NAME : HAMMAD ABID

I'd : 9134

In [53]:
```python

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

In [54]:
```python
#Reading csv_file
df = pd.read_csv("carprices.csv")
```

In [16]:
```python
df.head()
```

Out[16]:

	Car Model	Mileage	Sell Price($)	Age(yrs)
0	BMW X5	69000	18000	6
1	BMW X5	35000	34000	3
2	BMW X5	57000	26100	5
3	BMW X5	22500	40000	2
4	BMW X5	46000	31500	4

In [17]:
```python
#converting string data to numeric using one hot Encoding
carmodel = pd.get_dummies(df[['Car Model']], drop_first= True)
carmodel.head()
```

Out[17]:

	Car Model_BMW X5	Car Model_Mercedez Benz C class
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

In [18]:
```
#Now merging
data = pd.concat([df, carmodel ], axis = 1)
data.head()
```

Out[18]:

	Car Model	Mileage	Sell Price($)	Age(yrs)	Car Model_BMW X5	Car Model_Mercedez Benz C class
0	BMW X5	69000	18000	6	1	0
1	BMW X5	35000	34000	3	1	0
2	BMW X5	57000	26100	5	1	0
3	BMW X5	22500	40000	2	1	0
4	BMW X5	46000	31500	4	1	0

In [19]:
```
categorical_features = ['Car Model']
```

In [20]:
```
data = data.drop(columns=categorical_features, axis=1)
data.head()
```

Out[20]:

	Mileage	Sell Price($)	Age(yrs)	Car Model_BMW X5	Car Model_Mercedez Benz C class
0	69000	18000	6	1	0
1	35000	34000	3	1	0
2	57000	26100	5	1	0
3	22500	40000	2	1	0
4	46000	31500	4	1	0

In [23]:
```
Y = data['Sell Price($)']
X = data.drop('Sell Price($)', axis=1)
```

In [24]:
```
from sklearn.model_selection import train_test_split

np.random.seed(0)
trainX, testX, trainY, testY = train_test_split(X,Y, test_size = 0.3, random_sta
```

In [25]:
```
#LinearRegression
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(trainX, trainY)
```

Out[25]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [26]:
```
Lm_predict = lm.predict(testX)
print("Prediction Using Linear Regression for test set: {}".format(Lm_predict))
```

```
Prediction Using Linear Regression for test set: [19630.09708738 26418.44660194
28458.25242718 11600.        ]
```

In [27]:
```
data_diff = pd.DataFrame({'Actual value': testY, 'Predicted value': Lm_predict})
data_diff.head()
```

Out[27]:

	Actual value	Predicted value
10	20000	19630.097087
5	29400	26418.446602
9	22000	28458.252427
8	12000	11600.000000

In [28]:
```
#Performance Measurement of Linear Regression
from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(testY, Lm_predict)
meanSqErr = metrics.mean_squared_error(testY, Lm_predict)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(testY, Lm_predict))
print('R squared: {:.2f}'.format(lm.score(X,Y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 91.64
Mean Absolute Error: 2552.4271844660216
Mean Square Error: 12723878.310868148
Root Mean Square Error: 3567.0545707723827
```

In [29]:
```
#Implementing Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision = DecisionTreeClassifier(random_state=1)
decision.fit(trainX, trainY)
```

Out[29]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best')
```

In [30]:
```
descision_predict = decision.predict(testX)
print("Prediction Using Decision for test set: {}".format(descision_predict))
```

```
Prediction Using Decision for test set: [21000 26100 33000 19300]
```

In [31]:
```
data_diff = pd.DataFrame({'Actual value': testY, 'Predicted value':descision_pre
data_diff.head()
```

Out[31]:

	Actual value	Predicted value
10	20000	21000
5	29400	26100
9	22000	33000
8	12000	19300

In [32]:
```
#Performance Measurement of Decision
from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(testY,descision_predict)
meanSqErr = metrics.mean_squared_error(testY, descision_predict)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(testY, descision_predict))
print('R squared: {:.2f}'.format(decision.score(X,Y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 69.23
Mean Absolute Error: 5650.0
Mean Square Error: 46545000.0
Root Mean Square Error: 6822.389610686273
```

In [33]:
```
# Implementing KNN
from sklearn.neighbors import KNeighborsClassifier
kc = KNeighborsClassifier()
kc.fit(trainX, trainY)
```

Out[33]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [34]:
```
KN_predict = kc.predict(testX)
print("Prediction Using KNN for test set: {}".format(KN_predict))
```

```
Prediction Using KNN for test set: [18000 18000 18000 18000]
```

In [35]:
```
data_diff = pd.DataFrame({'Actual value': testY, 'Predicted value':KN_predict})
data_diff.head()
```

Out[35]:

	Actual value	Predicted value
10	20000	18000
5	29400	18000
9	22000	18000
8	12000	18000

In [36]:
```
#Performance Measurement of KNN
from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(testY,KN_predict)
meanSqErr = metrics.mean_squared_error(testY, KN_predict)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(testY, KN_predict))
print('R squared: {:.2f}'.format(kc.score(X,Y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 7.69
Mean Absolute Error: 5850.0
Mean Square Error: 46490000.0
Root Mean Square Error: 6818.357573492314
```

In [37]:
```
#now calculating the accuracy
print(f'Linear Model Test Accuracy: {lm.score(trainX, trainY)}')
print(f'Decision Model Test Accuracy: {decision.score(trainX, trainY)}')
print(f'KNN Model Test Accuracy: {kc.score(trainX, trainY)}')
```

```
Linear Model Test Accuracy: 0.9709843743171688
Decision Model Test Accuracy: 1.0
KNN Model Test Accuracy: 0.1111111111111111
```

In [38]:
```
print(f'Linear Model Accuracy: {lm.score(testX,Lm_predict)*100}%')
print(f'Decision Tree Model Accuracy: {decision.score(testX, descision_predict)*
print(f'KNN Model Accuracy: {kc.score(testX,KN_predict)*100}%')
```

```
Linear Model Accuracy: 100.0%
Decision Tree Model Accuracy: 100.0%
KNN Model Accuracy: 100.0%
```

ANSWER NO "3D Objects"

#as above data shows that :

#linear model:best fit we can accept this

#Decision model:best fit we can accept this

#KNN model:overifit

Question No 2

In [44]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [45]:
```
df = pd.read_csv("Q2data.csv")
df
```

Out[45]:

	x1	x2	x3	x4
0	6.5	8.6	NaN	4
1	1.2	5.0	5.2	9
2	4.0	2.0	8.7	5
3	3.2	3.6	7.3	1
4	5.2	7.1	NaN	3
5	7.3	2.0	6.4	8
6	5.5	4.1	NaN	5

In [46]:
```
print("Correlation of X1 and X3:",df.x1.corr(df.x3))
print("Correlation of X2 and X3:",df.x2.corr(df.x3))
print("Correlation of X4 and X3:",df.x4.corr(df.x3))
```

```
Correlation of X1 and X3: 0.24802925080387372
Correlation of X2 and X3: -0.6964857161414854
Correlation of X4 and X3: -0.6218855518005906
```

In [47]:
```
df1 = df
df1 = df1.dropna()
df1
```

Out[47]:

	x1	x2	x3	x4
1	1.2	5.0	5.2	9
2	4.0	2.0	8.7	5
3	3.2	3.6	7.3	1
5	7.3	2.0	6.4	8

In [48]:
```
train_X = np.array([1.2,4.0,3.2,7.3])
train_Y = np.array([5.2,8.7,7.3,6.4])
```

In [49]:
```
from sklearn.linear_model import LinearRegression
regression  = LinearRegression()
regression.fit(train_X.reshape(-1,1),train_Y)
```

Out[49]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [50]:
```
test_X = np.array([6.5,5.2,5.5])
```

In [51]:
```
test_Y = regression.predict(test_X.reshape(-1,1))
print(test_Y)
df
```

```
[7.27132704 7.08386096 7.12712237]
```

Out[51]:

	x1	x2	x3	x4
0	6.5	8.6	NaN	4
1	1.2	5.0	5.2	9
2	4.0	2.0	8.7	5
3	3.2	3.6	7.3	1
4	5.2	7.1	NaN	3
5	7.3	2.0	6.4	8
6	5.5	4.1	NaN	5

In [52]:
```
result = pd.DataFrame({'x1':[6.5,1.2,4.0,3.2,5.2,7.3,5.5],
                       'x2':[8.6,5.0,2.0,3.6,7.1,2.0,4.1],
                       'x3':[7.2,5.2,8.7,7.3,7.0,6.4,7.1],
                       'x4':[4,9,5,1,3,8,5]})
result
```

Out[52]:

	x1	x2	x3	x4
0	6.5	8.6	7.2	4
1	1.2	5.0	5.2	9
2	4.0	2.0	8.7	5
3	3.2	3.6	7.3	1
4	5.2	7.1	7.0	3
5	7.3	2.0	6.4	8
6	5.5	4.1	7.1	5