## Q1. Apply K-Means Clustering technique of machine learning to solve the given problem.

We have given a collection of 8 points. P1=[0.1,0.6] P2=[0.15,0.71] P3=[0.08,0.9]

P4=[0.16, 0.85] P5=[0.2,0.3] P6=[0.25,0.5] P7=[0.24,0.1] P8=[0.3,0.2]. Perform the k- mean clustering with initial centroids as m1=P1 =Cluster#1=C1 and m2=P8=cluster#2=C2.

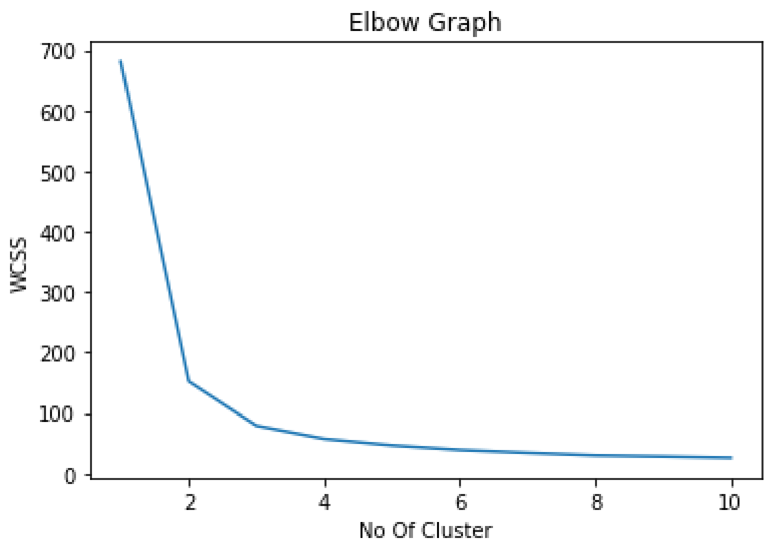
```
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

x = np.array([[0.1,0.6], [0.15,0.71], [0.08,0.9], [0.16,0.85], [0.2,0.3], [0.25,0.5], [0.2
centroid = np.array([[0.1,0.6], [0.15,0.71]])

km = KMeans(n_clusters=2, init=centroid, random_state=0)
km.fit(x)

# Q1. Which cluster does P6 belongs to?
print("P6 belongs to", km.labels_[5])

# Q2. What is the population of cluster around m2?
print("Population at m2", sum(km.labels_ == 1))
print("Population at m1", sum(km.labels_ == 0))

# Q3. What is updated value of m1 and m2?
print("Updated Value", km.cluster_centers_[0],km.cluster_centers_[1])

# Q4. What is the best value of K for the given problem?
wcss = []
for i in range(1, 9):
  km = KMeans(n_clusters=i, init='k-means++')
  km.fit(x)
  wcss.append(km.inertia_)
plt.plot(range(1,9), wcss)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/cluster/_kmeans.py:1146: RuntimeWarnir
  self._check_params(X)
P6 belongs to 0
Population at m2 4
Population at m1 4
Updated Value [0.2475 0.275 ] [0.1225 0.765 ]
[<matplotlib.lines.Line2D at 0x7f25f6a60b90>]
```

## Q2. Apply K-Means Clustering technique of machine learning to analyze the Iris dataset.

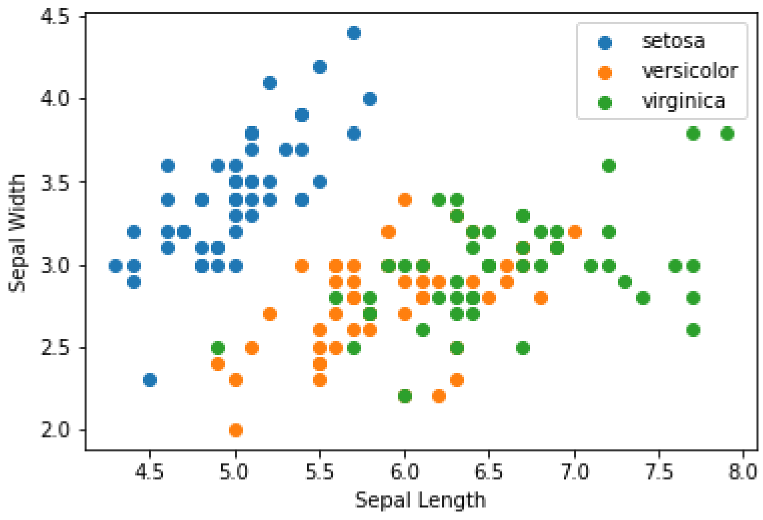
Use Elbow method to find best value of K.

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

dataset = load_iris()

x = dataset.data
y = dataset.target

wcss = []
for i in range(1, 11):
  km = KMeans(n_clusters=i, init='k-means++', random_state=0)
  km.fit(x,y)
  wcss.append(km.inertia_)

plt.plot(range(1,11), wcss)
plt.title("Elbow Graph")
plt.xlabel('No Of Cluster')
plt.ylabel('WCSS')
plt.show()
```



```
# kmns = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
# y_kmns = kmns.fit_predict(x)

# # Before Prediction
# n_classes=len(dataset.target_names)
# for i in range(n_classes):
#   index = np.where(y == i)
#   plt.scatter(x[index, 0], x[index, 1],
#   label=dataset.target_names[i])
# plt.legend()
# plt.xlabel("Sepal Length")
# plt.ylabel("Sepal Width")
```

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



```
# #After Prediction
# plt.scatter(x[y_kmns == 0, 0], x[y_kmns == 0, 1], c='red', label='Cluster 1')
# plt.scatter(x[y_kmns == 1, 0], x[y_kmns == 1, 1], c='blue', label='Cluster 2')
# plt.scatter(x[y_kmns == 2, 0], x[y_kmns == 2, 1], c='green', label='Cluster 3')
# # Plotting the centroids of the clusters
# plt.scatter(kmns.cluster_centers_[:, 0], kmns.cluster_centers_[:, 1], s=100, c='yellow',
# plt.xlabel("Sepal Length")
# plt.ylabel("Sepal Width")
# plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f10df631e90>
```

## Q3. Apply K-Means Clustering technique of machine learning to analyze the Bostan dataset.

Use Elbow method to find best value of K.

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

dataset = load_boston()

x = dataset.data
y = dataset.target

wcss = []
for i in range(1, 9):
  km = KMeans(n_clusters=i, init='k-means++', random_state=0)
  km.fit(x,y)
  wcss.append(km.inertia_)

plt.plot(range(1,9), wcss)
plt.title("Elbow Graph")
plt.xlabel('No Of Cluster')
plt.ylabel('WCSS')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarnin

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
```

```
print("Best value of k is:",4)
```

```
Best value of k is: 4
```

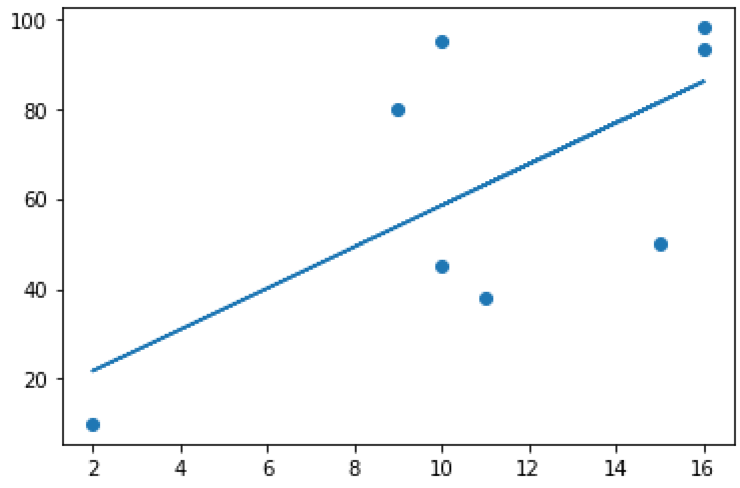
## Q4. Apply Linear Regression technique to solve the given problem.

The following table shows the results of a recently conducted study on the correlation of the number of hours spent driving with the risk of developing acute backache. Find the equation of the best fit line for this data.

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

x = pd.DataFrame([10, 9, 2, 15, 10, 16, 11, 16])
y = pd.DataFrame([95, 80, 10, 50, 45, 98, 38, 93])

lr = LinearRegression()
lr.fit(x,y)
```

```
print("Coeficient -", lr.coef_[0][0])
print("intercept -", lr.intercept_[0])

x = np.array(x)
y = np.array(y)
plt.scatter(x,y)
plt.plot(x, lr.predict(x))
```

```
Coeficient - 4.587898609975469
intercept - 12.584627964022907
[<matplotlib.lines.Line2D at 0x7f8981b31350>]
```



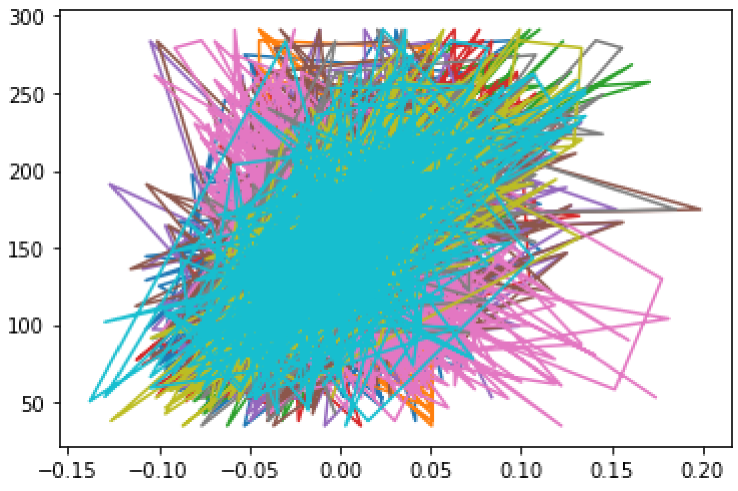
## Q5. Apply Linear Regression technique of machine learning to analyze the Diabetes dataset.

Display accuracy of the model. Find the equation of the best fit line for this data.

```
from sklearn.linear_model import LinearRegression
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.metrics import accuracy_score, r2_score
import numpy as np
import matplotlib.pyplot as plt

dataset = load_diabetes()

x = dataset.data
y = dataset.target

lr = LinearRegression()
lr.fit(x,y)

print("Score -", lr.score(x,y))
print("Coefficient -", lr.coef_.mean())
print("Intercept -", lr.intercept_)
print("R2 Score -", r2_score(y, lr.predict(x)))
print("y =", lr.coef_.mean(), 'x +', lr.intercept_)
```

```
plt.plot(x,lr.predict(x))
```

```
Score - 0.5177494254132934
Coefficient - 137.59740575237757
Intercept - 152.1334841628965
R2 Score - 0.5177494254132934
y = 137.59740575237757 x + 152.1334841628965
[<matplotlib.lines.Line2D at 0x7f897d40e150>,
 <matplotlib.lines.Line2D at 0x7f897d40e390>,
 <matplotlib.lines.Line2D at 0x7f897d40e550>,
 <matplotlib.lines.Line2D at 0x7f897d40e710>,
 <matplotlib.lines.Line2D at 0x7f897d40e8d0>,
 <matplotlib.lines.Line2D at 0x7f897d40ea90>,
 <matplotlib.lines.Line2D at 0x7f897d40ecd0>,
 <matplotlib.lines.Line2D at 0x7f897d40ee90>,
 <matplotlib.lines.Line2D at 0x7f897d40e890>,
 <matplotlib.lines.Line2D at 0x7f897d40ead0>]
```



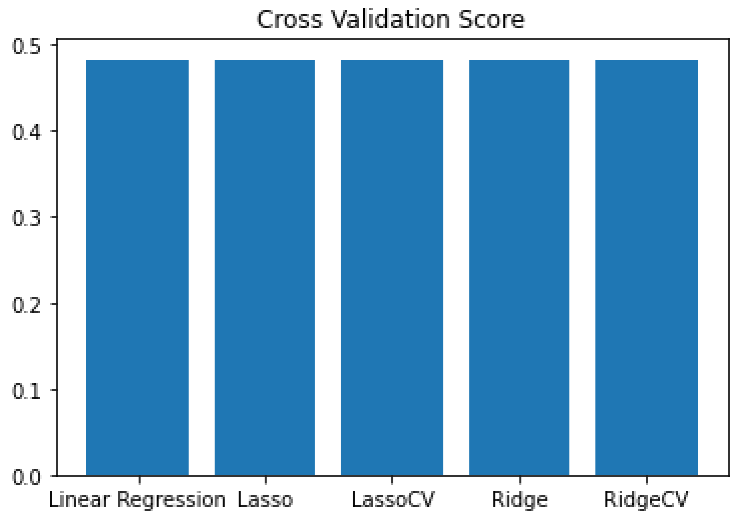
## Q6. Apply Linear, Ridge, Lasso Regression technique of machine learning to analyze and build the model of the Diabetes dataset.

Display and compare the accuracy (Cross-Validation, R2 Score) of all the models.

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LassoCV, RidgeCV
from sklearn.datasets import load_diabetes
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

dataset = load_diabetes()
x = dataset.data
y = dataset.target

lr = LinearRegression()
lr.fit(x,y)
print(r2_score(y, lr.predict(x)))

l = Lasso(alpha=0.001)
l.fit(x,y)
print(r2_score(y, l.predict(x)))
```

```
lc = LassoCV(alphas=[0.001, 0.004, 0.004, 0.6, 0.34])
lc.fit(x,y)
print(r2_score(y, lc.predict(x)))

r = Ridge(alpha=0.001)
r.fit(x,y)
print(r2_score(y, r.predict(x)))
rc = RidgeCV(alphas=[0.001, 0.004, 0.004, 0.6, 0.34])
rc.fit(x,y)
print(r2_score(y, rc.predict(x)))

models = ['Linear Regression', 'Lasso', 'LassoCV', 'Ridge', 'RidgeCV']
accuracy = []
accuracy.append(cross_val_score(lr, x,y, cv=5).mean())
accuracy.append(cross_val_score(l, x,y, cv=5).mean())
accuracy.append(cross_val_score(lc, x,y, cv=5).mean())
accuracy.append(cross_val_score(r, x,y, cv=5).mean())
accuracy.append(cross_val_score(rc, x,y, cv=5).mean())

plt.bar(models, accuracy)
plt.title('Cross Validation Score')
```

```
0.5177494254132934
0.5177149949908342
0.5174120091300026
0.5177078692545862
0.517362528193229
Text(0.5, 1.0, 'Cross Validation Score')
```



## Q7. Apply Decision Tree Classification technique to solve given problem:

A dataset collected in a cosmetics shop showing details of customers and whether or not they responded to a special offer to buy a new lip-stick is shown in table below. Use this dataset to build a decision tree, with Buys as the target variable, to help in buying lip-sticks in the future. Find the root node of decision tree. According to the decision tree you have made from previous training data set, what is the decision for the test data: [Age < 21, Income = Low, Gender = Female, Marital Status = Married]?

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from IPython.display import Image

df = pd.read_csv('data.csv')
df = df.iloc[:,1:]

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

le = LabelEncoder();
x = x.apply(le.fit_transform)

## If data is this -
print("Age:",list( zip(df.iloc[:,0], x.iloc[:,0])))
print("\nIncome:",list( zip(df.iloc[:,1], x.iloc[:,1])))
print("\nGender:",list( zip(df.iloc[:,2], x.iloc[:,2])))
print("\nmaritialStatus:",list( zip(df.iloc[:,3], x.iloc[:,3])))

dt = DecisionTreeClassifier()
dt.fit(x,y)

query=np.array([1,1,0,0])
pred=dt.predict([query])
print("Model Predicted -",pred[0])

export_graphviz(dt, out_file="data2.dot", class_names=["NO", "YES"])
!dot -Tpng data2.dot -o tree2.png
Image('tree2.png')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)


        5
```

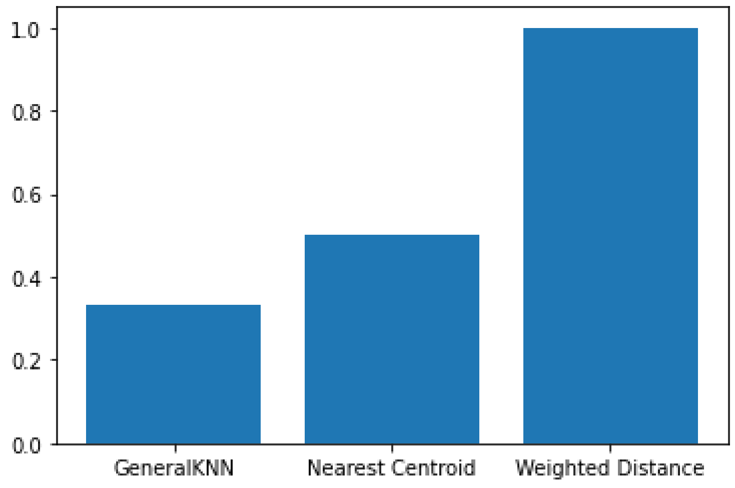
## Q8. Apply k-NN Classification technique to solve given problem:

In the following diagram let blue circles indicate positive examples and orange squares indicate negative examples. We want to use k-NN algorithm for classifying the points. If k=3, find the class of the point (6,6).

```
    705                 encoding=ioargs.encoding,
```

```
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
# Negative = 1, Positive = 0
x = pd.DataFrame([[2, 4], [4, 2], [4, 4], [4, 6], [6, 2], [6, 4]])
y = pd.DataFrame([1, 1, 0, 1, 0, 1])

accuracy = []

# GeneralKNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x,y)
y_preds = knn.predict(x)
accuracy.append(accuracy_score(y, y_preds))
print("Confusion Matrx: ", confusion_matrix(y, y_preds))
print("Accuracy: ", accuracy_score(y, y_preds))

# Nearest Centroid
nc = NearestCentroid()
nc.fit(x,y)
y_preds = nc.predict(x)
accuracy.append(accuracy_score(y, y_preds))
print("Confusion Matrx: ", confusion_matrix(y, y_preds))
print("Accuracy: ", accuracy_score(y, y_preds))

# Distance
knnd = KNeighborsClassifier(n_neighbors=3, weights='distance')
knnd.fit(x,y)
y_preds = knnd.predict(x)
accuracy.append(accuracy_score(y, y_preds))
print("Confusion Matrx: ", confusion_matrix(y, y_preds))
print("Accuracy: ", accuracy_score(y, y_preds))

labels = ['GeneralKNN', 'Nearest Centroid', 'Weighted Distance']
plt.bar(labels, accuracy)

# Q. find the class of the point (6,6).
print('(6, 6) belongs to - ',knnd.predict([[6,6]])[0])
```

```
Confusion Matrx:  [[0 2]
 [2 2]]
Accuracy:  0.3333333333333333
Confusion Matrx:  [[1 1]
 [2 2]]
Accuracy:  0.5
Confusion Matrx:  [[2 0]
 [0 4]]
Accuracy:  1.0
(6, 6) belongs to -  1
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198: Data
  return self._fit(X, y)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversio
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198: Data
  return self._fit(X, y)
```



## Q9. Consider the following training data set. Write a program to construct a decision tree using ID3 algorithm.

Display Accuracy measures for the same and predict a class of suitable query.

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from sklearn.metrics import accuracy_score
from IPython.display import Image

df = pd.read_csv('data1.csv')

x = df.iloc[:,:-1]
x1 = x
y = df.iloc[:,-1]
le = LabelEncoder()
x = x.apply(le.fit_transform)
y = le.fit_transform(y)

print("Day", set( zip(df.iloc[:,0], x.iloc[:,0])))
print("Out Look", set( zip(df.iloc[:,1], x.iloc[:,1])))
print("Tempreture", set( zip(df.iloc[:,2], x.iloc[:,2])))
print("Humidity", set( zip(df.iloc[:,3], x.iloc[:,3])))
print("Wind", set( zip(df.iloc[:,4], x.iloc[:,4])))



dt = DecisionTreeClassifier(criterion='entropy', max_depth=8, splitter='best')
dt.fit(x,y)

x_test = np.array([1,2,1,1,2])

pred = dt.predict(x)
pred1 = dt.predict([x_test])
print(pred1)
print("Accuracy = ", accuracy_score(y, pred)*100)

export_graphviz(dt, out_file="data2.dot")
!dot -Tpng data2.dot -o tree2.png
Image('tree2.png')
```

```
Day {('D11', 2), ('D14', 5), ('D1', 0), ('D7', 11), ('D12', 3), ('D2', 6), ('D6', 10)
Out Look {('Rain', 1), ('Overcast', 0), ('Sunny', 2)}
Tempreture {('Cool', 0), ('Hot', 1), ('Mild', 2)}
Humidity {('Normal', 1), ('High', 0)}
Wind {('Weak', 1), ('Strong', 0)}
[1]
Accuracy =  100.0
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not h
```

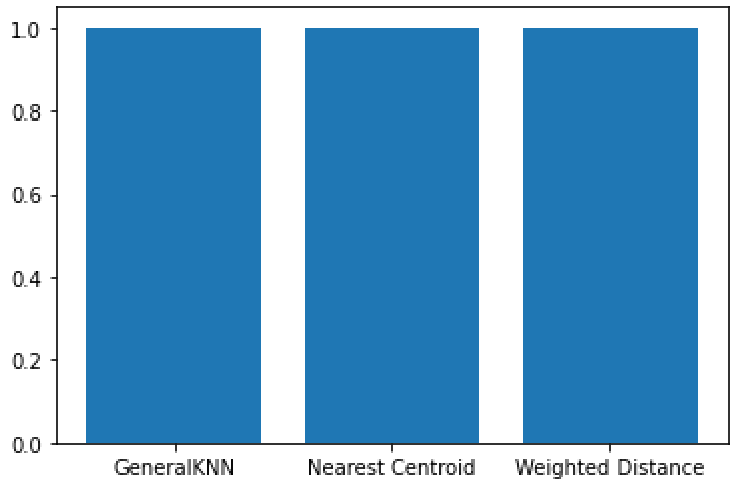
## Q10. Consider tissue paper factory application.

Apply KNN algorithm to find class of new tissue paper (X1= 3, X2=7). Assume K=3

| value = [5, 9] |

```
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix

x = pd.DataFrame([[7,7], [7,4], [3,4], [1,4]])
#Good = 1, Bad = 0
y = pd.DataFrame([0,0,1,1])
accuracy = []
# GeneralKNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x,y)
y_preds = knn.predict(x)
accuracy.append(accuracy_score(y, y_preds))
print("Confusion Matrx: ", confusion_matrix(y, y_preds))
print("Accuracy: ", accuracy_score(y, y_preds))

# Nearest Centroid
nc = NearestCentroid()
nc.fit(x,y)
y_preds = nc.predict(x)
accuracy.append(accuracy_score(y, y_preds))
print("Confusion Matrx: ", confusion_matrix(y, y_preds))
print("Accuracy: ", accuracy_score(y, y_preds))

# Distance
knnd = KNeighborsClassifier(n_neighbors=3, weights='distance')
knnd.fit(x,y)
y_preds = knnd.predict(x)
accuracy.append(accuracy_score(y, y_preds))
print("Confusion Matrx: ", confusion_matrix(y, y_preds))
print("Accuracy: ", accuracy_score(y, y_preds))

labels = ['GeneralKNN', 'Nearest Centroid', 'Weighted Distance']
plt.bar(labels, accuracy)

# Q. find the class of the point (3, 7).
print('(3, 7) belongs to - ',knnd.predict([[3,7]])[0])
```

```
Confusion Matrx:  [[2 0]
 [0 2]]
Accuracy:  1.0
Confusion Matrx:  [[2 0]
 [0 2]]
Accuracy:  1.0
Confusion Matrx:  [[2 0]
 [0 2]]
Accuracy:  1.0
(3, 7) belongs to -  1
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198: Data
  return self._fit(X, y)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversio
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neighbors/_classification.py:198: Data
  return self._fit(X, y)
```



```
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import pandas as pd
data = {
    'age': ['<21', '<21', '21-35', '>35', '>35', '>35', '21-35', '<21', '<21', '>35', '<21
    'income':['high','high','high','medium','low','low','low','medium','low','medium','med
    'gender':['male','male','male','male','female','female','female','male','female','fema
    'marital_status':['single', 'married', 'single', 'single', 'single', 'married', 'marri
    'buys':['no','no','yes','yes','yes','no','yes','no','yes','yes','yes','yes','yes','no'
}
df = pd.DataFrame.from_dict(data)
df
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()
x = df.iloc[:,[0,1,2,3]]
y = df.iloc[:,-1]
x['age'] = lb.fit_transform(x['age'])
x['income'] = lb.fit_transform(x['income'])
x['gender'] = lb.fit_transform(x['gender'])
x['marital_status'] = lb.fit_transform(x['marital_status'])
x
model = DecisionTreeClassifier()
model.fit(x,y)
model.predict(np.array([[1,1,0,0]]))[0]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not h
  "X does not have valid feature names, but"
'yes'
```

✓ 1s completed at 8:50 PM