LAB3_SURIYA S_225229140

In [2]:
```
#STEP 1: creating fuel_data csv file
```

In [3]:
```
#STEP 2
#importing csv file
import pandas as pd
dat=pd.read_csv("fuel_data.csv")
```

In [4]:
```
#reading csv file
dat
```

Out[4]:

	driveKm	fuelAmount
0	390.00	3600.0
1	403.00	3705.0
2	396.50	3471.0
3	383.50	3250.5
4	321.10	3263.7
5	391.30	3445.2
6	386.10	3679.0
7	371.80	3744.5
8	404.30	3809.0
9	392.20	3905.0
10	386.43	3874.0
11	395.20	3910.0
12	381.00	4020.7
13	372.00	3622.0
14	397.00	3450.5
15	407.00	4179.0
16	372.40	3454.2
17	375.60	3883.8
18	399.00	4235.9

In [5]:
```
#printing using head()
dat.head()
```

Out[5]:

	driveKm	fuelAmount
0	390.0	3600.0
1	403.0	3705.0
2	396.5	3471.0
3	383.5	3250.5
4	321.1	3263.7

In [6]:
```
#printing shape
dat.shape
```

Out[6]:
```
(19, 2)
```

In [7]:
```
#printing columns
dat.columns
```

Out[7]:
```
Index(['driveKm', 'fuelAmount'], dtype='object')
```

In [8]:
```
#printing info
dat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19 entries, 0 to 18
Data columns (total 2 columns):
driveKm       19 non-null float64
fuelAmount    19 non-null float64
dtypes: float64(2)
memory usage: 384.0 bytes
```

In [9]:
```
dat.info
```

Out[9]:
```
<bound method DataFrame.info of     driveKm  fuelAmount
0    390.00      3600.0
1    403.00      3705.0
2    396.50      3471.0
3    383.50      3250.5
4    321.10      3263.7
5    391.30      3445.2
6    386.10      3679.0
7    371.80      3744.5
8    404.30      3809.0
9    392.20      3905.0
10   386.43      3874.0
11   395.20      3910.0
12   381.00      4020.7
13   372.00      3622.0
14   397.00      3450.5
15   407.00      4179.0
16   372.40      3454.2
17   375.60      3883.8
18   399.00      4235.9>
```

In [10]:
```
#printing type
type(dat)
```

Out[10]:
```
pandas.core.frame.DataFrame
```

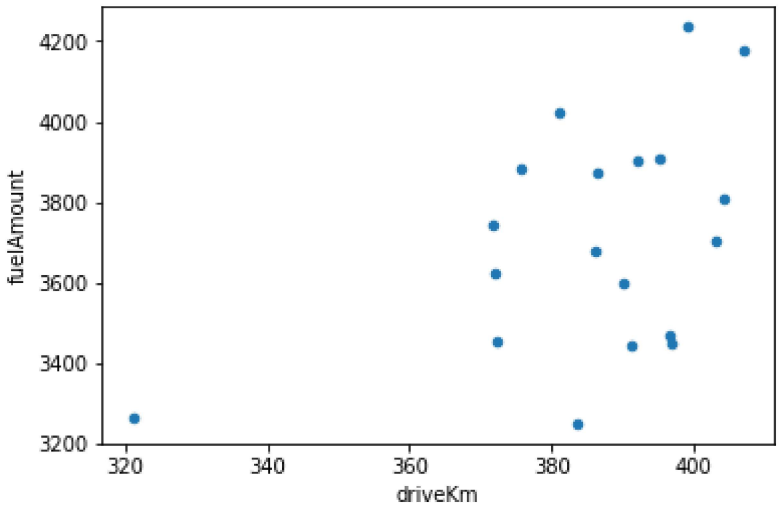
In [11]:
```
#STEP 3:
dat.isnull()
```

Out[11]:

	driveKm	fuelAmount
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False
7	False	False
8	False	False
9	False	False
10	False	False
11	False	False
12	False	False
13	False	False
14	False	False
15	False	False
16	False	False
17	False	False
18	False	False

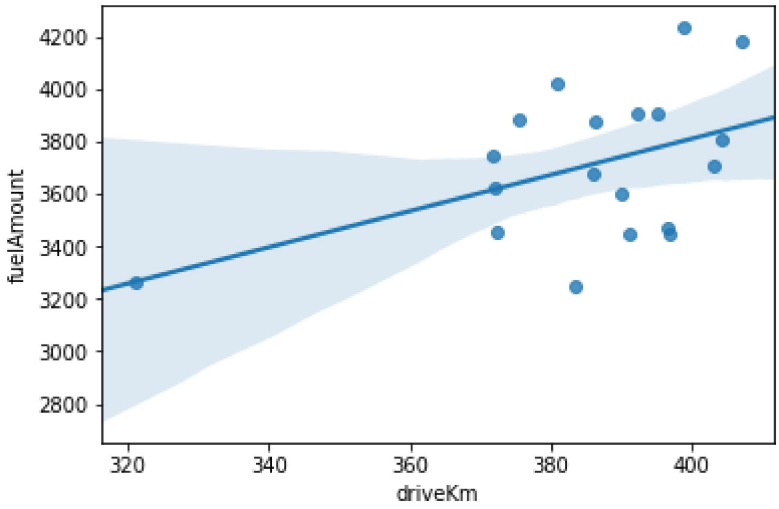
In [12]:
```
#STEP 4:
#visualize relationship
import matplotlib.pyplot as plt
import seaborn as sns
```

In [13]:
```
dat.plot(kind='scatter',x='driveKm',y='fuelAmount')
plt.show()
```



In [14]:
```
sns.regplot(data=dat,x="driveKm",y="fuelAmount")
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22627d94828>
```



In [15]:
```
#STEP 5: prepare X matrix and y vector
X=pd.DataFrame(dat['driveKm'])
```

In [16]:
```
y=pd.DataFrame(dat['fuelAmount'])
```

In [17]:
```
#STEP 6: Examine X and y
X
```

Out[17]:

	driveKm
0	390.00
1	403.00
2	396.50
3	383.50
4	321.10
5	391.30
6	386.10
7	371.80
8	404.30
9	392.20
10	386.43
11	395.20
12	381.00
13	372.00
14	397.00
15	407.00
16	372.40
17	375.60
18	399.00

In [18]:
```
y
```

Out[18]:

	fuelAmount
0	3600.0
1	3705.0
2	3471.0
3	3250.5
4	3263.7
5	3445.2
6	3679.0
7	3744.5
8	3809.0
9	3905.0
10	3874.0
11	3910.0
12	4020.7
13	3622.0
14	3450.5
15	4179.0
16	3454.2
17	3883.8
18	4235.9

In [19]:
```
type(X)
```

Out[19]:
```
pandas.core.frame.DataFrame
```

In [20]:
```
type(y)
```

Out[20]:
```
pandas.core.frame.DataFrame
```

In [21]:
```
#STEP 7: split dataset
#split into train test sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random_state=42)
```

In [22]:
```
X_train.shape
```

Out[22]:
```
(15, 1)
```

In [23]:
```
y_train.shape
```

Out[23]:
```
(15, 1)
```

In [24]:
```
X_test.shape
```

Out[24]:
```
(4, 1)
```

In [25]:
```
y_test.shape
```

Out[25]:
```
(4, 1)
```

Part-I. Linear Regression Baseline Model

In [26]:
```
#STEP 8: Build Model
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(X_train,y_train)
```

Out[26]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [27]:
```
#STEP 9:Predict price for 800km
pred_800_KM=reg.predict([[800]])
print("Deisel price for 800KM:",pred_800_KM[0])
```

```
Deisel price for 800KM: [6905.64571567]
```

In [28]:
```
#step 10: predict on entire dataset
y_pred=reg.predict(X_test)
```

In [29]:
```
y_pred
```

Out[29]:
```
array([[3775.81615646],
       [3785.74000628],
       [3815.51155575],
       [3875.05465468]])
```

In [30]:
```
#STEP 11:Print MSE and R2 Error
import sklearn.metrics as metrics
mse=metrics.mean_squared_error(y_test,y_pred)
r2=metrics.r2_score(y_test,y_pred)
print("MSE: ",mse)
print("R2: ",r2)
print("MODEL PARAMETERS:")
print("coefficient:",reg.coef_)
print("Intercept:",reg.intercept_)
```

```
MSE:  46181.36710639155
R2:  -0.6180990161577022
MODEL PARAMETERS:
coefficient: [[7.63373063]]
Intercept: [798.6612099]
```

Part-II Linear Regression wiht scaling using standard scaler

In [31]:
```
#STEP 12:Normalize X_train and X_test values
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss_X_train=ss.fit_transform(X_train)
ss_X_train
```

Out[31]:
```
array([[ 1.0601947 ],
       [-0.5322439 ],
       [ 0.02186483],
       [-0.55221178],
       [ 1.19497791],
       [-0.37250084],
       [ 0.670821  ],
       [ 0.45616627],
       [ 0.79562026],
       [-3.09312478],
       [-0.10293443],
       [-0.56219572],
       [ 0.16812957],
       [ 0.69578085],
       [ 0.15165606]])
```

In [32]:
```
ss_X_test=ss.transform(X_test)
ss_X_test
```

Out[32]:
```
array([[0.34634292],
       [0.41123853],
       [0.60592538],
       [0.99529908]])
```

In [33]:
```
#STEP 13:Build LR model
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(ss_X_train,y_train)
```

Out[33]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

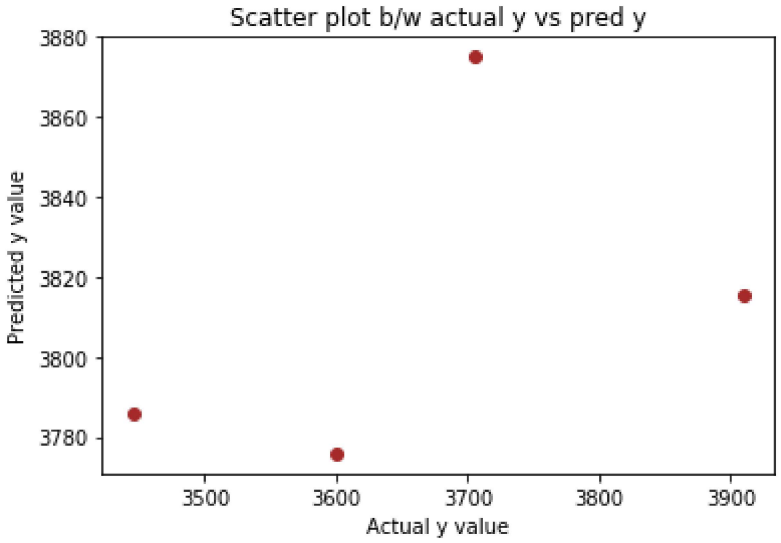
In [34]:
```
ss_y_pred=lr.predict(ss_X_test)
ss_y_pred
```

Out[34]:
```
array([[3775.81615646],
       [3785.74000628],
       [3815.51155575],
       [3875.05465468]])
```

In [35]:
```
#STEP 14: Print MSE and R2 Error
ss_mse=metrics.mean_squared_error(y_test,ss_y_pred)
ss_r2=metrics.r2_score(y_test,ss_y_pred)
print("SS_MSE: ",ss_mse)
print("SS_R2: ",ss_r2)
```

```
SS_MSE:  46181.36710639172
SS_R2:  -0.6180990161577082
```

In [36]:
```
#STEP 15:Plot scatter plot
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(y_test,y_pred,color='Brown',marker='o')
plt.title("Scatter plot b/w actual y vs pred y")
plt.xlabel('Actual y value')
plt.ylabel('Predicted y value')
plt.show()
```



Part-III:Linear Regression with scaling using MinMaxScaler and Comparison with KNeighborsRegressor and SGDRegressor

In [37]:
```
# STEP 16:Repeat with MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
mm=MinMaxScaler()
mm_X_train=mm.fit_transform(X_train)
mm_X_test=mm.transform(X_test)
mm_lr=LinearRegression()
mm_lr.fit(mm_X_train,y_train)
mm_y_pred=mm_lr.predict(mm_X_test)
print("Predictions of scaled data using MinMaxScaler:",mm_y_pred)

mm_mse=metrics.mean_squared_error(y_test,mm_y_pred)
mm_r2=metrics.r2_score(y_test,mm_y_pred)
print("MM_MSE: ",mm_mse)
print("MM_R2: ",mm_r2)
```

```
Predictions of scaled data using MinMaxScaler: [[3775.81615646]
 [3785.74000628]
 [3815.51155575]
 [3875.05465468]]
MM_MSE:  46181.3671063917
MM_R2:  -0.6180990161577073
```

In [38]:
```
#STEP 17:compare KNN Regressor
from sklearn.neighbors import KNeighborsRegressor
knr=KNeighborsRegressor()
knr.fit(X_train,y_train)
knr_y_pred=knr.predict(X_test)
print("Predictions of scaled data using KNeighborsRegressor:",knr_y_pred)
knr_mse=metrics.mean_squared_error(y_test,knr_y_pred)
knr_r2=metrics.r2_score(y_test,knr_y_pred)
print("KNR_MSE: ",knr_mse)
print("KNR_R2: ",knr_r2)
```

```
Predictions of scaled data using KNeighborsRegressor: [[3635.9 ]
 [3675.9 ]
 [3787.28]
 [3829.08]]
KNR_MSE:  21241.836200000045
KNR_R2:  0.2557302563733307
```

In [39]:
```
#STEP 18: compare SGD Regressor
from sklearn.linear_model import SGDRegressor
sgd=SGDRegressor()
sgd.fit(X_train, y_train)
sgd_y_pred=sgd.predict(X_test)
print("Predictions of scaled data using SGDRegressor:", sgd_y_pred)
sgd_mse=metrics.mean_squared_error(y_test, sgd_y_pred)
sgd_r2=metrics.r2_score(y_test, sgd_y_pred)
print("SGD_MSE:",sgd_mse)
print("SGD_R2:",sgd_r2)
```

```
Predictions of scaled data using SGDRegressor: [-2.66232311e+14 -2.67119745e+14
-2.69782047e+14 -2.75106650e+14]
SGD_MSE: 7.267465580438892e+28
SGD_R2: -2.5463687288808327e+24

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-pac
kages\sklearn\linear_model\stochastic_gradient.py:128: FutureWarning: max_iter
and tol parameters have been added in <class 'sklearn.linear_model.stochastic_g
radient.SGDRegressor'> in 0.19. If both are left unset, they default to max_ite
r=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From
0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-pac
kages\sklearn\utils\validation.py:578: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_sam
ples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [40]:
```
#STEP 19: [select best model]
# Storing the MSE in a dictionary
data_mse = {'lr_mse':[46181.36710639157],'ss_mse':[46181.36710639172],'mm_mse':[4
def best_model(data_mse):
    mse_min = min(data_mse.values())
    result = [key for key in data_mse if data_mse[key] == mse_min]
    Model_name = []
    if result == ['lr_mse']:
        a = 'LinearRegression'
        Model_name.append(a)
    elif result == ['ss_mse']:
        b = 'StandardScaler'
        Model_name.append(b)
    elif result == ['mm_mse']:
        c = 'MinMaxScaler'
        Model_name.append(c)
    elif result == ['knr_mse']:
        d = 'KNeighborsRegressor'
        Model_name.append(d)
    elif result == ['sgd_mse']:
        e = 'SGDRegressor'
        Model_name.append(e)
    print("The best model with the lowest MSE to be selected is", Model_name)
best_model(data_mse)
```

```
The best model with the lowest MSE to be selected is ['KNeighborsRegressor']
```

In [ ]: