PML_lab-3_205229118_Mahalakshmi.S

March 22, 2021

0.0.1 Lab3. Fuel Amount Prediction using Linear Regression

0.0.2 Step1. [Prepare your dataset]. Create fuel_data.csv file as shown above.

[2]:
```
import pandas as pd
import csv
```

[3]:
```
fuel=pd.read_csv("fuel_data.csv")
fuel
```

[3]:

	drivenKM	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

0.0.3 Step2. [Import dataset]. Using Pandas, import "fuel_data.csv" file and print properties such as head(), shape, columns, type and info.

[4]:
```
fuel.head()
```

[4]:

	drivenKM	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
```

[5]:
```
fuel.tail()
```

[5]:

	drivenKM	fuelAmount
14	397.0	3450.5
15	407.0	4179.0
16	372.4	3454.2
17	375.6	3883.8
18	399.0	4235.9

[6]:
```
fuel.shape
```

[6]: (19, 2)

[7]:
```
df = pd.read_csv("fuel_data.csv")
```

[8]:
```
df
```

[8]:

	drivenKM	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

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19 entries, 0 to 18
Data columns (total 2 columns):
```

```
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   drivenKM    19 non-null     float64
 1   fuelAmount  19 non-null     float64
dtypes: float64(2)
memory usage: 432.0 bytes
```

[10]:
```python
f = df.columns
f
```

[10]:
```
Index(['drivenKM', 'fuelAmount'], dtype='object')
```

0.0.4 Step3. [Preprocessing]. Check for missing values (Use isnull() method)

[11]:
```python
df.isnull()
```

[11]:

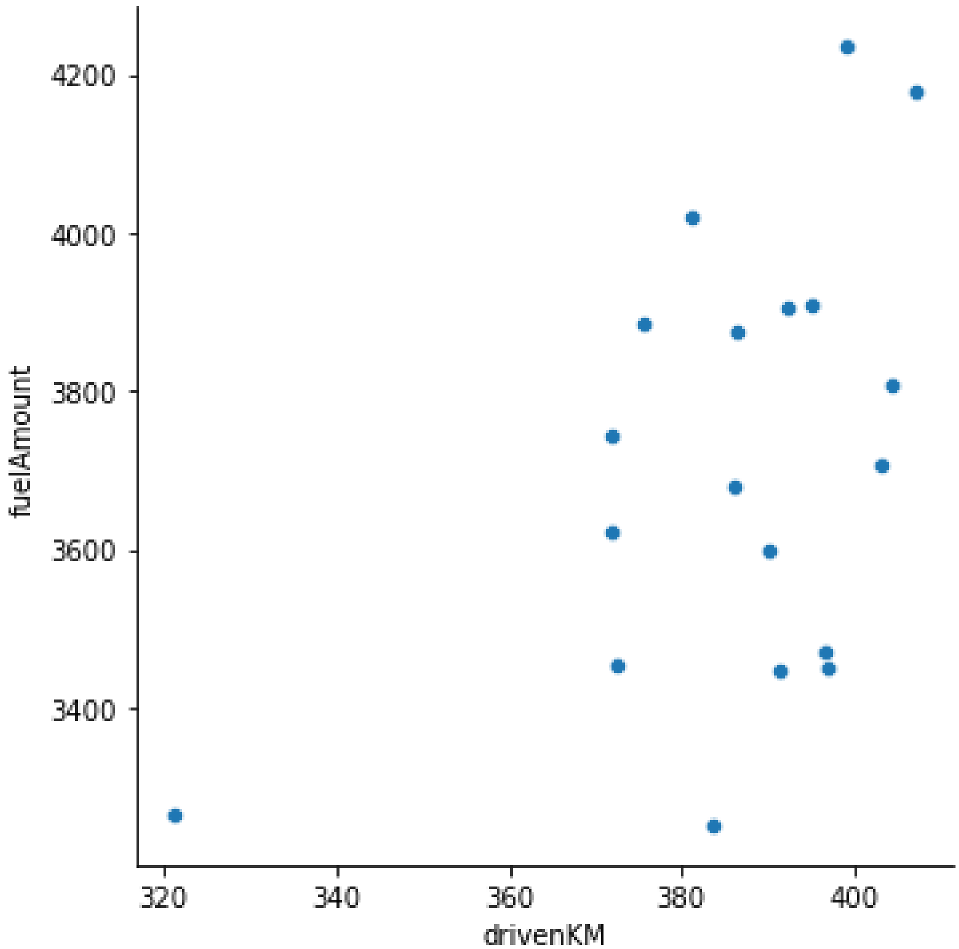
	drivenKM	fuelAmount
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False
7	False	False
8	False	False
9	False	False
10	False	False
11	False	False
12	False	False
13	False	False
14	False	False
15	False	False
16	False	False
17	False	False
18	False	False

0.0.5 Step4. [Visualize Relationships]. Plot relplot between "drivenKM" and "fuelAmount".

[12]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

[13]:
```python
sns.relplot(x="drivenKM", y="fuelAmount",data=fuel)
```

[13]:
```
<seaborn.axisgrid.FacetGrid at 0x1e0f1afa7f0>
```



0.0.6 Step5. [Prepare X matrix and y vector]. Extract "drivenKM" column and store into new dataframe X. Similarly, extract "fuelAmount" and store into y.

[14]:
```
data1 = ['drivenKM']
X=fuel[data1]
data2 = ['fuelAmount']
y=fuel.fuelAmount
```

0.0.7 Step6. [Examine X and y]. Print X, y, type of X and type of y.

[15]:
```
print(X)
X.dtypes
```

```
    drivenKM
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
```

[15]:
```
drivenKM    float64
dtype: object
```

[16]:
```python
print(y)
y.dtypes
```

```
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
Name: fuelAmount, dtype: float64
```

[16]:
```
dtype('float64')
```

0.0.8 Step7. [Split dataset]. Split dataset into 4 parts using train_test_split() method, such as X_train, X_test, y_train and y_test. Use 20% for test size. Later you can play around with this test size. Print the shape of all 4 parts.

[18]:
```
from sklearn.model_selection import train_test_split
```

[19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.
↪8,test_size=0.2)
```

[20]:
```
X_train, X_test, y_train, y_test
```

[20]:
```
(    drivenKM
 11    395.20
 9     392.20
 6     386.10
 5     391.30
 15    407.00
 8     404.30
 18    399.00
 13    372.00
 12    381.00
 1     403.00
 10    386.43
 17    375.60
 7     371.80
 16    372.40
 3     383.50,
     drivenKM
 14     397.0
 0      390.0
 4      321.1
 2      396.5,
 11    3910.0
 9     3905.0
 6     3679.0
 5     3445.2
 15    4179.0
 8     3809.0
 18    4235.9
 13    3622.0
 12    4020.7
 1     3705.0
 10    3874.0
 17    3883.8
 7     3744.5
 16    3454.2
 3     3250.5
 Name: fuelAmount, dtype: float64,
```

```
14    3450.5
0     3600.0
4     3263.7
2     3471.0
Name: fuelAmount, dtype: float64)
```

[21]:
```
X_train.shape
```

[21]:
```
(15, 1)
```

[22]:
```
X_test.shape
```

[22]:
```
(4, 1)
```

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

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

[24]:
```
y_test.shape
```

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

0.0.9 Part-I. Linear Regression Baseline Model

0.0.10 Step8. [Build Model]. Create Linear Regression model and train with fit() using X_train and y_train values.

[25]:
```
from sklearn.linear_model import LinearRegression
```

[26]:
```
#create a linear regression object
model = LinearRegression()
#train a model
model.fit( X_train,y_train)
```

[26]:
```
LinearRegression()
```

0.0.11 Step9. [Predict price for 800 KM]. If I need to travel 800 KM, how much do I need to spend on Diesel?. Are you getting this ouput, array([6905.64571567]). ?

[30]:
```
n=[[800]]
m=model.predict(n)
m
```

[30]:
```
array([7799.94281895])
```

0.0.12 Step10. [Predict on entire dataset]. Now, perform prediction using entire X__test and store result as y__pred.

[33]:
```
y_pred=model.predict(X_test)
y_pred
```

[33]:
```
array([3868.44701279, 3800.15800375, 3127.99904334, 3863.56922643])
```

0.0.13 Step11. [Print Mean Squared Error and R2 Error]. Are you getting output "MSE: 46181.0". Also, print values of model parameters: coef__ and intercept__ values.

[37]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

[38]:
```
mse_ln=mean_squared_error(y_test,y_pred)
mse_ln
```

[38]:
```
96817.06978545067
```

[39]:
```
r2_score(y_test,y_pred)
```

[39]:
```
-5.724087036950648
```

[40]:
```
model.coef_
```

[40]:
```
array([9.75557272])
```

[41]:
```
model.intercept_
```

[41]:
```
-4.515357046707777
```

0.0.14 Part-II. Linear Regression with Scaling using StandardScaler

0.0.15 Step12. [Normalize X__train and X__test values]. Use StandardScaler, scale X__train using fit__transform() method and X__test using transform() method.

[42]:
```
from sklearn.preprocessing import StandardScaler
```

[43]:
```
scaler = StandardScaler()
ss3=scaler.fit_transform(X_train)    # scale X_train using fit_transform()␣
 ↪method
print(ss3)
```

```
[[ 0.61206548]
 [ 0.35506309]
 [-0.16750845]
 [ 0.27796237]
 [ 1.62294157]
```

```
 [ 1.39163942]
 [ 0.93760185]
 [-1.37541971]
 [-0.60441252]
 [ 1.28027171]
 [-0.13923819]
 [-1.06701683]
 [-1.3925532 ]
 [-1.34115272]
 [-0.39024386]]
```

[44]:
```
ss5=scaler.transform(X_test)     #X_test using transform() method
print(ss5)
```

```
[[ 0.76626692]
 [ 0.16659466]
 [-5.73589369]
 [ 0.72343319]]
```

0.0.16 Step13. [Build LR model]. Create a new LR model, fit on scaled X_train and predict on scaled X_test.

[45]:
```
model1 = LinearRegression()
model1.fit(ss3,y_train)
```

[45]:
```
LinearRegression()
```

[46]:
```
s1_y_pred = model1.predict(ss5)
s1_y_pred
```

[46]:
```
array([3868.44701279, 3800.15800375, 3127.99904334, 3863.56922643])
```

0.0.17 Step14. [Print Mean Squared Error and R2 Error]. What is the output?. MSE reduced or not?. Why?.

[48]:
```
mean_squared_error(y_test,s1_y_pred)
```

[48]:
```
96817.06978545096
```

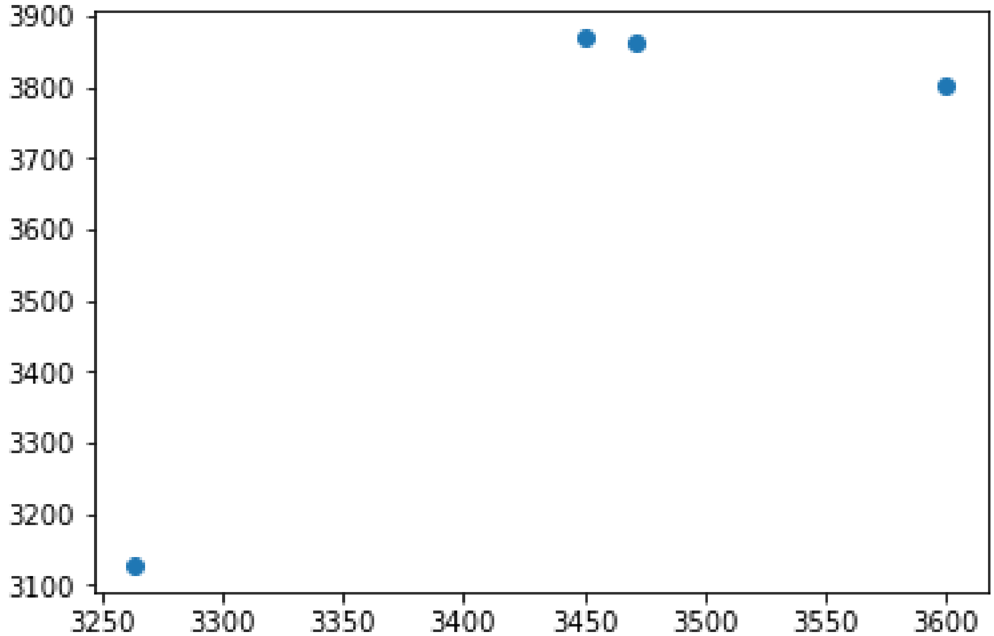
[49]:
```
r2_score(y_test, s1_y_pred)
```

[49]:
```
-5.724087036950668
```

0.0.18 Step15. [Plot scatter plot]. Display Scatter Plot between actual y (aka ground truth) vs predicted y values. That is, between y_test and y_pred.

[53]:
```
plt.scatter(y_test,y_pred)
```

[53]: <matplotlib.collections.PathCollection at 0x1e0f52aa700>



0.0.19 Part-III. Linear Regression with Scaling using MinMaxScaler and Comparison with KNeighborsRegressor and SGDRegressor

0.0.20 Step16. [Repeat with MinmaxScaler]. Repeat scaling using MinMaxScaler, LR model creation, fit, predict and error computation steps.

[57]:
```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler()
```

[58]:
```
mm_ss = mm_scaler.fit_transform(X_train)
mm_ss
```

[58]:
```
array([[0.66477273],
       [0.57954545],
       [0.40625   ],
       [0.55397727],
       [1.        ],
       [0.92329545],
       [0.77272727],
       [0.00568182],
       [0.26136364],
       [0.88636364],
       [0.415625  ],
       [0.10795455],
```

```
       [0.        ],
       [0.01704545],
       [0.33238636]])
```

[59]:
```
mm_ss5 = mm_scaler.transform(X_test)
mm_ss5
```

[59]:
```
array([[ 0.71590909],
       [ 0.51704545],
       [-1.44034091],
       [ 0.70170455]])
```

[60]:
```
model2 = LinearRegression()
model2.fit(mm_ss,y_train)
```

[60]:
```
LinearRegression()
```

[62]:
```
mms_y_pred = model2.predict(mm_ss5)
mms_y_pred
```

[62]:
```
array([3868.44701279, 3800.15800375, 3127.99904334, 3863.56922643])
```

[63]:
```
mean_squared_error(y_test,mms_y_pred)
```

[63]:
```
96817.06978545104
```

[64]:
```
r2_score(y_test,mms_y_pred)
```

[64]:
```
-5.724087036950673
```

0.0.21 Step17. [Compare KNN Regressor]. Repeat the above steps for KNeighborsRegressor model and compare MSE of LR with KNN Regressor.

[65]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[66]:
```
m_neig = KNeighborsRegressor(n_neighbors=5)
m_neig.fit(X, y)
```

[66]:
```
KNeighborsRegressor()
```

[67]:
```
n1_y_pred = m_neig.predict(X)
n1_y_pred
```

[67]:
```
array([3700.64, 3875.88, 3794.48, 3684.84, 3593.64, 3746.84, 3684.84,
       3745.04, 3875.88, 3666.24, 3569.74, 3794.48, 3741.6 , 3745.04,
       3794.48, 3875.88, 3745.04, 3745.04, 3754.48])
```

[68]:
```
mse=mean_squared_error(y,n1_y_pred)
mse
```

[68]: 70460.30507368421

[69]:
```
r2_score(y,n1_y_pred)
```

[69]: 0.06403925984775638

0.0.22 Step18. [Compare SGD Regressor]. Repeat the above steps for SGDRegressor model and compare MSE of LR with SGD Regressor.

[70]:
```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
```

[71]:
```
r = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
r.fit(X, y)
```

[71]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdregressor', SGDRegressor())])
```

[72]:
```
re_y_pred = r.predict(X)
re_y_pred
```

[72]:
```
array([3740.86178879, 3830.59155467, 3785.72667173, 3695.99690584,
       3265.29402959, 3749.83476537, 3713.94285902, 3615.24011655,
       3839.56453126, 3756.04682609, 3716.22061462, 3776.75369514,
       3678.74118163, 3616.62057448, 3789.17781657, 3858.20071341,
       3619.38149036, 3641.46881734, 3802.98239594])
```

[73]:
```
mse3=mean_squared_error(y,re_y_pred)
mse3
```

[73]: 58823.49485131113

[74]:
```
r2_score(y,re_y_pred)
```

[74]: 0.2186170394550624

0.0.23 Step19. [Select best model]. Tabulate MSE values of LR, KNNR and SGDR and select the model with the lowest MSE.

[76]:
```
print("LR model ",mse_ln)
print("KNNR model ",mse)
print("SGDR model ",mse3)
```

```
LR model  96817.06978545067
KNNR model  70460.30507368421
SGDR model  58823.49485131113
```