Step 2 : Importing Dataset

SURUTHI S
225229141

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

```
data = pd.read_csv("/content/fueldata.csv")
```

```
data.head()
```

	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

```
data.shape
```

```
(19, 2)
```

```
data.columns
```

```
Index(['drivenKm', 'fuelAmount'], dtype='object')
```

```
data.dtypes
```

```
drivenKm      float64
fuelAmount    float64
dtype: object
```

```
data.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
```

STEP 3: Pre Processing

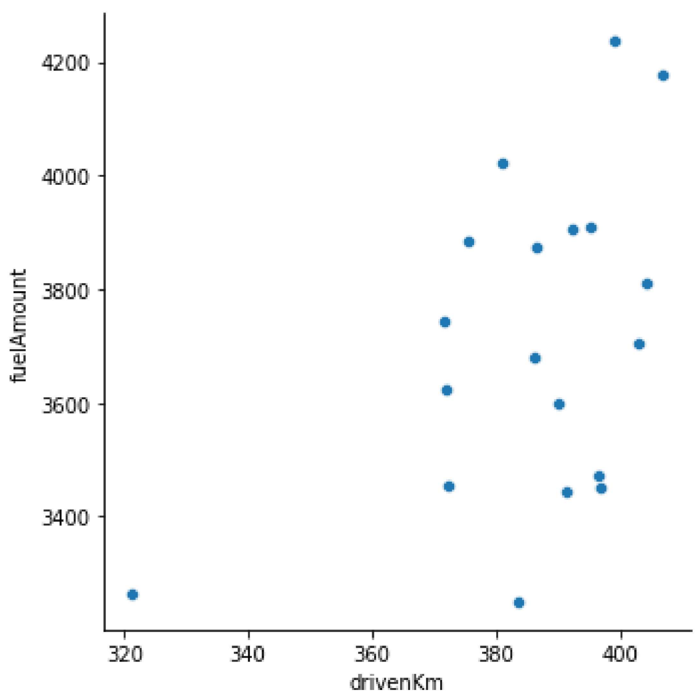
```
data.isnull().sum()
```

```
drivenKm      0
fuelAmount    0
dtype: int64
```

Step 4 : Vizualize Relationship

```
sns.relplot(data = data,x=data.drivenKm,y=data.fuelAmount)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f3d2795ffa0>
```



STEP 5 Prepare X Matrix and Y vector

```
feature_list = data[['drivenKm']]
```

```
feature_list
```

	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

```
label =data[['fuelAmount']]
```

```
label
```

	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

```
data.describe()
```

	drivenKm	fuelAmount
count	19.000000	19.000000
mean	385.548947	3710.684211
std	19.094297	281.892805
min	321.100000	3250.500000
25%	378.300000	3462.600000
50%	390.000000	3705.000000
75%	396.750000	3894.400000
max	407.000000	4235.900000

Step 6 Examine X and Y

```
print(feature_list)
print("Type of X Matrix",type(feature_list))
print(label)
print("Type of Y Vector ",type(label))
```

```
        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
Type of X Matrix <class 'pandas.core.frame.DataFrame'>
    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
Type of Y Vector  <class 'pandas.core.frame.DataFrame'>
```

Step 7 Split dataset

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(feature_list, label,
                                                    test_size=0.20, random_state=42)

print(x_train, x_test, y_train, y_test)
```

```
        drivenKm
    8     404.30
    16    372.40
    3     383.50
    13    372.00
    15    407.00
    17    375.60
    2     396.50
    9     392.20
    18    399.00
    4     321.10
    12    381.00
    7     371.80
    10    386.43
    14    397.00
    6     386.10     drivenKm
    0      390.0
    5      391.3
    11     395.2
    1      403.0     fuelAmount
    8        3809.0
    16       3454.2
    3        3250.5
    13       3622.0
    15       4179.0
    17       3883.8
    2        3471.0
    9        3905.0
    18       4235.9
    4        3263.7
    12       4020.7
    7        3744.5
    10       3874.0
    14       3450.5
    6        3679.0     fuelAmount
    0        3600.0
    5        3445.2
    11       3910.0
    1        3705.0
```

```
print(type(x_train))
```

```
<class 'pandas.core.frame.DataFrame'>
```

PART 1 :LR BASELINE MODEL

STEP 8 BUILD MODEL

```
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()
```

```
lin_reg.fit(x_train,y_train)
```

```
LinearRegression()
```

STEP 9 PREDICT PRICE FOR 800 KM

```
lin_reg.predict([[800]])
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have
  warnings.warn(
array([[6905.64571567]])
```

STEP 10 PREDICT ON ENTIRE DATASET

```
from sklearn.metrics import mean_squared_error
yPred = lin_reg.predict(x_test)
```

```
yPred
```

```
array([[3775.81615646],
       [3785.74000628],
       [3815.51155575],
       [3875.05465468]])
```

STEP 11 MSE

```
mse = mean_squared_error(y_test,yPred)
```

```
mse
```

```
46181.36710639155
```

```
lin_reg.coef_
```

```
array([[7.63373063]])
```

```
lin_reg.intercept_
```

```
array([798.6612099])
```

```
y_pred_data = lin_reg.predict(x_train)
y_pred_data
```

```
array([[3884.9785045 ],
       [3641.46249733],
       [3726.19690735],
       [3638.40900508],
       [3905.58957721],
       [3665.89043536],
       [3825.43540557],
       [3792.61036385],
       [3844.51973215],
       [3249.8521159 ],
       [3707.11258077],
       [3636.88225895],
       [3748.5637381 ],
       [3829.25227089],
       [3746.04460699]])
```

```
lin_reg.score(x_test,y_test)
```

```
-0.6180990161577022
```

PART 2 - LR WITH SCALING USING STANDARD SCALER (STANDARDIZATION)

STEP 12 NORMALIZE USING STANDARD SCALER

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
norm_x_train = scaler.fit_transform(x_train)
norm_y_train = scaler.fit_transform(y_train)
norm_x_test = scaler.transform(x_test)
norm_y_test = scaler.transform(y_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:493: FutureWarning: The feature r
```

```
Feature names unseen at fit time:
- drivenKm
Feature names seen at fit time, yet now missing:
- fuelAmount

  warnings.warn(message, FutureWarning)
```

STEP 13 BUILD LR MODEL

```
norm_lreg = LinearRegression()
```

```
norm_lreg.fit(norm_x_train,norm_y_train)
```

```
LinearRegression()
```

```
norm_yPred = norm_lreg.predict(norm_x_test)
```

STEP 14 MSE

```
norm_mse = mean_squared_error(norm_y_test,norm_yPred)
```

```
norm_mse
```

```
32.25557286448923
```

STEP 15 SCATTER PLOT

```
plt.plot(norm_y_test,norm_yPred,"go")
```

```
[<matplotlib.lines.Line2D at 0x7f3d23e89700>]
```

PART 3 - LR WITH SCALING USING MinMax SCALER (NORMALIZATION)

STEP 16 NORMALIZING USING MINMAX SCALER

```
from sklearn.preprocessing import MinMaxScaler
```

```
minmax = MinMaxScaler()
```

```
mm_norm_x_train = minmax.fit_transform(x_train)
mm_norm_y_train = minmax.fit_transform(y_train)
mm_norm_x_test = minmax.transform(x_test)
mm_norm_y_test = minmax.transform(y_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:493: FutureWarning: The feature r
Feature names unseen at fit time:
- drivenKm
Feature names seen at fit time, yet now missing:
- fuelAmount

  warnings.warn(message, FutureWarning)
```

```
mm_norm_lreg = LinearRegression()
```

```
mm_norm_lreg.fit(mm_norm_x_train,mm_norm_y_train)
```

```
LinearRegression()
```

```
mm_norm_y_pred = mm_norm_lreg.predict(mm_norm_x_test)
```

```
mm_norm_y_pred
```

```
array([[-1.93238929],
       [-1.93151139],
       [-1.92887767],
       [-1.92361023]])
```

```
mm_norm_y_test
```

```
array([[0.3546783 ],
       [0.19758474],
       [0.66927136],
       [0.46123402]])
```

```
minmax_norm_mse = mean_squared_error(mm_norm_y_test,
                                     mm_norm_y_pred)
```

```
minmax_norm_mse
```

```
5.550397233153768
```

prepare the model with input scaling

pipeline = Pipeline(steps=[('normalize', MinMaxScaler()), ('model', LinearRegression())])

fit pipeline

pipeline.fit(train_x, train_y)

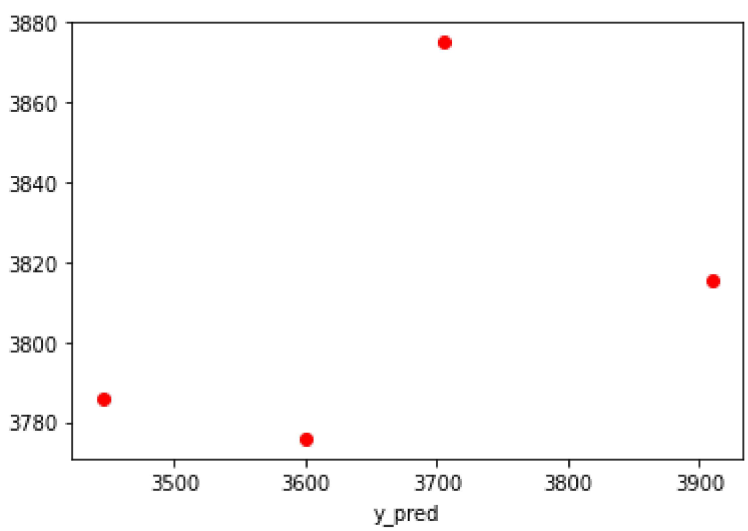
make predictions

yhat = pipeline.predict(test_x)

```
y_test = y_test.to_numpy()
```

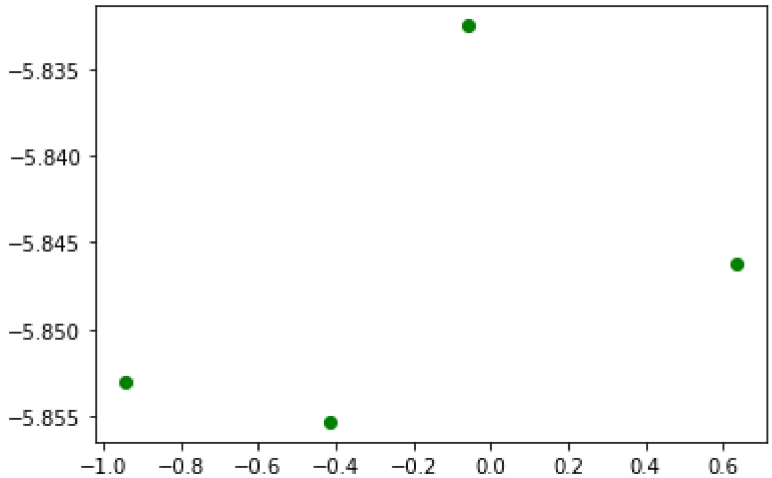
```
plt.xlabel("y_test")
plt.xlabel("y_pred")
plt.plot(y_test,yPred,"ro")
```

```
[<matplotlib.lines.Line2D at 0x7f3d23df3be0>]
```



```
plt.plot(norm_y_test,norm_yPred,"go")
```

```
[<matplotlib.lines.Line2D at 0x7f3d23dcf340>]
```



STEP 17 KNN REGRESSOR

```
from sklearn.neighbors import KNeighborsRegressor
```

```
# creating Instance for the model

knn = KNeighborsRegressor(n_neighbors=5)

# Training / Fitting Data

knn.fit(x_train,y_train)
```

```
KNeighborsRegressor()
```

```
print(knn.predict([[800]]))
```

```
[[3829.08]]
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have
  warnings.warn(
```

```
knn_y_pred = knn.predict(x_test)
```

```
knn_mse = mean_squared_error(knn_y_pred,y_test)
```

```
knn_mse
```

```
21241.836200000045
```

STEP 18 SGD REGRESSOR

```
from sklearn import linear_model
```

```
from sklearn.linear_model import SGDRegressor
```

```
from sklearn.pipeline import make_pipeline
```

```
max_iter = np.ceil(10**6/x_train.shape[0])
```

```
sgd = make_pipeline(StandardScaler(),linear_model.
                    SGDRegressor(max_iter = max_iter,tol=1e-3))
```

```
print(type(x_train))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
x_train = x_train.to_numpy()
y_train = y_train.to_numpy()
```

```
sgd.fit(x_train,y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdregressor', SGDRegressor(max_iter=66667.0))])
```

```
sgd_y_pred = sgd.predict(x_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:443: UserWarning: X has feature r
  warnings.warn(
```

```
sgd_y_pred
```

```
array([3775.49866169, 3785.4202774 , 3815.18512453, 3874.71481879])
```

```
sgd_mse = mean_squared_error(y_test,sgd_y_pred)
```

```
sgd_mse
```

```
46085.64943360797
```

STEP 19 SELECTING THE BEST MODEL

```
from tabulate import tabulate
```

```
data = [["MODELS","MSE VALUE"],
       ["LINEAR REGRESSION",round(mse)],
       ["STANDARD SCALER LR ",round(norm_mse)],
       [" MINMAX  LR",round(minmax_norm_mse) ],
       ["KNN",round(knn_mse)],
       ["SGD",round(sgd_mse)]]
```

```
print(tabulate(data))
```

```
------------------  ---------
MODELS              MSE VALUE
LINEAR REGRESSION   46181
STANDARD SCALER LR  32
MINMAX  LR          6
KNN                 21242
SGD                 46086
------------------  ---------
```

LINEAR REGRESSION MODEL AFTER NORMALIZING USING MINMAX SCALAR WOULD BE THE BEST MODEL AND HAS LOWEST MSE VALUE

✓ 0s completed at 8:43 AM