# mtcar

February 14, 2023

### 0.0.1 Importing required libraries to regression analysis using mtcar dataset

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import math
```

[3]:
```
# to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

### 0.0.2 With the help of Pandas, read the ‘‘.csv” file and performing some task

[4]:
```
data1 = pd.read_csv("auto.csv")
```

[5]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   mpg:           398 non-null    float64
 1   cylinders:     398 non-null    int64
 2   displacement:  398 non-null    float64
 3   horsepower:    398 non-null    object
 4   weight:        398 non-null    int64
 5   acceleration:  398 non-null    float64
 6   model          398 non-null    int64
 7   origin:        398 non-null    int64
 8   car            398 non-null    object
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

### 0.0.3 Renaming the columns for better understanding

[6]:
```
data1.rename(columns = {'mpg:':'mpg','cylinders:':'cyl','displacement:':
 ↪'disp','horsepower:':'hp','weight:':'wt','acceleration:':'acc','origin:':
 ↪'origin'}, inplace = True)
```

[7]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   mpg     398 non-null    float64
 1   cyl     398 non-null    int64  
 2   disp    398 non-null    float64
 3   hp      398 non-null    object 
 4   wt      398 non-null    int64  
 5   acc     398 non-null    float64
 6   model   398 non-null    int64  
 7   origin  398 non-null    int64  
 8   car     398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

[8]:
```
# horsepower value will throw an error, because some of values are in object␣
 ↪form
# Obeject must be neglected or converted into suitable form
data1 = data1[data1['hp'] != '?']
```

[9]:
```
data1.head(4)
```

[9]:

| | mpg | cyl | disp | hp | wt | acc | model | origin | car |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130 | 3504 | 12.0 | 70 | 1 | chevrolet chevelle malibu |
| 1 | 15.0 | 8 | 350.0 | 165 | 3693 | 11.5 | 70 | 1 | buick skylark 320 |
| 2 | 18.0 | 8 | 318.0 | 150 | 3436 | 11.0 | 70 | 1 | plymouth satellite |
| 3 | 16.0 | 8 | 304.0 | 150 | 3433 | 12.0 | 70 | 1 | amc rebel sst |

[10]:
```
data1.tail(4)
```

[10]:

| | mpg | cyl | disp | hp | wt | acc | model | origin | car |
|---|---|---|---|---|---|---|---|---|---|
| 394 | 44.0 | 4 | 97.0 | 52 | 2130 | 24.6 | 82 | 2 | vw pickup |
| 395 | 32.0 | 4 | 135.0 | 84 | 2295 | 11.6 | 82 | 1 | dodge rampage |
| 396 | 28.0 | 4 | 120.0 | 79 | 2625 | 18.6 | 82 | 1 | ford ranger |
| 397 | 31.0 | 4 | 119.0 | 82 | 2720 | 19.4 | 82 | 1 | chevy s-10 |

[11]:
```
data1.isnull().any()
```

[11]:
```
mpg       False
cyl       False
disp      False
hp        False
wt        False
acc       False
model     False
origin    False
car       False
dtype: bool
```

[12]:
```
data1.isnull().sum()
```

[12]:
```
mpg       0
cyl       0
disp      0
hp        0
wt        0
acc       0
model     0
origin    0
car       0
dtype: int64
```

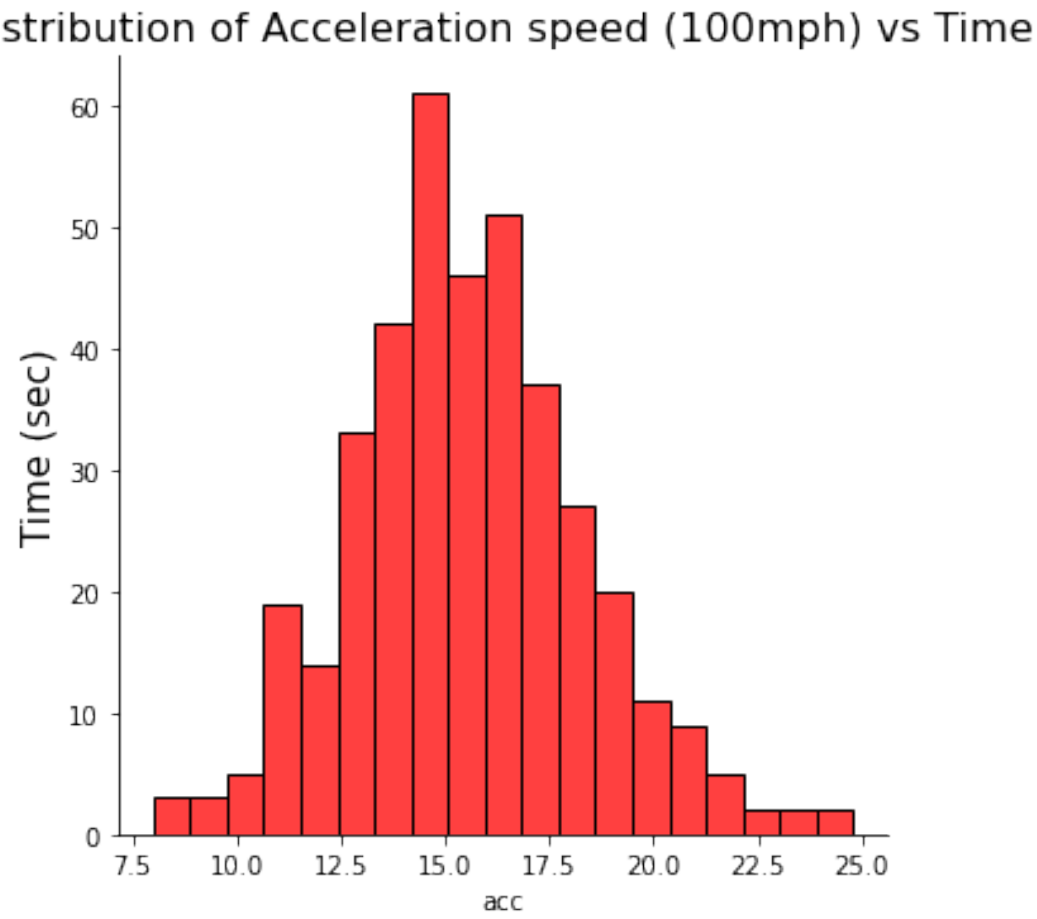
[13]:
```
# Dropping the column which is not required for further analysis
data1 = data1.drop(['car'], axis=1)
```

### 0.0.4 Using libraries seaborn and matplotlib to display the total time require for a car to reach 100miles speed

[14]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[15]:
```
plt.figure(figsize=(7,5))
sns.displot(data1['acc'],color='red')
plt.title('Distribution of Acceleration speed (100mph) vs Time ', fontsize=16)
plt.ylabel("Time (sec)", fontsize=15)
plt.show()
```

```
<Figure size 504x360 with 0 Axes>
```




### 0.0.5 Loading the data to X and Y for Testing and Training

[16]:
```
X = data1.drop("mpg", axis=1)
y = data1["mpg"]
```

[17]:
```
#in order to make linear regression model we have to seperate target variable
 ↪and predictor variable
print(X)
```

```
     cyl   disp   hp    wt   acc  model  origin
0      8  307.0  130  3504  12.0     70       1
1      8  350.0  165  3693  11.5     70       1
2      8  318.0  150  3436  11.0     70       1
3      8  304.0  150  3433  12.0     70       1
4      8  302.0  140  3449  10.5     70       1
..   …    …  …   …    …     …      …
393    4  140.0   86  2790  15.6     82       1
394    4   97.0   52  2130  24.6     82       2
```

```
395    4  135.0   84  2295  11.6     82       1
396    4  120.0   79  2625  18.6     82       1
397    4  119.0   82  2720  19.4     82       1

[392 rows x 7 columns]
```

[18]:
```
#displayes the mpg values
print(y)
```

```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
       …
393    27.0
394    44.0
395    32.0
396    28.0
397    31.0
Name: mpg, Length: 392, dtype: float64
```

### 0.0.6 Define and perform K-fold cross validation

[19]:
```
kf = KFold(n_splits=5)
```

[20]:
```
# Perform k-fold cross validation
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

### 0.0.7 Intialize the linear regression model and fit the model

[21]:
```
model = LinearRegression()
```

[22]:
```
model.fit(X_train, y_train)
```

[22]:
```
LinearRegression()
```

[23]:
```
# Make predictions on the test data
y_pred = model.predict(X_test)
```

[24]:
```
# Initialize a list to store the mean squared errors
mse_scores = []
```

[25]:
```
# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
mse_scores.append(mse)
```

[32]:
```
# Calculating the Mean Square Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE): ", mse)
```

```
Mean Squared Error (MSE):  27.844743081984227
```

[33]:
```
# Calculating the R2 Score
r2 = r2_score(y_test, y_pred)
print("Coefficient of determination R-squared (R2): ", r2)
```

```
Coefficient of determination R-squared (R2):  0.22505939810248488
```

[29]:
```
# Calculating the Root Mean Squared error
rmse = math.sqrt(mse)
print("Root Mean Squared Error (RMSE): ", rmse)
```

```
Root Mean Squared Error (RMSE):  5.276811829313627
```

[30]:
```
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE): ", mae)
```

```
Mean Absolute Error (MAE):  3.9728146766120083
```

### 0.0.8 Plotting regression graph

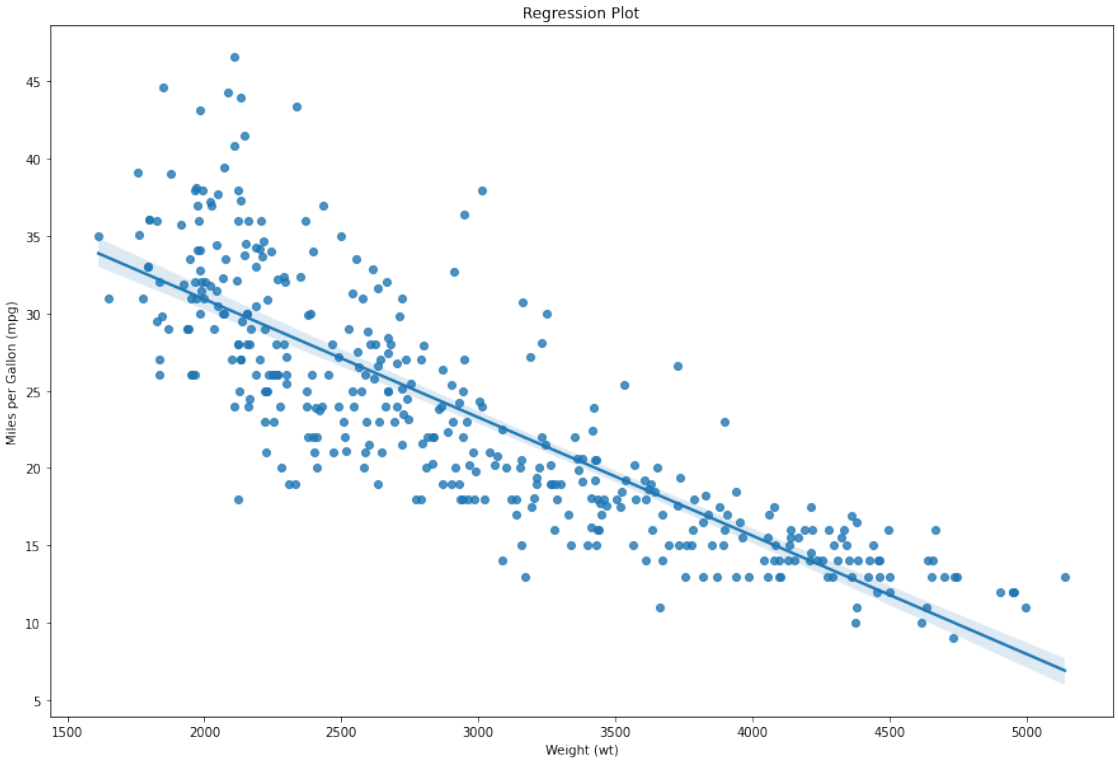
[31]:
```
# Selecting the relevant columns for the regression
x = data1['wt']
y = data1['mpg']

#enlarging the figure for better visualization
fig, ax = plt.subplots(figsize=(15, 10))

# Fit a regression model
sns.regplot(x, y, ax=ax)

# Add a title and labels to the plot
plt.title("Regression Plot")
plt.xlabel("Weight (wt)")
plt.ylabel("Miles per Gallon (mpg)")

# Show the plot
plt.show()
```


Regression Plot
Miles per Gallon (mpg)
Weight (wt)
45
40
35
30
25
20
15
10
5
1500
2000
2500
3000
3500
4000
4500
5000

[ ]: