Random Forest Regression

Data: auto_pricing.csv

```python
# Jesus is my Saviour!
import os
os.chdir('C:\\Users\\Dr Vinod\\Desktop\\tree')
# our exported file will appear here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd

from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from sklearn.metrics import mean_squared_error
from statsmodels.stats.stattools import durbin_watson

df = pd.read_csv("C:/Users/Dr Vinod/Desktop/Auto_Pricing.csv")
df.info()
df.isnull().sum() # no null
sns.heatmap(df.isnull(), cmap='viridis')
list(df)
data = df.loc[:, ['fuel-type', 'length', 'curb-weight', 'horsepower', 'price']]

data.info()
```

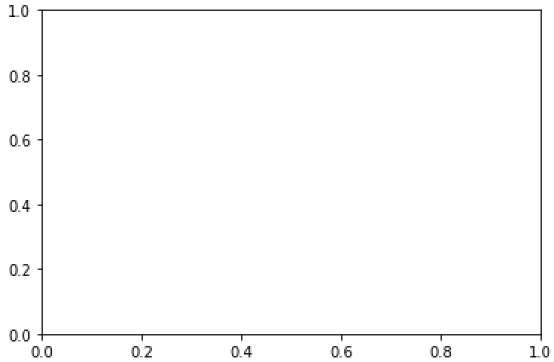
```
In [55]: data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   fuel-type    205 non-null    object
 1   length       205 non-null    float64
 2   curb-weight  205 non-null    int64
 3   horsepower   205 non-null    object
 4   price        205 non-null    object
dtypes: float64(1), int64(1), object(3)
memory usage: 8.1+ KB
```

Libraries and Data

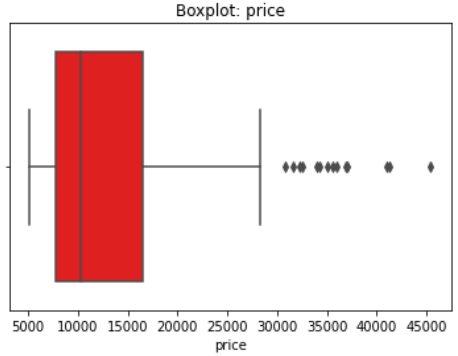
price

```
#______________data cleaning
sns.distplot(data.price)
```

```
ValueError: could not convert string to float: '?'
```

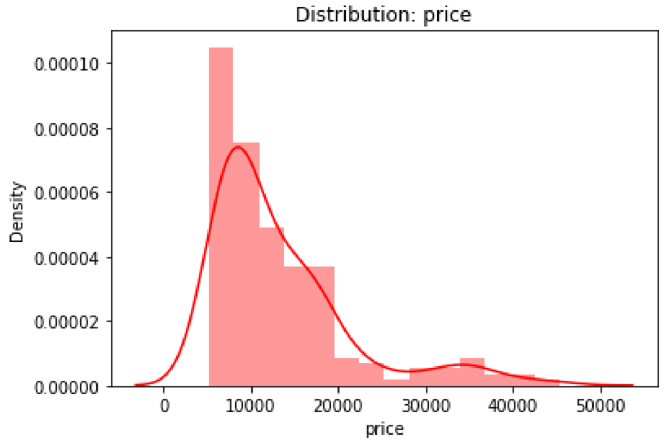



```
In [69]: data['price'] = data['price'].astype('float64')
    ...: sns.boxplot(x=data.price, color = 'red', orient= 'h' )
    ...: plt.title('Boxplot: price')
Out[69]: Text(0.5, 1.0, 'Boxplot: price')
```




```
In [57]: data[data.price == '?']
Out[57]:
     fuel-type  length  curb-weight  horsepower  price
9          gas   178.2         3053         160      ?
44         gas   155.9         1874          70      ?
45         gas   155.9         1909          70      ?
129        gas   175.7         3366         288      ?
```

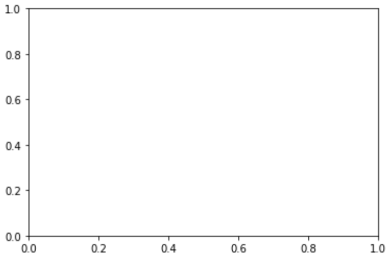
```
data[data.price == '?']
data = data.drop([9,44,45,129]) # now 201 data points
sns.distplot(data.price, color = 'red')
plt.title('Distribution: price')
```



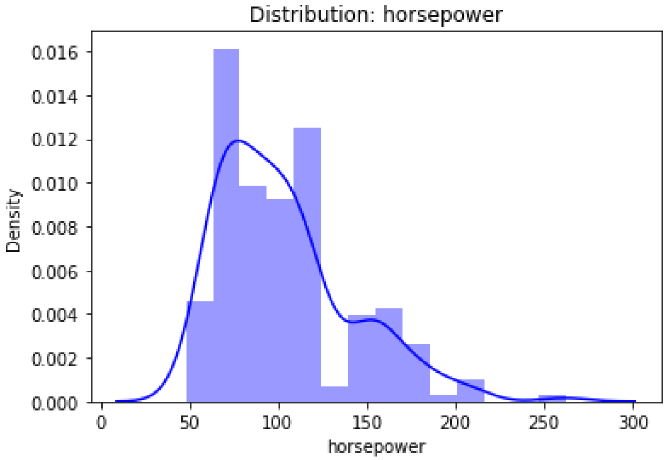
horsepower

```
#_____2  horsepower
sns.distplot(data.horsepower)
```

```
ValueError: could not convert string to float: '?'
```



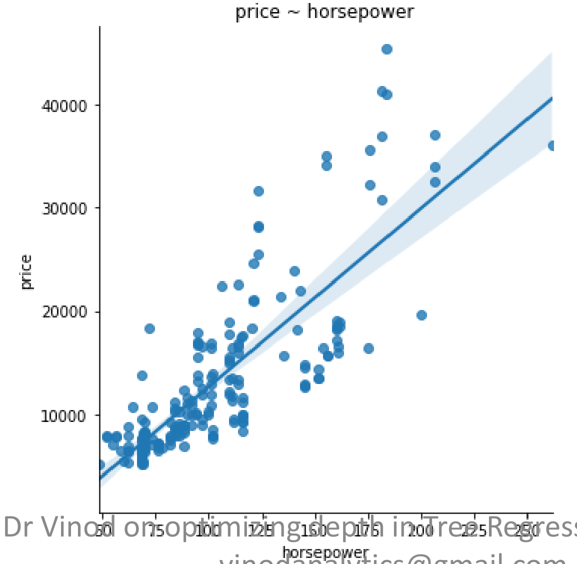
```
data[data.horsepower == '?'] # 130,131
data = data.drop([130,131]) # now 199 data points
sns.distplot(data.horsepower, color = 'blue')
plt.title('Distribution: horsepower')
```

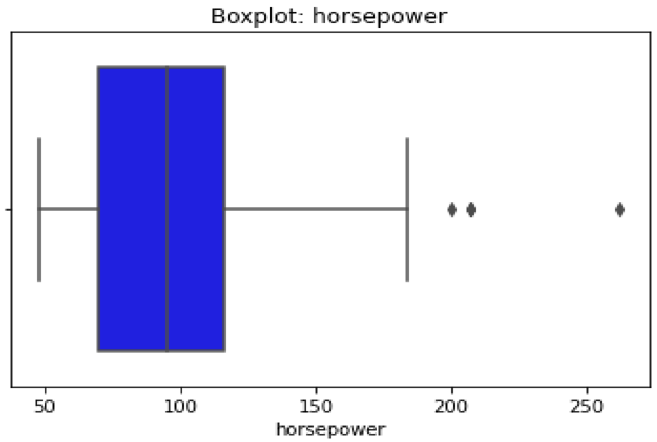



```
sns.lmplot(x='horsepower', y='price', data = data, fit_reg= True)
plt.title('price ~ horsepower')
```

```
data['horsepower'] = data['horsepower'].astype('float64')
sns.boxplot(x=data.horsepower, color = 'blue', orient= 'h' )
plt.title('Boxplot: horsepower')
```

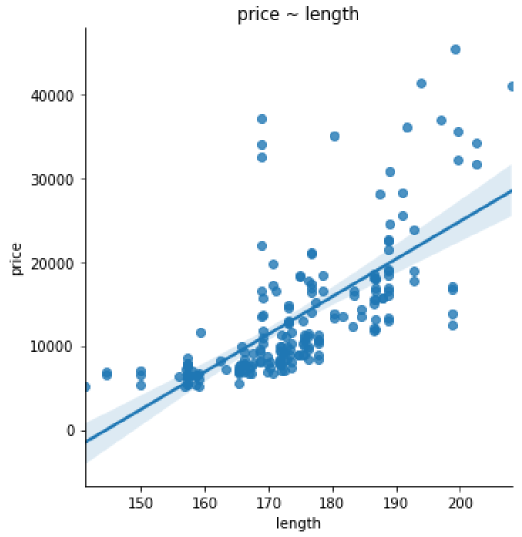
```
In [85]: np.corrcoef(data.price, data.horsepower)
Out[85]:
array([[1.        , 0.81053308],
       [0.81053308, 1.        ]])
```





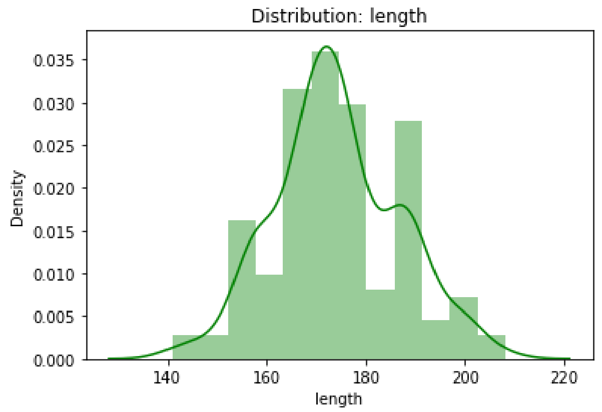
length

```
sns.lmplot(x='length', y='price', data = data, fit_reg= True)
plt.title('price ~ length')
```

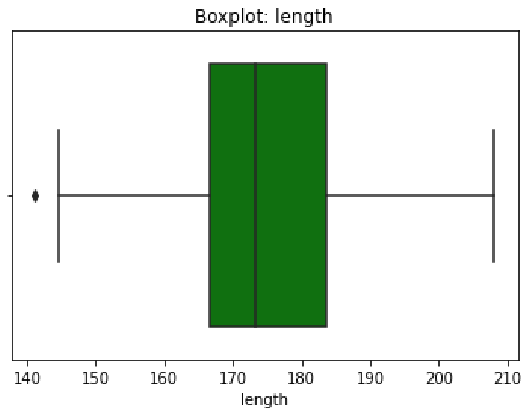



```
In [90]: np.corrcoef(data.price, data.length)
Out[90]:
array([[1.        , 0.69396477],
       [0.69396477, 1.        ]])
```

```
sns.distplot(data.length, color = 'green')
plt.title('Distribution: length')
```



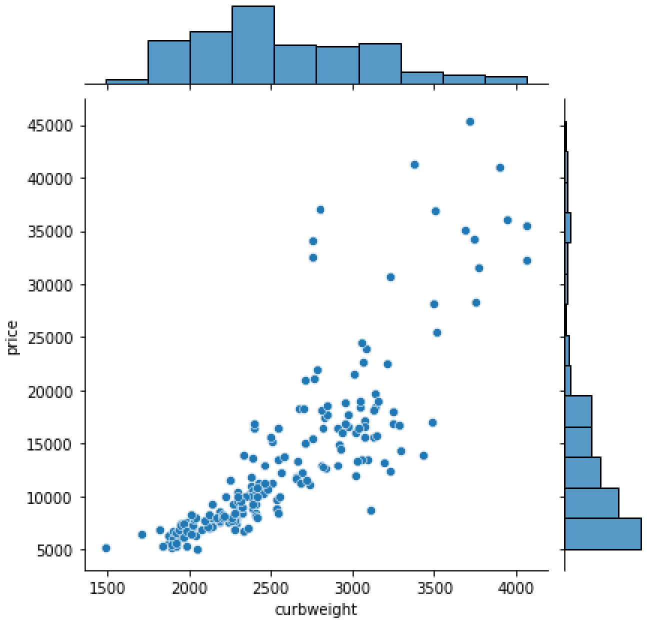

```
sns.boxplot(x=data.length, color = 'green', orient= 'h' )
plt.title('Boxplot: length')
```

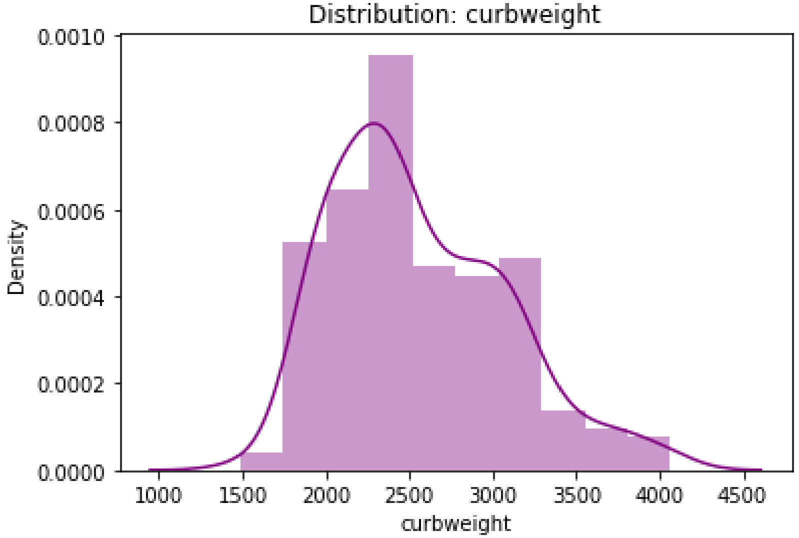


curbweight

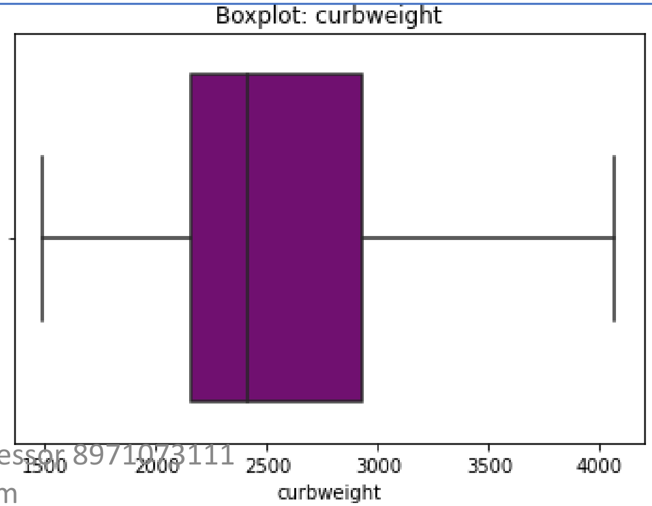
```
# need to change names of curb-weight and fuel-type

data = data.rename(columns = {'curb-weight': 'curbweight', 'fuel-type':'fueltype'})

sns.distplot(data.curbweight, color = 'purple')
plt.title('Distribution: curbweight')
```

```
sns.jointplot(x='curbweight', y='price', data = data)
```






```
sns.boxplot(x=data.curbweight, color = 'purple', orient= 'h' )
plt.title('Boxplot: curbweight')
```




```
In [103]: np.corrcoef(data.price, data.curbweight)
Out[103]:
array([[1.        , 0.83509045],
       [0.83509045, 1.        ]])
```

correlations

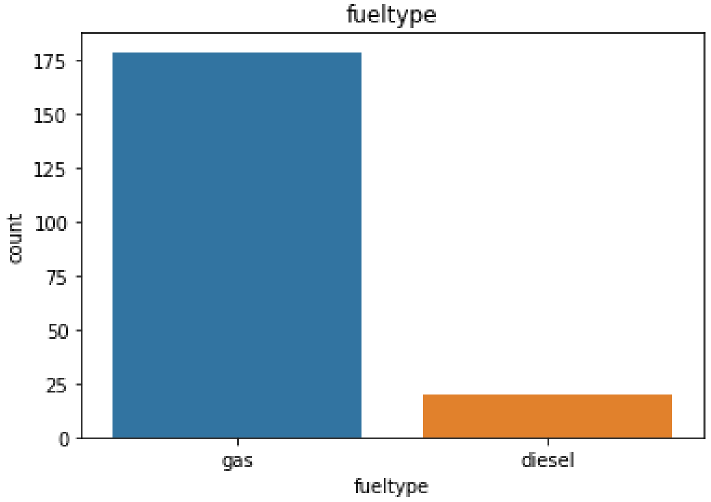
```
data.corr() # object ignored automatically!
sns.heatmap(data.corr(), annot = True, cmap = 'viridis')
```

```
In [110]: data.corr()
Out[110]:
              length  curbweight  horsepower     price
length      1.000000    0.881688    0.580309  0.693965
curbweight  0.881688    1.000000    0.758063  0.835090
horsepower  0.580309    0.758063    1.000000  0.810533
price       0.693965    0.835090    0.810533  1.000000

In [111]: sns.heatmap(data.corr(), annot = True, cmap = 'viridis')
Out[111]: <AxesSubplot:>
```

fueltype

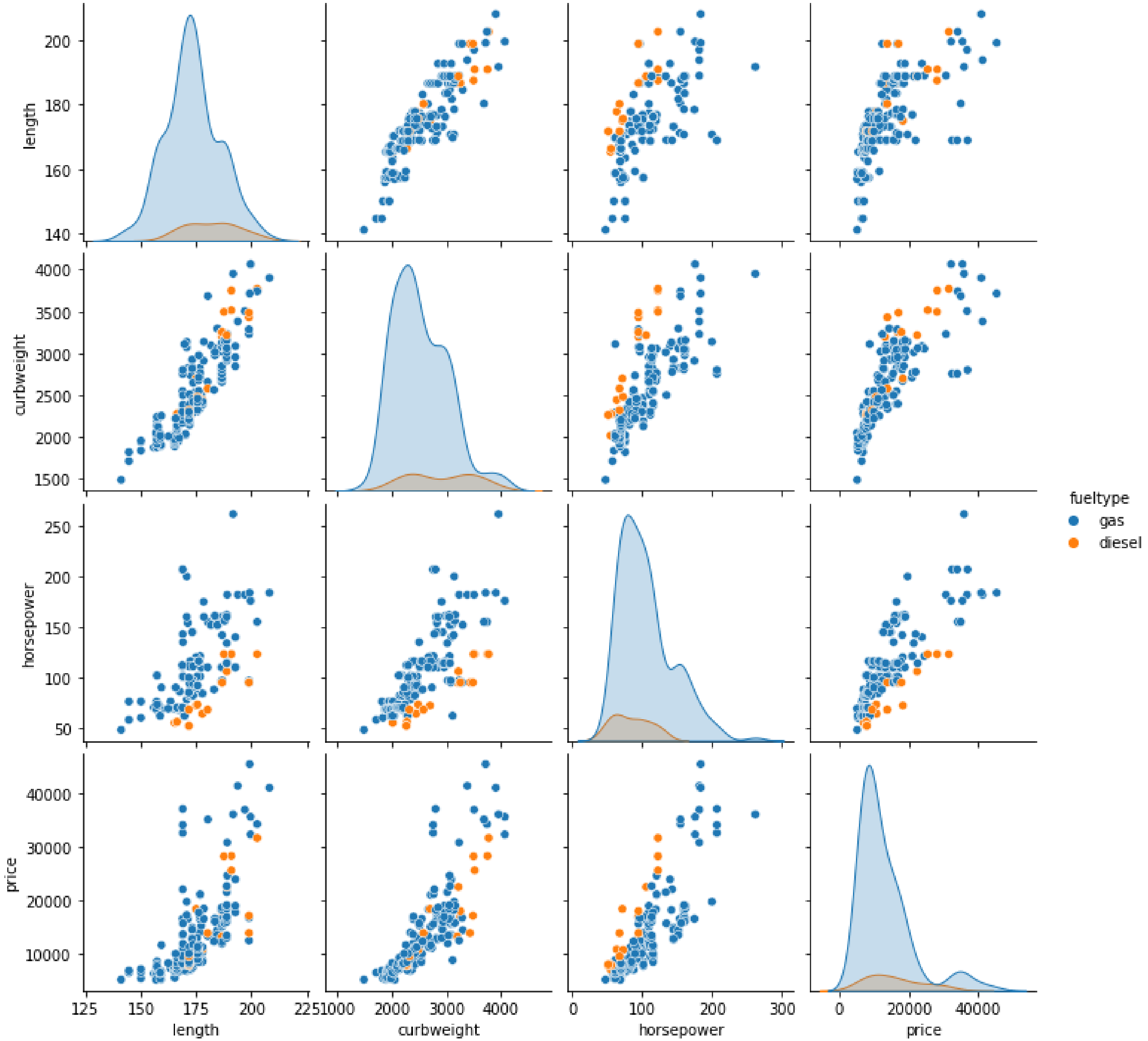
```
In [106]: data.fueltype.value_counts()
Out[106]:
gas       179
diesel     20
Name: fueltype, dtype: int64
```

```
sns.countplot(x = data.fueltype, data = data)
plt.title('fueltype')
```



pairplot

```
sns.pairplot(data, hue = 'fueltype')
```



We have done a good amount of work. For avoiding repetition of the same, let's save at destop/somewhere else.

Further work we will do with this pre-processed data!

```
# _______better write this data
data.to_csv("C:/Users/Dr Vinod/Desktop/autopriceTree.csv")
'''
post exporting the data at desktop
open the file and delete 1st un-necessary
column ....unmamed, and save the file
'''
```

Data

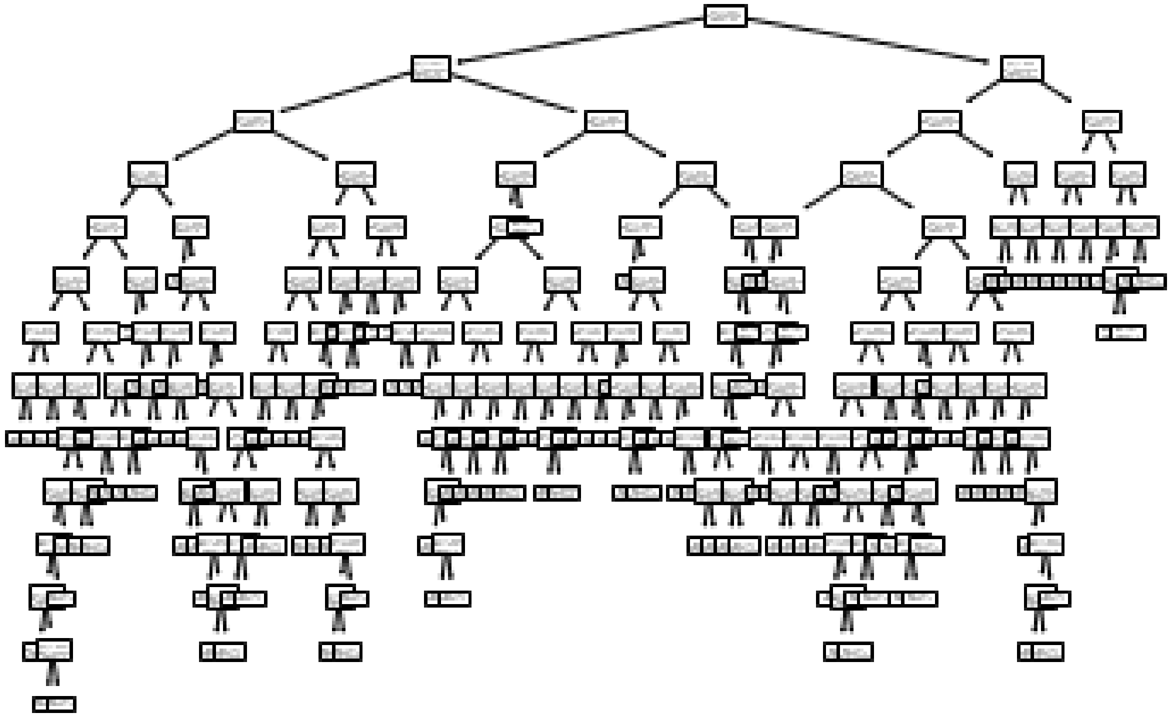
```
In [2]: df = pd.read_csv("C:/Users/Dr Vinod/Desktop/autopriceTree.csv")

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199 entries, 0 to 198
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   fueltype    199 non-null    object
 1   length      199 non-null    float64
 2   curbweight  199 non-null    int64
 3   horsepower  199 non-null    int64
 4   price       199 non-null    int64
dtypes: float64(1), int64(3), object(1)
memory usage: 7.9+ KB
```

```
#____________lets start a fresh
df = pd.read_csv("C:/Users/Dr Vinod/Desktop/autopriceTree.csv")
df.info()
#_________Before proceeding furthet, label fueltype
# Need to label fueltype as 0 and 1
df['fueltype'] = df.get('fueltype').replace('diesel', 0)
df['fueltype'] = df.get('fueltype').replace('gas', 1)

#_____X and y
df.info()
X = df.drop('price', axis=1) # Easy
y = df['price']
```

Fit Tree

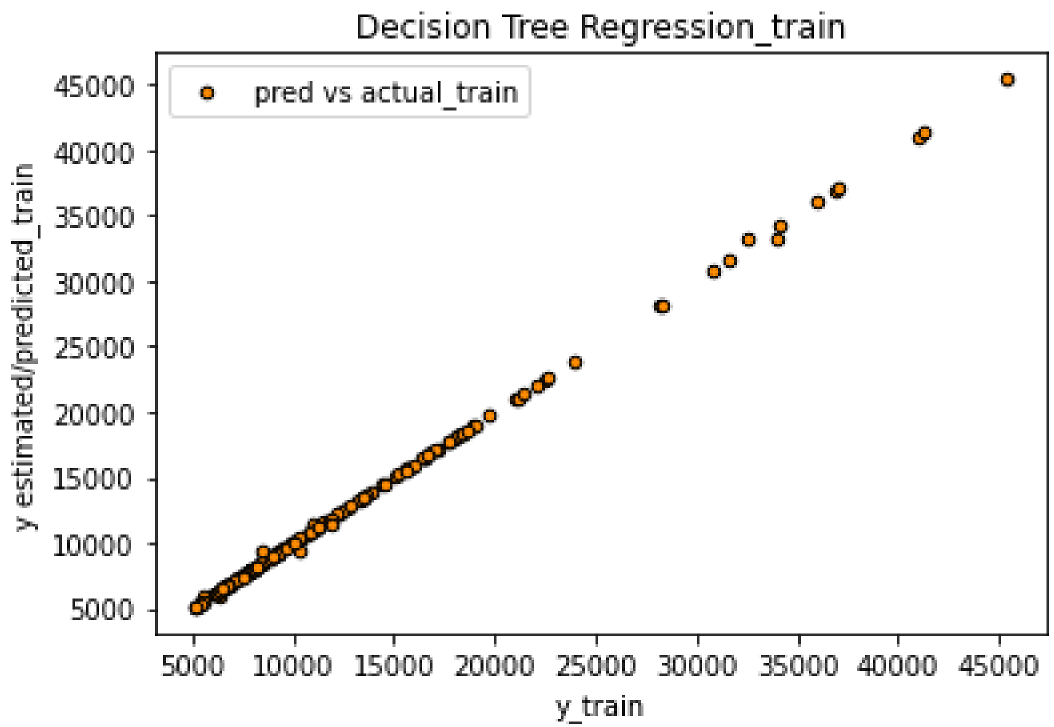
```
# train test

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20, random_state = 45)

# fit tree on train data
# model
regr = DecisionTreeRegressor()
# Fit regression model on train set
regr.fit(X_train, y_train)
# predict/estimate_train X_train
yest_train = regr.predict(X_train)

# see tree_train
from sklearn import tree
tree.plot_tree(regr.fit(X_train, y_train)) # 13 deep
```

Large tree , 13 deep



Train data, 13 deep

```
# Plot the results; looks tree has learnt everything!
plt.figure()
plt.scatter(y_train, yest_train, s=20, edgecolor="black",
            c="darkorange", label="pred vs actual_train")
plt.xlabel("y_train")
plt.ylabel("y estimated/predicted_train")
plt.title("Decision Tree Regression_train")
plt.legend()
plt.show()
```

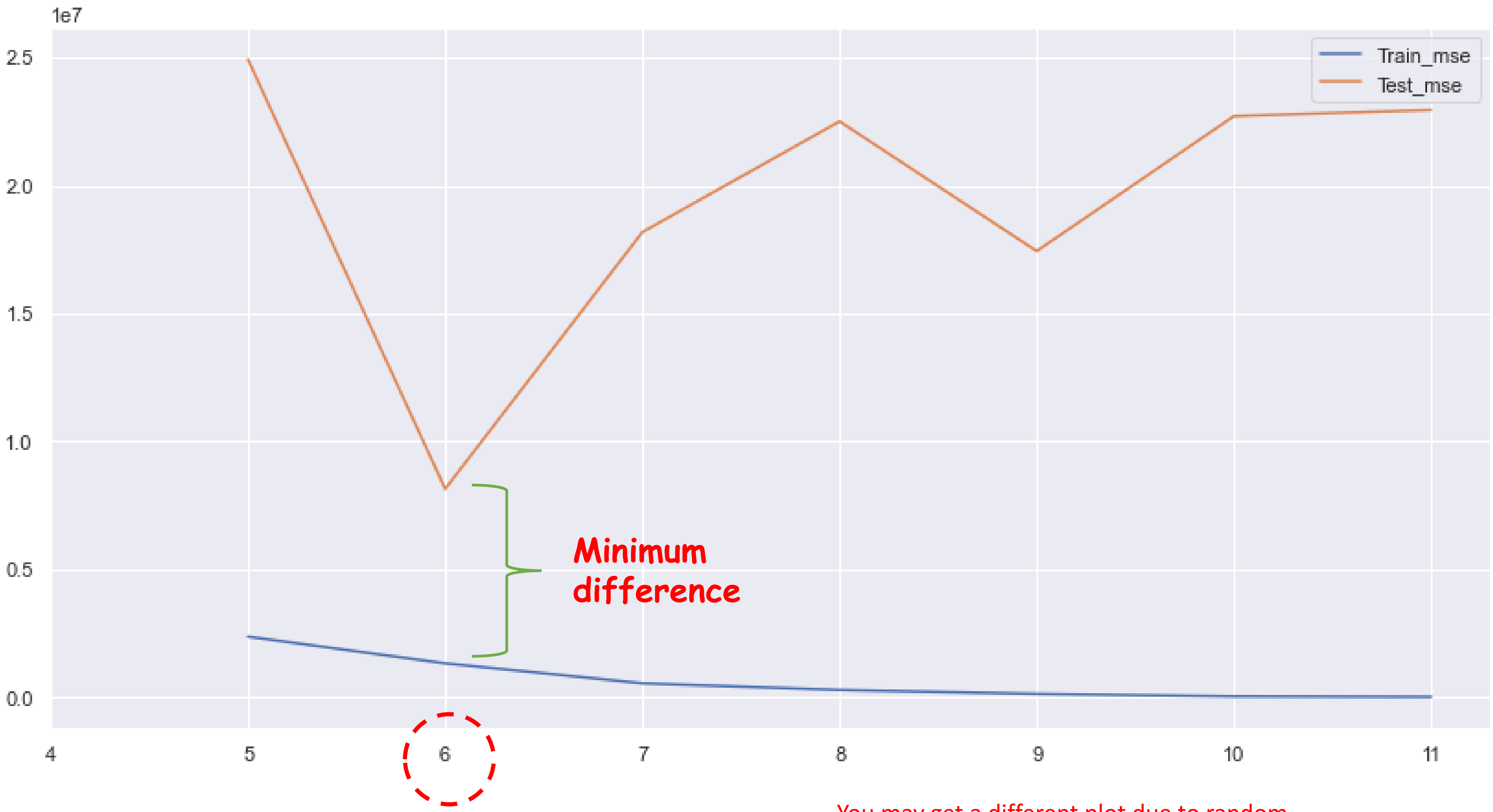


Look at the mse values of train and test, HUGE!

```python
# mse/rmse_train
from sklearn.metrics import mean_squared_error
mse_train = mean_squared_error(y_train, yest_train)
print(mse_train) #21,396.27

#______________mse @ test
# predict/estimate_test X_est
yest_test = regr.predict(X_test)
# residual test
test_residual = y_test - yest_test
# mse/rmse_test
from sklearn.metrics import mean_squared_error
mse_test = mean_squared_error(y_test, yest_test)
print(mse_test) #18821007.65
```

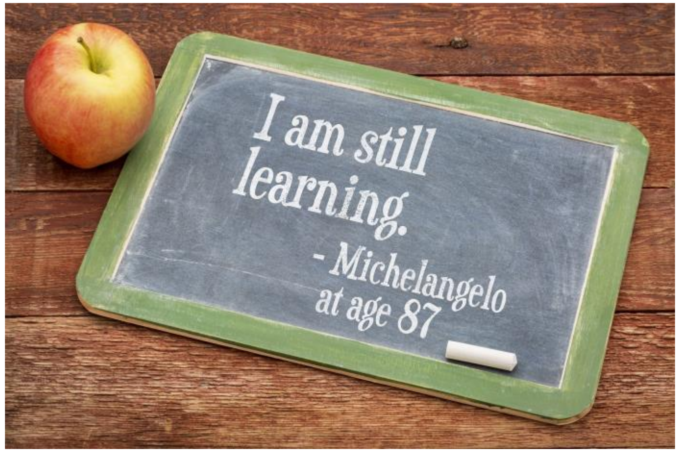
Your values may differe due to random sampling!

```python
#___________________________find optimum no of depth
depth = [5,6,7,8,9,10,11]

mse_train, mse_test = [],[]

for i in depth:
    prdctr = DecisionTreeRegressor(max_depth= i)
    prdctr.fit(X_train, y_train)
    y_train_pred=prdctr.predict(X_train)
    y_test_pred=prdctr.predict(X_test)

    mse_train.append(mean_squared_error(y_train, y_train_pred))
    mse_test.append(mean_squared_error(y_test, y_test_pred))

# now we have mses
# lets, plot

sns.set()
plt.figure(figsize = (14,7))
sns.lineplot(y=mse_train, x = depth, label = 'Train_mse')
sns.lineplot(y=mse_test, x = depth, label = 'Test_mse')
plt.xticks(ticks=np.arange(4,12,1))
plt.show()
```



You may get a different plot due to random sampling

And, now you can go ahead and finalize your tree with 6 deep!



Random Forest Regression

```
# Jesus is my Saviour!

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.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

d = pd.read_csv("C:/Users/Dr Vinod/Desktop/DataSets1/autopriceTree.csv")
d.info()

# need to convert 'fueltype, diesel and gas' into numbers
# Need to label fueltype as 0 and 1
d['fueltype'] = d.get('fueltype').replace('diesel', 0)
d['fueltype'] = d.get('fueltype').replace('gas', 1)
x = d.iloc[:,0:4] #4 Predictors;4th will not be picked!
x.info()
y = d.iloc[:,4]
```

```
# _______better write this data
data.to_csv("C:/Users/Dr Vinod/Desktop/autopriceTree.csv")
'''
post exporting the data at desktop
open the file and delete 1st un-necessary
column ....unmamed, and save the file
'''
```

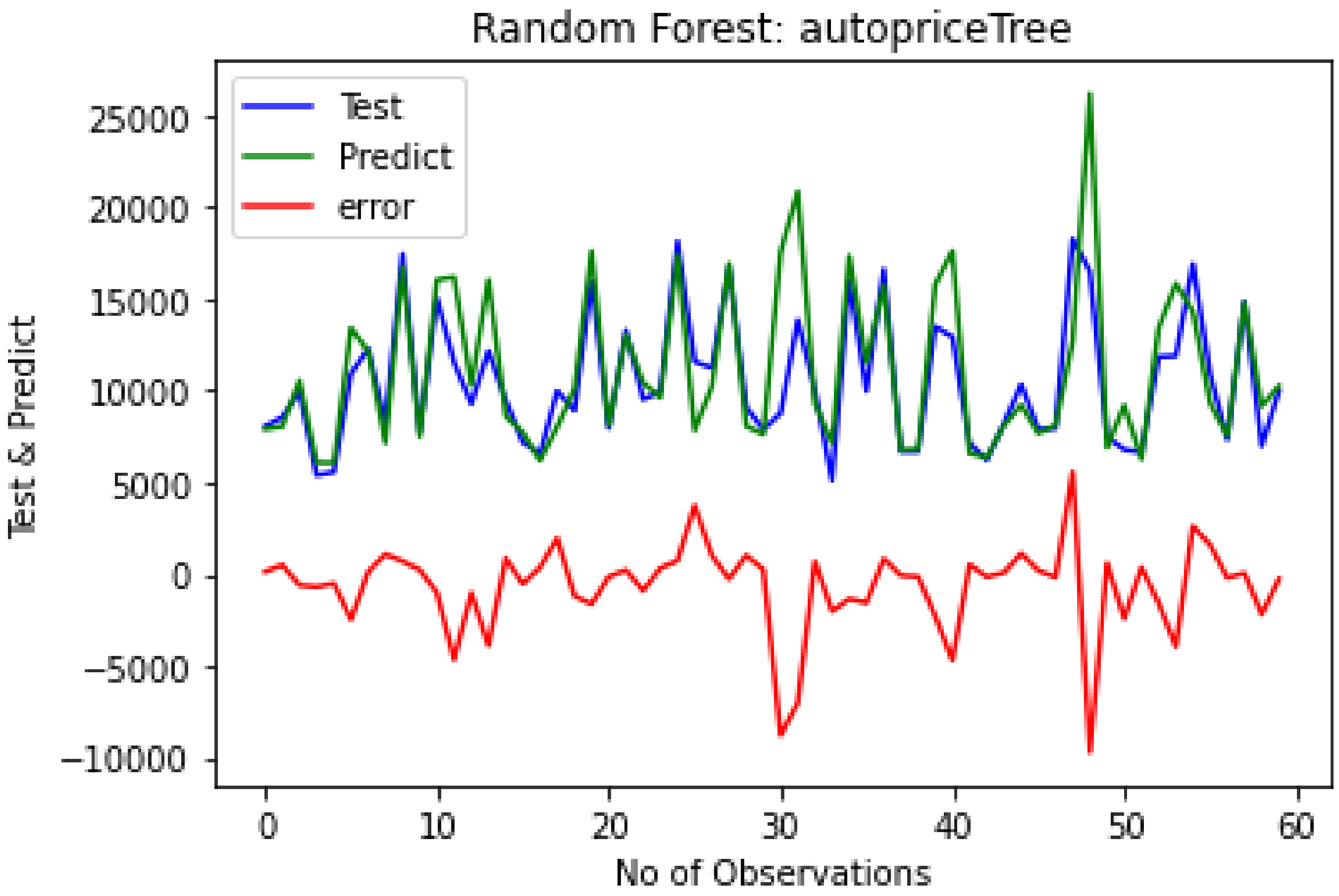
Random Forest Regression

```
#Splitting the Dataset
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_state=123)

# import the regressor
from sklearn.ensemble import RandomForestRegressor

# create regressor object
# in our previous experiment, we found 6 is the best depth
regressor = RandomForestRegressor(n_estimators = 100, max_depth=6, random_state = 0)

# fit the regressor with x and y data=TRAIN
mod = regressor.fit(x_train, y_train)

#Prediction
y_pred = mod.predict(x_test) # array
```

Random Forest Regression

```
#RMSE
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
mse
RMSE = np.sqrt(mse)
print(RMSE) # 2593.5

obsno = np.arange(60)
resid = y_test - y_pred

#Plot
plt.plot(obsno, y_test, 'b')
plt.plot(obsno, y_pred, 'g')
plt.plot(obsno, resid, 'r')
plt.xlabel('No of Observations')
plt.ylabel('Test & Predict')
plt.title('Random Forest: autopriceTree')
plt.legend(labels= ('Test', 'Predict', 'error'), loc= 'upper left')
plt.show()
```

Random Forest Regression



Random Forest Regression

```
#_____________________________________Extract Feature importance
fi = pd.DataFrame({'feature': list(x_train.columns),
                   'importance': mod.feature_importances_}).\
                    sort_values('importance', ascending=False)

fi.head()
'''
feature  importance
2  curbweight    0.648642
3  horsepower    0.308764
1      length    0.038601
0    fueltype    0.003993
'''
```

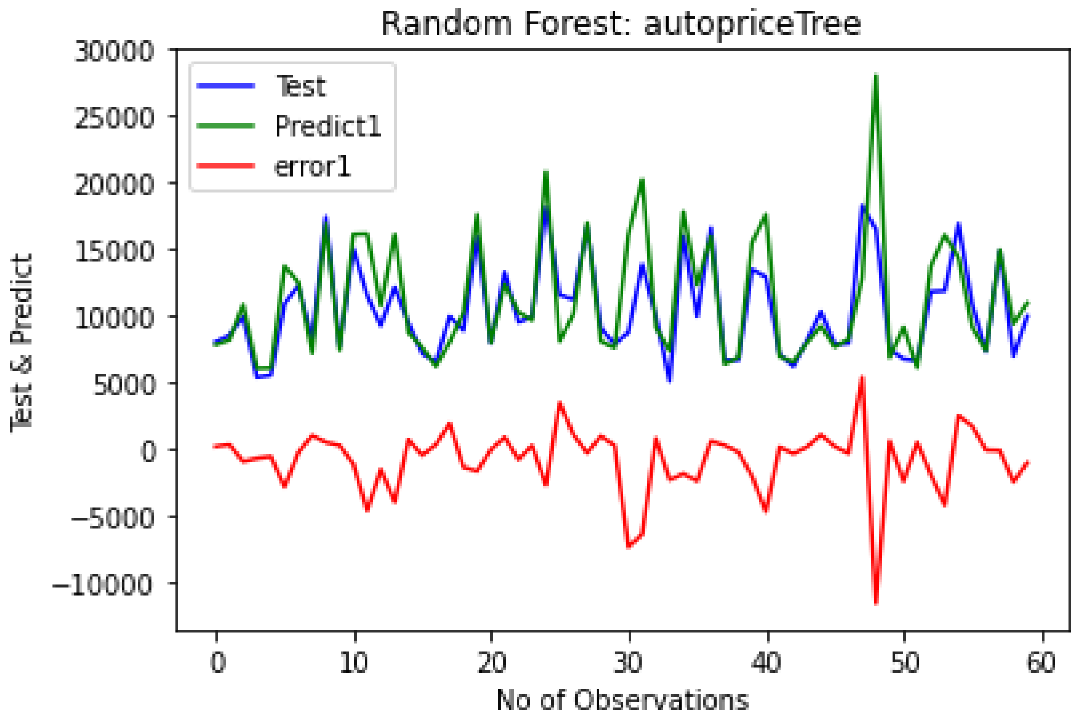
Random Forest Regression: 2nd Model

```
#________why not we should build a model with curbweight and horsepower?
# import the regressor
from sklearn.ensemble import RandomForestRegressor
# create regressor object
# in our previous experiment, we found 6 is the best depth
regressor = RandomForestRegressor(n_estimators = 100, max_depth=6, random_state = 0)
# fit the regressor with x and y data=TRAIN, this time with curbweight and horsepower only
x_train.info()
x_train1 = x_train.iloc[:,[2,3]]
mod1 = regressor.fit(x_train1, y_train)
#Prediction
x_test.info()
x_test1 = x_test.iloc[:,[2,3]]
y_pred1 = mod1.predict(x_test1) # array
#RMSE
from sklearn.metrics import mean_squared_error
mse1 = mean_squared_error(y_test, y_pred1)
mse1
RMSE1 = np.sqrt(mse1)
print(RMSE1) # 2593.5(all 4); 2668.5(only2)

obsno = np.arange(60)
resid1 = y_test - y_pred1
```

Random Forest Regression: 2nd model

```
print(RMSE1) # 2593.5(all 4); 2668.5(only2)
```

```
obsno = np.arange(60)
resid1 = y_test - y_pred1

#Plot
plt.plot(obsno, y_test, 'b')
plt.plot(obsno, y_pred1, 'g')
plt.plot(obsno, resid1, 'r')
plt.xlabel('No of Observations')
plt.ylabel('Test & Predict')
plt.title('Random Forest: autopriceTree')
plt.legend(labels= ('Test', 'Predict1', 'error1'), loc= 'upper left')
plt.show()
```



3rd Model through Grid Search

```
#___________________________Grid SEARCH and RANDOM SEARCH

from sklearn.model_selection import GridSearchCV

param_grid = {'bootstrap': [True],'max_depth': [5,6,7,8,9,10,11],
              'max_features': ['auto', 'log2'],
              'n_estimators': [25, 50, 100, 150, 200]}

rfr = RandomForestRegressor(random_state = 1)

g_search = GridSearchCV(estimator = rfr, param_grid = param_grid, cv = 3)

gmod = g_search.fit(x_train, y_train)

# best parameters
print(gmod.best_params_)

gprd = gmod.predict(x_test)

#RMSE
from sklearn.metrics import mean_squared_error
mse_g = mean_squared_error(y_test, gprd)
mse_g
RMSE_g = np.sqrt(mse_g)
print(RMSE_g) # 2593.5(all 4); 2668.5(only2); 2413.2 (gridsearch)
```

```
In [22]: print(gmod.best_params_)
{'bootstrap': True, 'max_depth': 10, 'max_features': 'auto',
'n_estimators': 50}
```

Random Forest Regression: 4^{th} Model

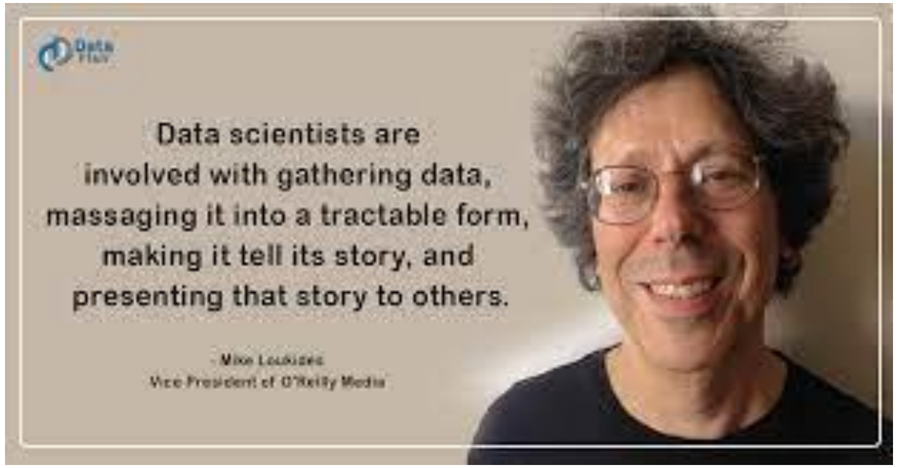
```
#_____________RANDOM SEARCH
from sklearn.model_selection import RandomizedSearchCV
rfr_random = RandomizedSearchCV(estimator=rfr,
                                param_distributions=param_grid,
                                n_iter = 20, cv = 3, random_state=421)

rmod = rfr_random.fit(x_train, y_train)

# best parameter, hyper tuned
print(rmod.best_params_)

rprd = rmod.predict(x_test)

#RMSE
from sklearn.metrics import mean_squared_error
mse_r = mean_squared_error(y_test, rprd)
mse_r
RMSE_r = np.sqrt(mse_r)
print(RMSE_r) # 2593.5(all 4); 2668.5(only2); 2413.2 (gridsearch); 2397.6 (rsearch)
```

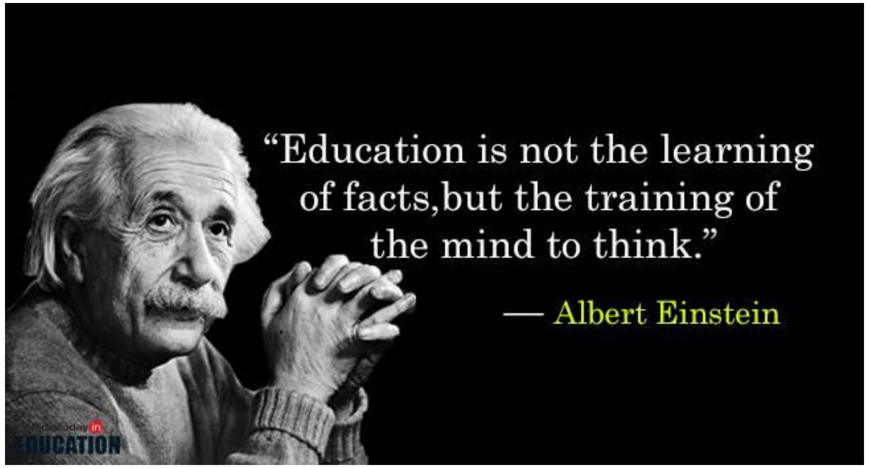
```
In [38]: print(rmod.best_params_)
{'n_estimators': 50, 'max_features': 'auto', 'max_depth': 8,
'bootstrap': True}
```

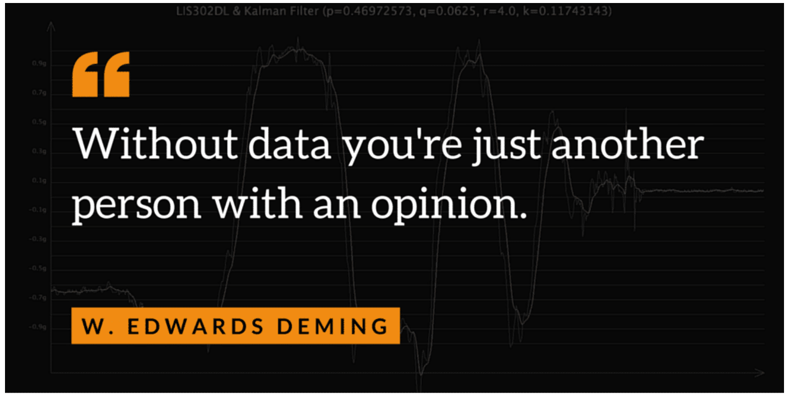
Time!

```
In [39]: from timeit import default_timer

In [40]: begining = default_timer()
    ...: gmod = g_search.fit(x_train, y_train)
    ...: ending = default_timer()
    ...: print((ending-begining)*1000) # 42055.1
20818.372499999896

In [41]: from timeit import default_timer
    ...: begining = default_timer()
    ...: rmod = rfr_random.fit(x_train, y_train)
    ...: ending = default_timer()
    ...: print((ending-begining)*1000) # 42055.1 (grid); 11788.63
6630.828400000155
```

```
11788/42055 # only 28% of time !
6630.83/20818.37 # only 32% of time!
```


Data scientists are involved with gathering data, massaging it into a tractable form, making it tell its story, and presenting that story to others.
- Mike Loukides
Vice President of O'Reilly Media


"Education is not the learning of facts,but the training of the mind to think."
— Albert Einstein


Without data you're just another person with an opinion.
W. EDWARDS DEMING


We should teach the students, as well as executives, how to conduct experiments, how to examine data, and how to use these tools to make better decisions.
- Dan Ariely
Researcher