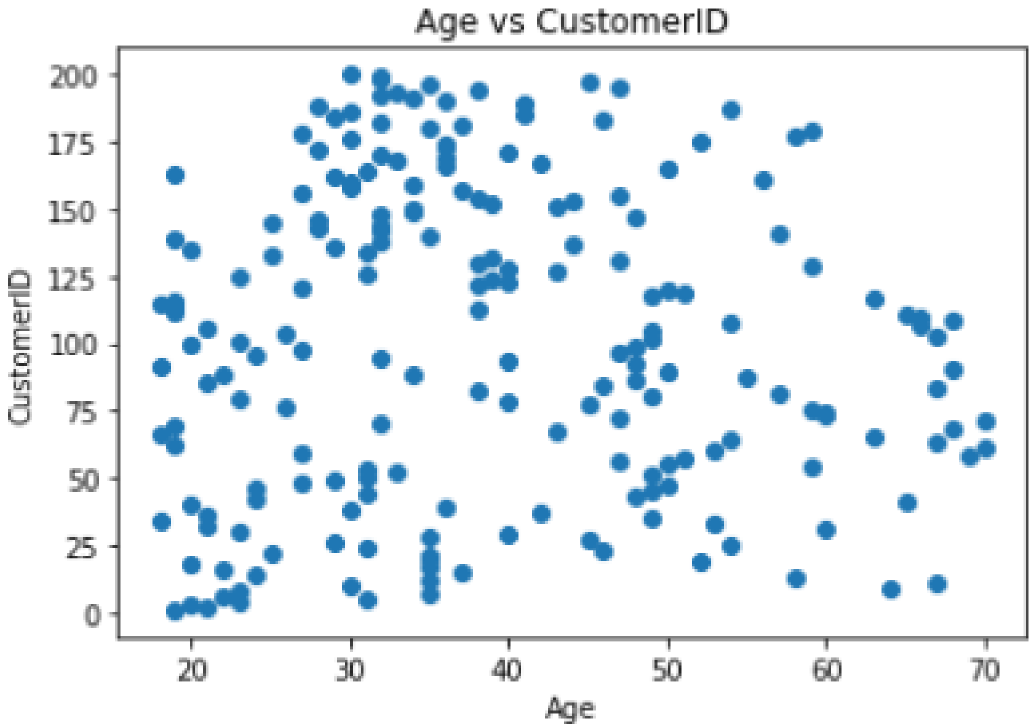
```
sns.violinplot(x=data["Age"])
<matplotlib.axes._subplots.AxesSubplot at 0x7fa3726b9490>
```



Bivariate analysis

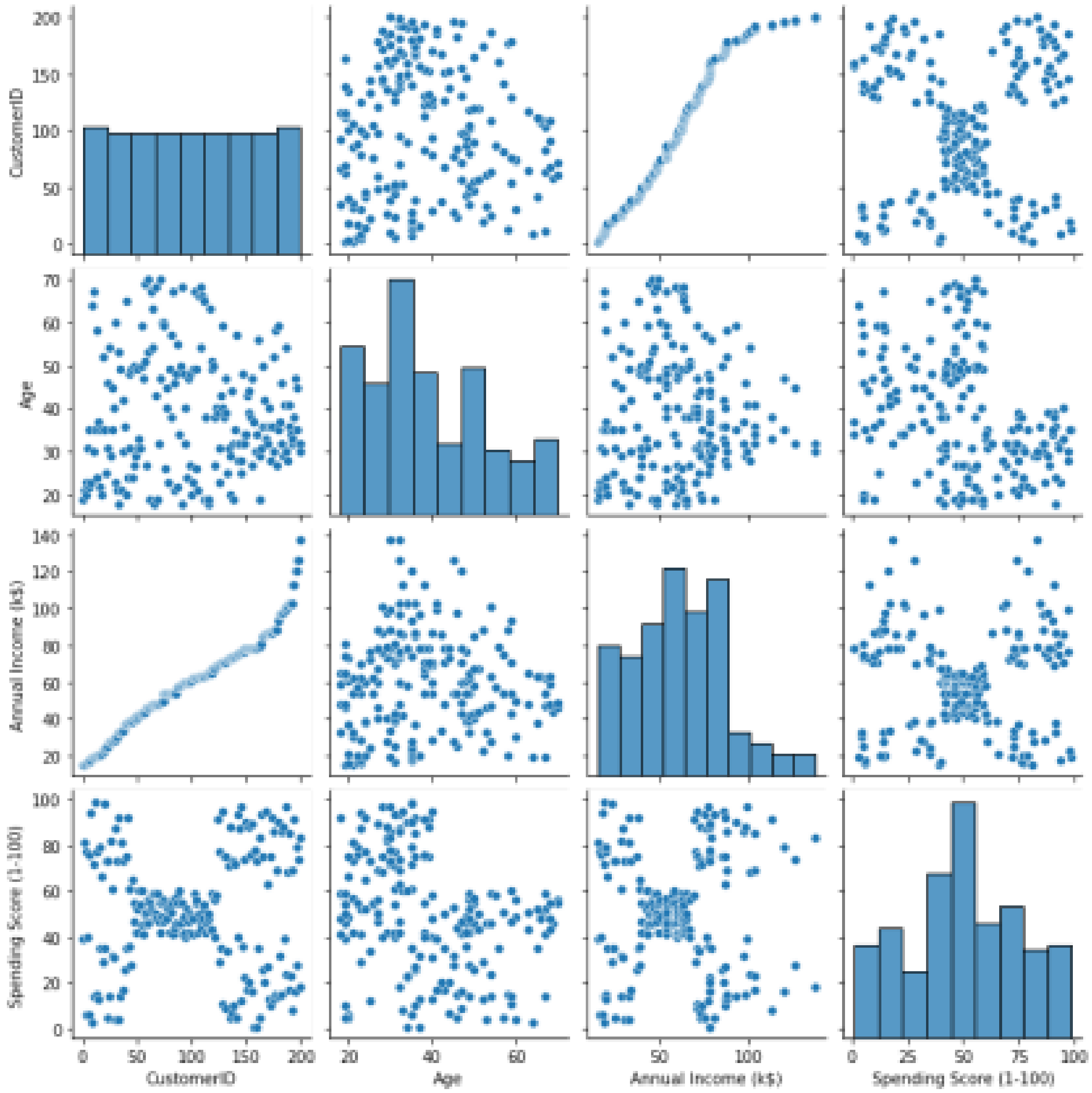
```
import matplotlib.pyplot as plt

#create scatterplot of hours vs. score
plt.scatter(data.Age, data.CustomerID)
plt.title('Age vs CustomerID')
plt.xlabel('Age')
plt.ylabel('CustomerID')
```

```
Text(0, 0.5, 'CustomerID')
```



Multivariate analysis

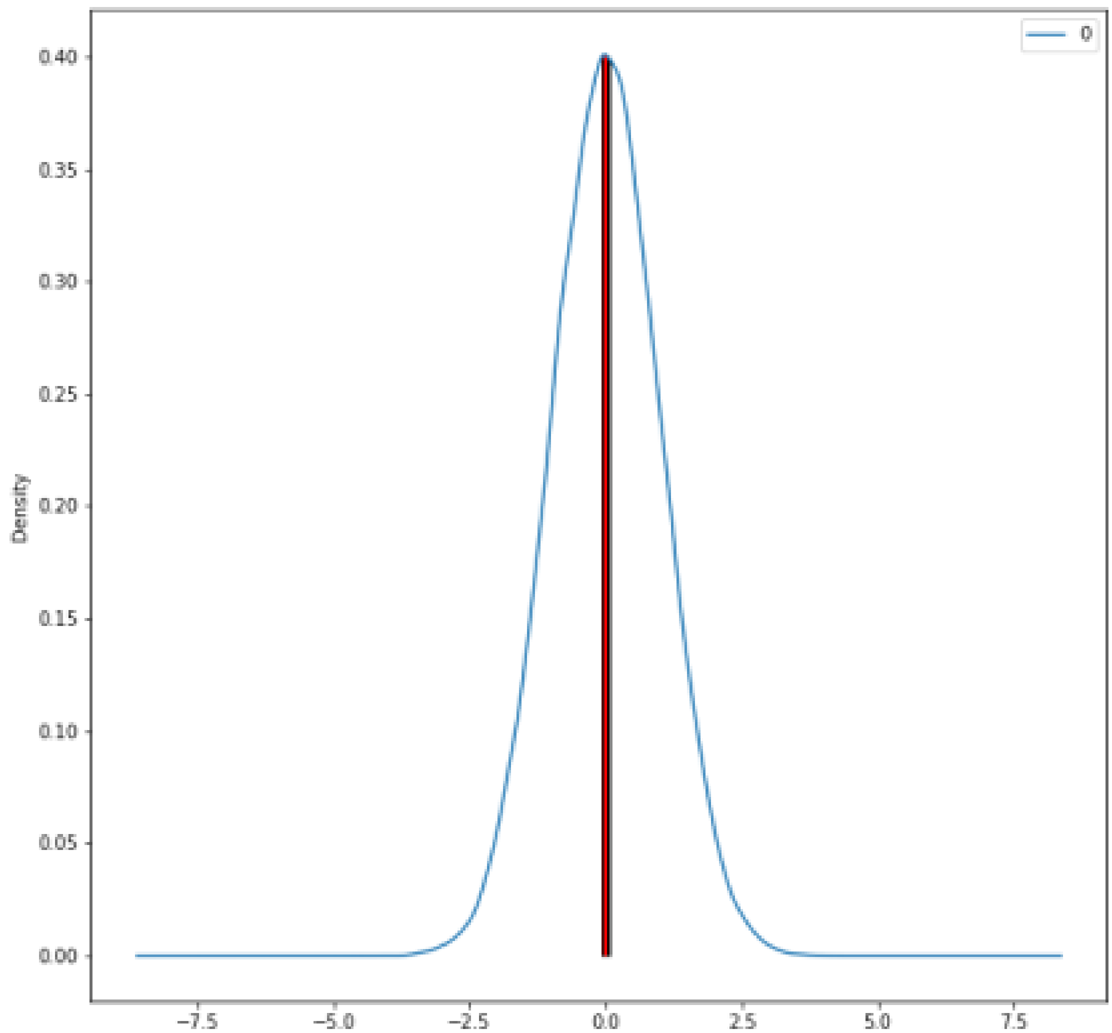
```
sns.pairplot(data);
```



4. Perform descriptive statistics on the dataset.

```
data.mean()

CustomerID 100.50
Age 38.85
Annual Income (k$) 60.56
Spending Score (1-100) 50.20
dtype: float64
data.median()

CustomerID 100.5
Age 36.0
Annual Income (k$) 61.5
Spending Score (1-100) 50.0
dtype: float64
```

```
norm_data = pd.DataFrame(np.random.normal(size=100000))

norm_data.plot(kind="density",
              figsize=(10,10));


plt.vlines(norm_data.mean(), # Plot black line at mean
           ymin=0,
           ymax=0.4,
           linewidth=5.0);

plt.vlines(norm_data.median(), # Plot red line at median
           ymin=0,
           ymax=0.4,
           linewidth=2.0,
           color="red");
```



5. Check for Missing values and deal with them.

Identifying the missing value

```
df = pd.DataFrame(data)
df.isnull()
```

```
     CustomerID Gender Age Annual Income (k$) Spending Score (1-100) 0
False False False False False 1 False False False False False 2 False
False False False False 3 False False False False False 4 False False
False False False .. ... ... ... ... ... 195 False False False False False
196 False False False False False 197 False False False False False
198 False False False False False 199 False False False False False

[200 rows x 5 columns]
```

Filling the missing value with previous value

```
df.fillna(method ='pad')
```

```
     CustomerID Gender Age Annual Income (k$) Spending Score (1-100) 0 1
Male 19 15 39 1 2 Male 21 15 81 2 3 Female 20 16 6 3 4 Female 23 16 77 4
5 Female 31 17 40 .. ... ... ... ... ... 195 196 Female 35 120 79 196
197 Female 45 126 28 197 198 Male 32 126 74 198 199 Male 32 137 18 199
200 Male 30 137 83

[200 rows x 5 columns]
```

Filling null values in missing values

```
data[0:]
```

```
     CustomerID Gender Age Annual Income (k$) Spending Score (1-100) 0 1
Male 19 15 39 1 2 Male 21 15 81 2 3 Female 20 16 6 3 4 Female 23 16 77 4
5 Female 31 17 40 .. ... ... ... ... ... 195 196 Female 35 120 79 196
197 Female 45 126 28 197 198 Male 32 126 74 198 199 Male 32 137 18 199
200 Male 30 137 83

[200 rows x 5 columns]
```

6. Find the outliers and replace them outliers

Identifying the outliers

```
print(df['Annual Income (k$)'].skew())
df['Annual Income (k$)'].describe()
```

```
0.3218425498619055

count 200.000000
mean 60.560000
```

```
std 26.264721
min 15.000000
25% 41.500000
50% 61.500000
75% 78.000000
max 137.000000
Name: Annual Income (k$), dtype: float64
```

Replacing the outliers

```
print(df['Annual Income (k$)'].quantile(0.50))
print(df['Annual Income (k$)'].quantile(0.95))
df['Annual Income (k$)'] = np.where(df['Annual Income (k$)'] > 325, 140,
df['Annual Income (k$)'])
df.describe()
```

```
61.5
103.0
```

```
       CustomerID Age Annual Income (k$) Spending Score (1-100) count
200.000000 200.000000 200.000000 200.000000 mean 100.500000 38.850000
60.560000 50.200000 std 57.879185 13.969007 26.264721 25.823522 min
1.000000 18.000000 15.000000 1.000000 25% 50.750000 28.750000 41.500000
34.750000 50% 100.500000 36.000000 61.500000 50.000000 75% 150.250000
49.000000 78.000000 73.000000 max 200.000000 70.000000 137.000000
99.000000
```

7. Check for Categorical columns and perform encoding.

Perform encoding

```
from sklearn.compose import make_column_selector as selector

categorical_columns_selector =

selector(dtype_include=object)

categorical_columns = categorical_columns_selector(data)
categorical_columns
```

```
['Gender']
```

```
data_categorical = data[categorical_columns]
data_categorical.head()
```

```
   Gender
0 Male
```

```
1 Male
2 Female
3 Female
4 Female
```

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Gender']= label_encoder.fit_transform(df['Gender'])
df['Gender'].unique()
```

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

```
# import packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# importing data
print(df.shape)

# head of the data
print('Head of the dataframe : ')
print(df.head())

print(df.columns)

X= df['Annual Income (k$)']
y=df['Spending Score (1-100)']

# using the train test split function
X_train, X_test, y_train, y_test = train_test_split(
X,y , random_state=104,test_size=0.25, shuffle=True)

# printing out train and test sets

print('X_train : ')
print(X_train.head())
print(X_train.shape)

print('')
print('X_test : ')
print(X_test.head())
print(X_test.shape)
```

```
print('')
print('y_train : ')
print(y_train.head())
print(y_train.shape)

print('')
print('y_test : ')
print(y_test.head())
print(y_test.shape)
```

```
(200, 5)
Head of the dataframe :
   CustomerID Gender Age Annual Income (k$) Spending Score (1-100) 0 1
1 19 15 39 1 2 1 21 15 81 2 3 0 20 16 6 3 4 0 23 16 77 4 5 0 31 17 40
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
X_train :
73 50
30 30
23 25
155 78
157 78
Name: Annual Income (k$), dtype: int64
(150,)

X_test :
104 62
128 71
49 40
34 33
64 48
Name: Annual Income (k$), dtype: int64
(50,)

y_train :
73 56
30 4
23 73
155 89
157 78
Name: Spending Score (1-100), dtype: int64
(150,)

y_test :
104 56
128 11
```

```
49 42
34 14
64 51
Name: Spending Score (1-100), dtype: int64
(50,)
```

8. Scaling the data

Scaling

```
df_scaled = df.copy()
col_names = ['Annual Income (k$)', 'Spending Score (1-100)']
features = df_scaled[col_names]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(5, 10))

df_scaled[col_names] = scaler.fit_transform(features.values)
df_scaled
```

```
    CustomerID Gender Age Annual Income (k$) Spending Score (1-100) 0 1
1 19 5.000000 6.938776 1 2 1 21 5.000000 9.081633 2 3 0 20 5.040984
5.255102 3 4 0 23 5.040984 8.877551 4 5 0 31 5.081967 6.989796 .. ...
... ... ... ... 195 196 0 35 9.303279 8.979592 196 197 0 45 9.549180
6.377551 197 198 1 32 9.549180 8.724490 198 199 1 32 10.000000 5.867347
199 200 1 30 10.000000 9.183673

[200 rows x 5 columns]
```

9. Perform any of the clustering algorithms

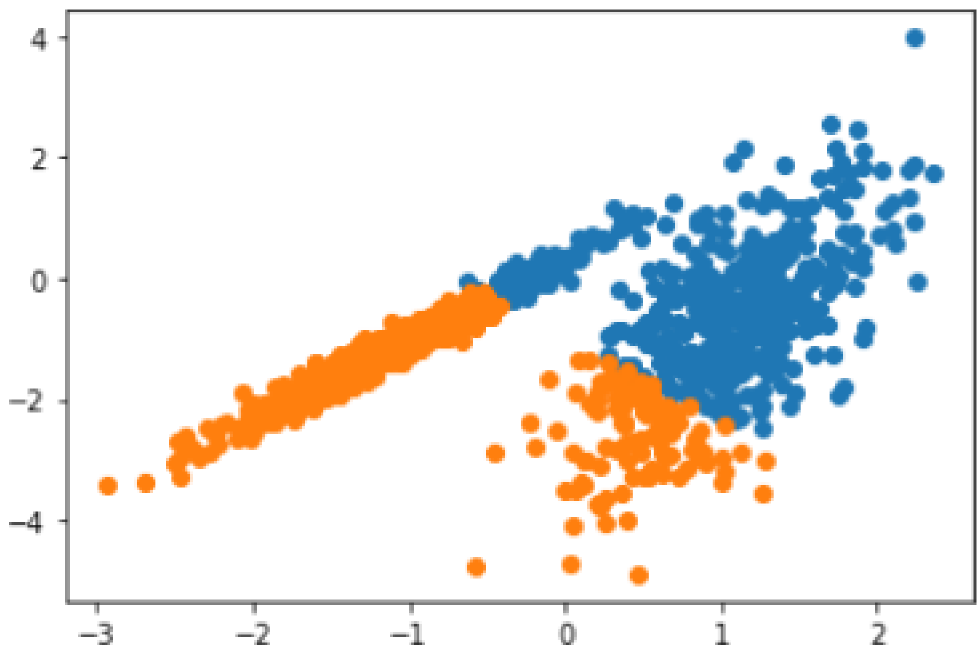
```
# k-means clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot
```

10. Add the cluster data with the primary dataset

```
# define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2,
n_redundant=0, n_clusters_per_class=1, random_state=4)
```

11. Split the data into dependent and independent variables.

```
# define the model
model = KMeans(n_clusters=2)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
    # get row indexes for samples with this cluster
    row_ix = where(yhat == cluster)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```



12. Split the data into training and testing

```
#testing and training
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=0)
```

```
print(X_train, X_test, y_train, y_test)

     CustomerID Gender Age Annual Income (k$)
61 62 1 19 46
125 126 0 31 70
180 181 0 37 97
154 155 0 47 78
80 81 1 57 54
.. ... ... ... ...
67 68 0 68 48
192 193 1 33 113
117 118 0 49 65
47 48 0 27 40
172 173 1 36 87

[190 rows x 4 columns] CustomerID Gender Age Annual Income (k$) 18 19 1
52 23
170 171 1 40 87
107 108 1 54 63
98 99 1 48 61
177 178 1 27 88
182 183 1 46 98
5 6 0 22 17
146 147 1 48 77
12 13 0 58 20
152 153 0 44 78 61 55 125 77
180 32
154 16
80 51
       ..
67 48
192 8
117 59
47 47
172 10
Name: Spending Score (1-100), Length: 190, dtype: int64 18 29 170
13
107 46
98 42
177 69
182 15
5 76
146 36
12 15
152 20
```

13. Build the Model

```
Name: Spending Score (1-100), dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
logreg= LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
print (X_test) #test dataset
print (y_pred) #predicted values
```

```
     CustomerID Gender Age Annual Income (k$)
18 19 1 52 23
170 171 1 40 87
107 108 1 54 63
98 99 1 48 61
177 178 1 27 88
182 183 1 46 98
5 6 0 22 17
146 147 1 48 77
12 13 0 58 20
152 153 0 44 78 [ 4 1 48 55 86 75 81 5 4 5]
```

14. Train the Model

```
X_train
```

```
     CustomerID Gender Age Annual Income (k$) 61
62 1 19 46 125 126 0 31 70 180 181 0 37 97 154
155 0 47 78 80 81 1 57 54 .. ... ... ... ... 67
68 0 68 48 192 193 1 33 113 117 118 0 49 65 47
48 0 27 40 172 173 1 36 87
```

```
[190 rows x 4 columns]
```

```
X_test
```

```
     CustomerID Gender Age Annual Income (k$) 18
19 1 52 23 170 171 1 40 87 107 108 1 54 63 98 99
1 48 61 177 178 1 27 88 182 183 1 46 98 5 6 0 22
17 146 147 1 48 77 12 13 0 58 20 152 153 0 44 78
```

```
y_train
```

```
61 55
125 77
180 32
154 16
80 51
        ..
67 48
192 8
117 59
47 47
172 10
Name: Spending Score (1-100), Length: 190, dtype: int64
```

```
y_test
```

```
18 29
170 13
107 46
98 42
177 69
182 15
5 76
146 36
12 15
152 20
Name: Spending Score (1-100), dtype: int64
```

```
# Select algorithm
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
model = DecisionTreeClassifier()
# Fit model to the data
model.fit(X_train, y_train)
```

15. Test the Model

```
# Check model performance on training data
predictions = model.predict(X_train)
print(accuracy_score(y_train, predictions))
```

```
1.0
```

16. Measure the performance using Evaluation Metrics.

```
# Evaluate the model on the test data
```

```
predictions = model.predict(X_test)
predictions
```

```
array([14, 95, 56, 49, 75, 28, 77, 16, 14, 28])
```

```
print(accuracy_score(y_test, predictions)) 0.0
```

```
df = X_test.copy()
df['Actual'] = y_test
df['Prediction'] = predictions
df
```

CustomerID Gender Age Annual Income (k$) Actual Prediction 18 19 1 52 23 29 14 170 171 1 40 87 13 95 107 108 1 54 63 46 56 98 99 1 48 61 42 49 177 178 1 27 88 69 75 182 183 1 46 98 15 28 5 6 0 22 17 76 77 146 147 1 48 77 36 16 12 13 0 58 20 15 14 152 153 0 44 78 20 28

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
X_actual = [0, 1, 0, 1, 0, 0, 1, 0, 0, 0]
Y_predic = [1, 0, 1, 1, 1, 0, 1, 1, 0, 0]
results = confusion_matrix(X_actual, Y_predic)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))
print ('Classification Report : ')
print (classification_report(X_actual, Y_predic))
print('AUC-ROC:',roc_auc_score(X_actual, Y_predic))
print('LOGLOSS Value is',log_loss(X_actual, Y_predic))
```

```
Confusion Matrix :
[[3 4]
 [1 2]]
Accuracy Score is 0.5
Classification Report :
              precision recall f1-score support

           0 0.75 0.43 0.55 7
           1 0.33 0.67 0.44 3

    accuracy 0.50 10
   macro avg 0.54 0.55 0.49 10
weighted avg 0.62 0.50 0.52 10
```