import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import DecisionTreeRegressor
```

In [2]:
```
# data loading

dataset = pd.read_csv('petrol_consumption.csv')
dataset.head()
```

Out[2]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

In [3]:
```
dataset.shape
```

Out[3]:
```
(48, 5)
```

In [4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 5 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Petrol_tax                      48 non-null     float64
 1   Average_income                  48 non-null     int64  
 2   Paved_Highways                  48 non-null     int64  
 3   Population_Driver_licence(%)    48 non-null     float64
 4   Petrol_Consumption              48 non-null     int64  
dtypes: float64(2), int64(3)
memory usage: 2.0 KB
```

In [5]:
```
dataset.describe()
```

Out[5]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
count	48.000000	48.000000	48.000000	48.000000	48.000000
mean	7.668333	4241.833333	5565.416667	0.570333	576.770833
std	0.950770	573.623768	3491.507166	0.055470	111.885816
min	5.000000	3063.000000	431.000000	0.451000	344.000000
25%	7.000000	3739.000000	3110.250000	0.529750	509.500000
50%	7.500000	4298.000000	4735.500000	0.564500	568.500000
75%	8.125000	4578.750000	7156.000000	0.595250	632.750000
max	10.000000	5342.000000	17782.000000	0.724000	968.000000

In [6]:
```
# seperate out features and target value from dataset

X = dataset.drop(['Petrol_Consumption'],axis = 1).values
y = dataset['Petrol_Consumption'].values
```

In [7]:
```
X.shape
```

Out[7]: (48, 4)

In [8]:
```
y.shape
```

Out[8]: (48,)

In [9]:
```
# split the data in training and testing set

X_train, X_test, y_train,y_test = train_test_split(X,y, test_size = 0.1, random_state = 42)
```

In [10]:
```
# Model

reg = DecisionTreeRegressor()

# fitting
reg.fit(X_train,y_train)
```

Out[10]:
```
DecisionTreeRegressor()
```

In [11]:
```
# predicting
y_pred = reg.predict(X_test)
y_pred
```

Out[11]:
```
array([603., 649., 580., 699., 510.])
```

In [12]:
```
# calculate RMSE

rmse = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
print(rmse)
```

```
61.320469665520335
```

In [13]:
```
y_pred_df = pd.DataFrame(y_pred)
y_pred_df
```

Out[13]:

	0
0	603.0
1	649.0
2	580.0
3	699.0
4	510.0

In [14]:
```
y_pred_df['actual'] = y_test
```

In [15]:
```
y_pred_df
```

Out[15]:

	0	actual
0	603.0	631
1	649.0	587
2	580.0	577
3	699.0	591
4	510.0	460

In [16]:
```
y_pred_df.columns = ['Predicted', 'actual']
```

In [17]:
```
y_pred_df
```

Out[17]:

	Predicted	actual
0	603.0	631
1	649.0	587
2	580.0	577
3	699.0	591
4	510.0	460

In [ ]:
```

```

using hyper parameter tuning

In [34]:
```
parameters = {"splitter" : ['best','random'],
              "max_depth": [1,3,5,7],
              "min_samples_leaf" :[1,2,4,5,6],
              "min_weight_fraction_leaf":[0.2,0.3],
              "max_features":['auto','log2','sqrt',None],
              "max_leaf_nodes":[None,10,20,30]

}
```

In [35]:
```
#parameters =
#{"splitter" : ['best','random'],
#              "max_depth": [1,3,5,7,9,11],
#              "min_sample_leaf" :[1,2,3,4,5,6,7,8,9,10],
#              "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5,0.6,0.8],
#              "max_features":['auto','log2','sqrt',None],
#              "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90]

#}
```

In [36]:
```
# using grid search cv
from sklearn.model_selection import GridSearchCV

```

In [37]:
```
tuning_model = GridSearchCV(reg,param_grid=parameters,
                            scoring = 'neg_mean_squared_error',
                           cv = 3, verbose = 3)
```

In [38]:
```
tuning_model.fit(X,y)
```

```
Fitting 3 folds for each of 1280 candidates, totalling 3840 fits
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best
[CV]  max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best, s
core=-11943.659, total=   0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best
[CV]  max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best, s
core=-8402.213, total=   0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best
[CV]  max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best, s
core=-14008.339, total=   0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random
[CV]  max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random,
score=-12826.466, total=   0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random
[CV]  max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random,
score=-8092.219, total=   0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random
[CV]  max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=random,
score=-21077.797, total=   0.0s
[CV] max_depth=1, max_features=auto, max_leaf_nodes=None, min_samples_leaf=1, min_weight_fraction_leaf=0.2, splitter=best
```

In [28]:
```
reg.get_params().keys()
```

Out[28]:
```
dict_keys(['ccp_alpha', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_sp
lit', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'presort', 'random_state', 'splitter'])
```

In [39]:
```
# best parameters

tuning_model.best_params_
```

Out[39]:
```
{'max_depth': 5,
 'max_features': 'log2',
 'max_leaf_nodes': 30,
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.2,
 'splitter': 'best'}
```

In [40]:
```
# using this hyper parameters to train our model once again

tuned_model = DecisionTreeRegressor(max_depth= 5,max_features= 'log2',
 max_leaf_nodes= 30,min_samples_leaf= 1,min_weight_fraction_leaf= 0.2,
 splitter= 'best')
```

In [41]:
```
# fitting model

tuned_model.fit(X_train,y_train)
```

Out[41]:
```
DecisionTreeRegressor(max_depth=5, max_features='log2', max_leaf_nodes=30,
                      min_weight_fraction_leaf=0.2)
```

In [42]:
```
# prediction

tuned_pred = tuned_model.predict(X_test)
```

In [43]:
```
# calculate RMSE

rmse_tuned = np.sqrt(metrics.mean_squared_error(y_test,tuned_pred))
print(rmse_tuned)
```

```
61.57271376469405
```

In [ ]:
```

```