CASE STUDY ON REGRESSION

BY:-

SHRADDHA JAIN 20-PBD-002

RIDDHI SHAH 20-PBD-032

Data Set Information:

- This dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1993) in predicting the attribute "mpg“.

- "The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes." (Quinlan, 1993)

Source

- This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.
-

  Attribute Information:
- 1. mpg: continuous
  2. cylinders: multi-valued discrete
  3. displacement: continuous
  4. horsepower: continuous
  5. weight: continuous
  6. acceleration: continuous
  7. model year: multi-valued discrete
  8. origin: multi-valued discrete
  9. car name: string (unique for each instance)
-

CASE STUDY

- Firstly we installed pandas as pip install pandas in terminal.
- Imported .csv file and gave column names.
- Using head command output is shown.
- Head command is used to show first 5 data.

CODE

```
import numpy as np
import pandas as pd

fname="/content/drive/MyDrive/auto-mpg.csv"
column=["mpg","cylinders","displacement","horsepower","weight","acceleration","model year","origin","car name"]
data=pd.read_csv(fname,delim_whitespace=True,names=column)
data.head()
```

Output for above code:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

Checked if any null values present in dataset.

```
data.isnull().any()
```

```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration    False
model year      False
origin          False
car name        False
dtype: bool
```

Checked data types of columns in dataset.

```
[18] data.dtypes
```

```
mpg              float64
cylinders          int64
displacement     float64
horsepower        object
weight           float64
acceleration     float64
model year         int64
origin             int64
car name          object
dtype: object
```

Observing the data we can find that there is '?‘ in horsepower column which is used as a placeholder for missing values. So, remove entries having ‘?’ .

```
[27] data = data[data.horsepower != '?']
```

```
print('?' in data.horsepower)
```

```
False
```

```
[30] data.dtypes
```

```
mpg              float64
cylinders          int64
displacement     float64
horsepower        object
weight           float64
acceleration     float64
model year         int64
origin             int64
car name          object
dtype: object
```

However, we can observe that the horsepower data is still an object type and not float. That is because pandas forcefully assigned the entire column as object when we imported the data set due to '?', so it is changed as below.

```
[31] data.horsepower = data.horsepower.astype('float')
     data.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight          float64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

Pandas describe() is used to view some basic statistical details like count , mean, std etc. of a data frame or a series of numeric values.

Dataset is described as below : -

```
data.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000
mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.541327	75.979592	1.576531
std	7.805007	1.705783	104.644004	38.491160	849.402560	2.758864	3.683737	0.805518
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.000000	4.000000	105.000000	75.000000	2225.250000	13.775000	73.000000	1.000000
50%	22.750000	4.000000	151.000000	93.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	275.750000	126.000000	3614.750000	17.025000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

DATA CLEANING

- ***Data cleaning or cleansing*** *is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.*
- Here as car names are unique so it is just used for identification purpose so here it is dropped as below.

```
to_drop=['car name']
data.drop(to_drop, inplace=True, axis=1)
data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1

Here we have added country code column as below used command.

```
[72] data['Country_code'] = data.origin.replace([1,2,3],['INDIA','AUSTRALIA','CANADA'])
     data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	Country_code
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	INDIA
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	INDIA
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	INDIA
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	INDIA
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	INDIA

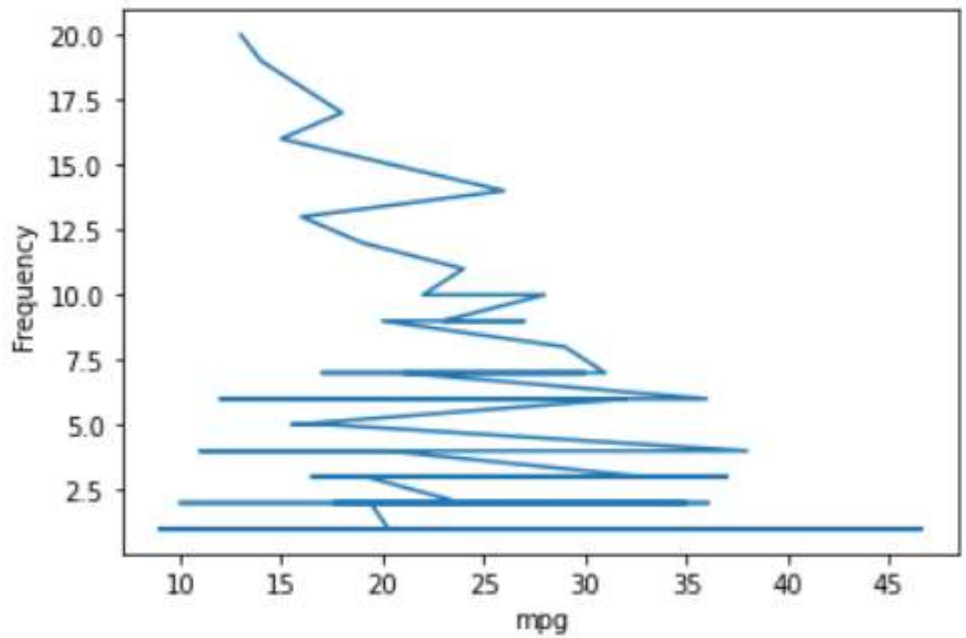
VISUALIZATION OF DATA

- Data Visualization is the process of communicating complex information with simple graphics and charts.
- Data Visualization has the power to tell data-driven stories while allowing people to see patterns and relationships found in data.

- **Frequency distribution of mpg as below:**

```
data['mpg'].value_counts()[:].plot(xlabel='mpg',ylabel='Frequency')
```

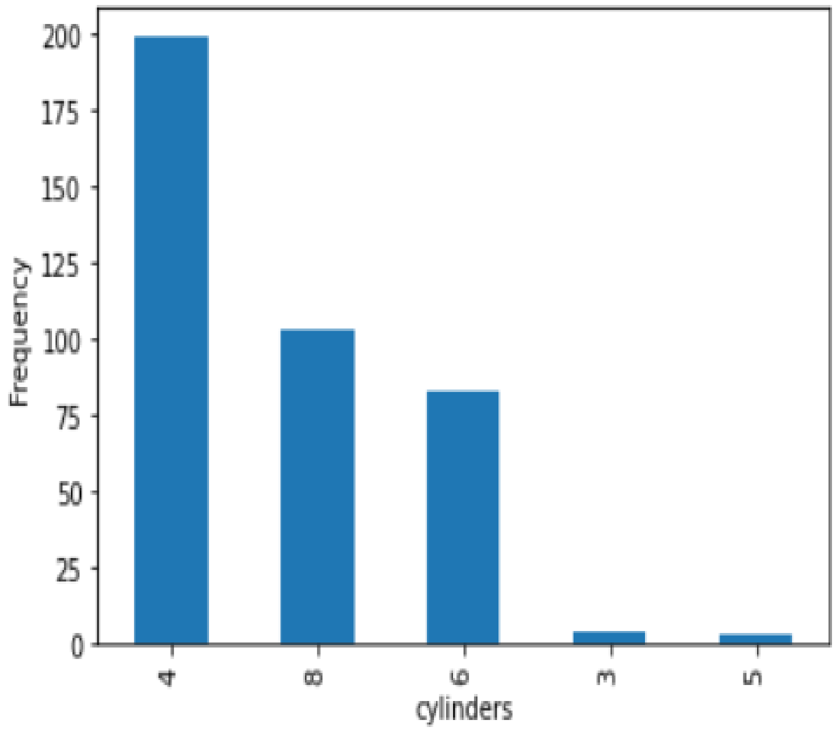
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2e4db2b0>
```



Frequency distribution of cylinders

```
data['cylinders'].value_counts()[:].plot(xlabel='cylinders',ylabel='Frequency',kind='bar')
```

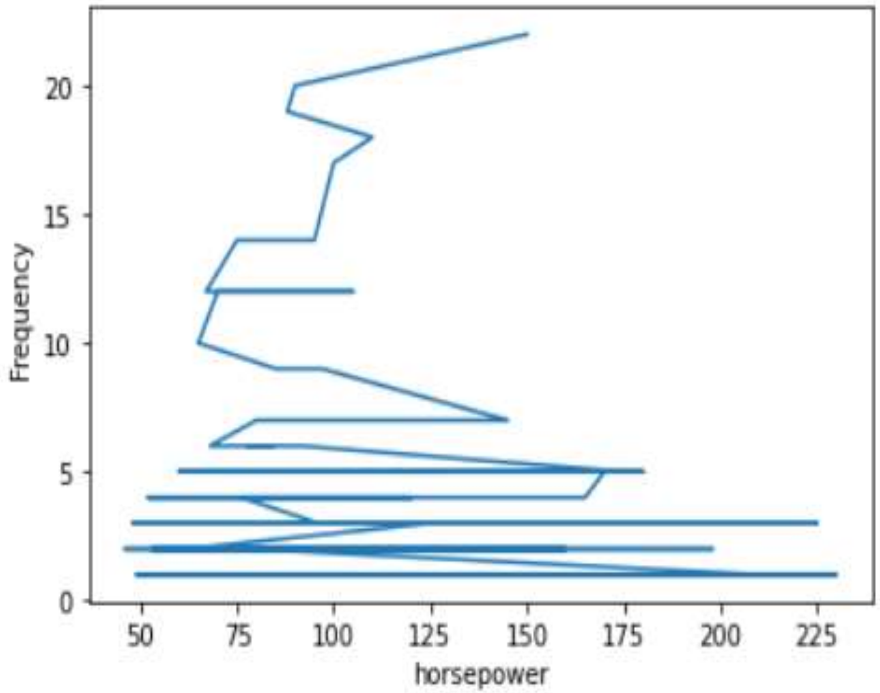
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3cc9426710>
```



Frequency distribution of horsepower

```
[32] data['horsepower'].value_counts()[:].plot(xlabel='horsepower',ylabel='Frequency')
```

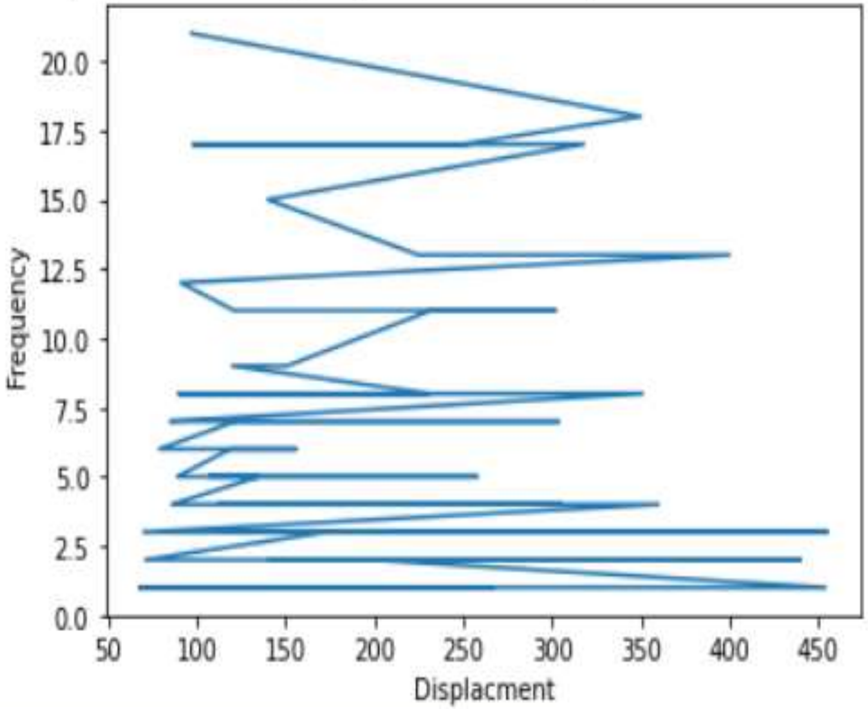
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2d0a2198>
```



Frequency distribution of displacement

```
[34] data['displacement'].value_counts()[:].plot(xlabel='Displacment',ylabel='Frequency')
```

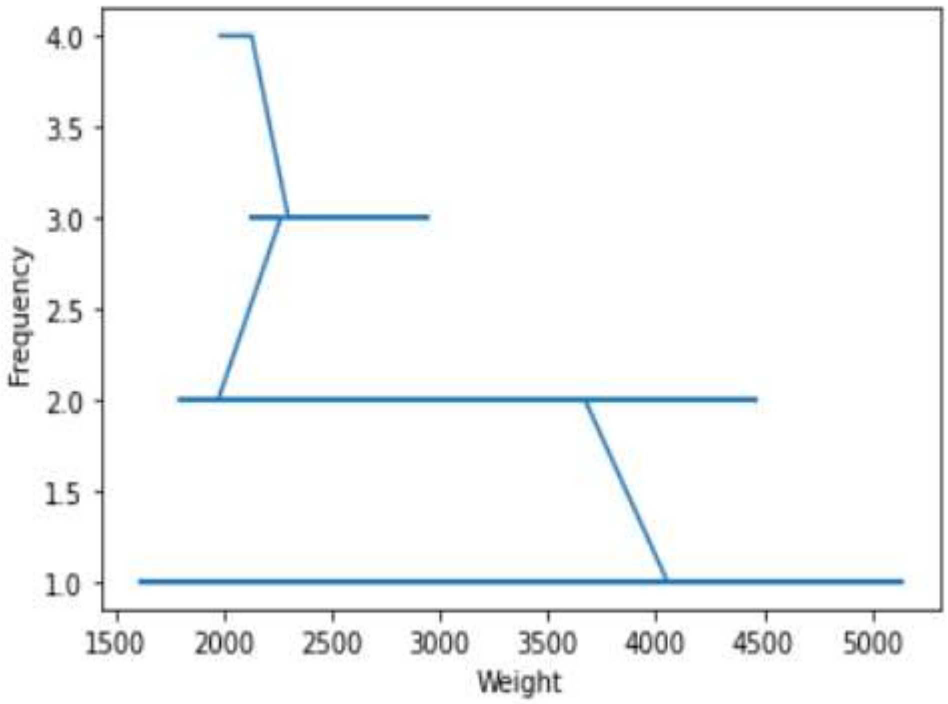
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2e1d6588>
```



Frequency distribution for weight

```
[35] data['weight'].value_counts()[:].plot(xlabel='Weight',ylabel='Frequency')
```

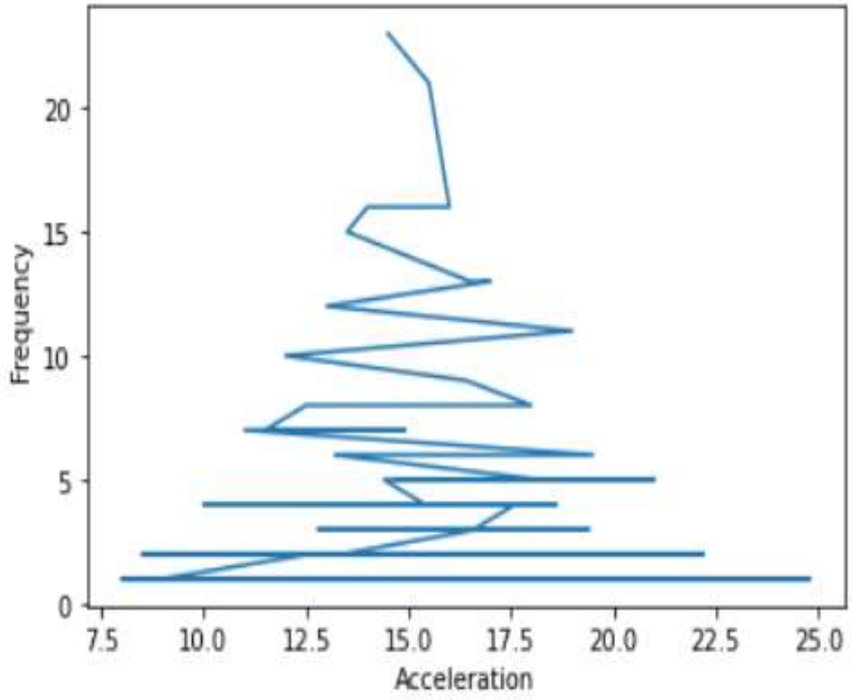
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2e188e10>
```



Frequency distribution of acceleration.

```
[36] data['acceleration'].value_counts()[:].plot(xlabel='Acceleration',ylabel='Frequency')
```

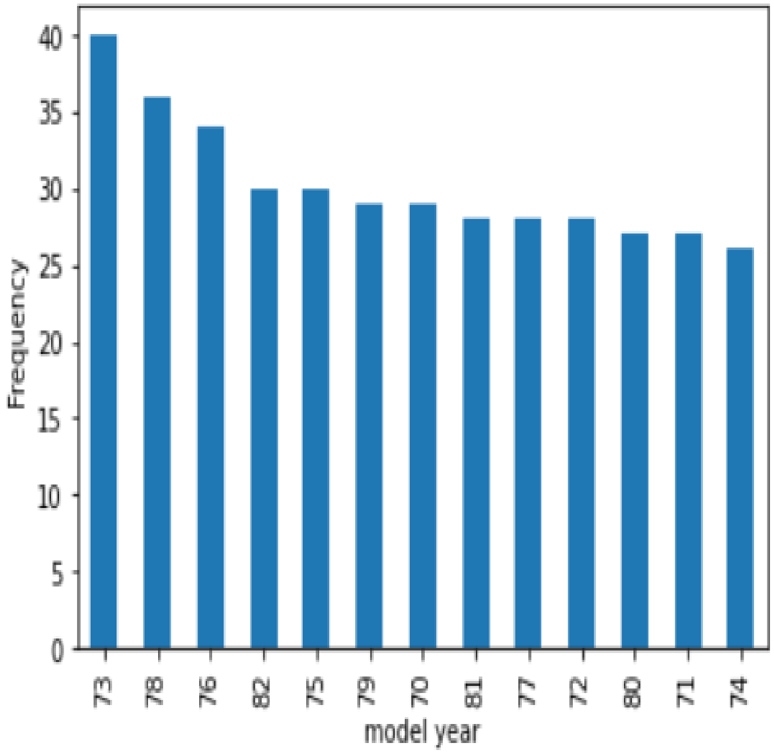
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2e216470>
```



Frequency distribution of model year.

```
[ ]  data['model year'].value_counts()[:].plot(xlabel='model year',ylabel='Frequency',kind='bar')
```

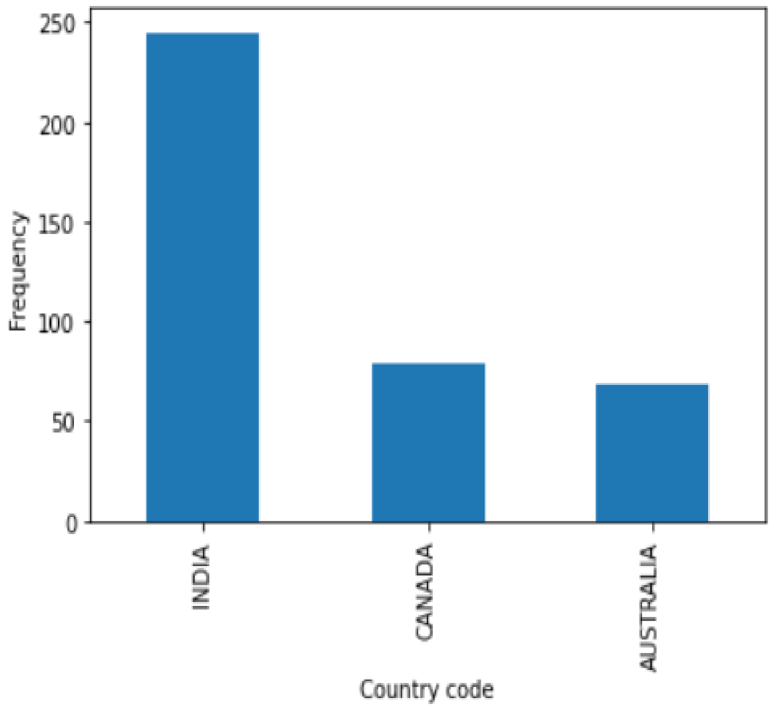
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3cc98ee4e0>
```



Frequency distribution of country code.

```
data['Country_code'].value_counts()[:].plot(xlabel='Country code',ylabel='Frequency',kind='bar')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f9d2d550f60>



Correlation matrix

```
data.corr()
```

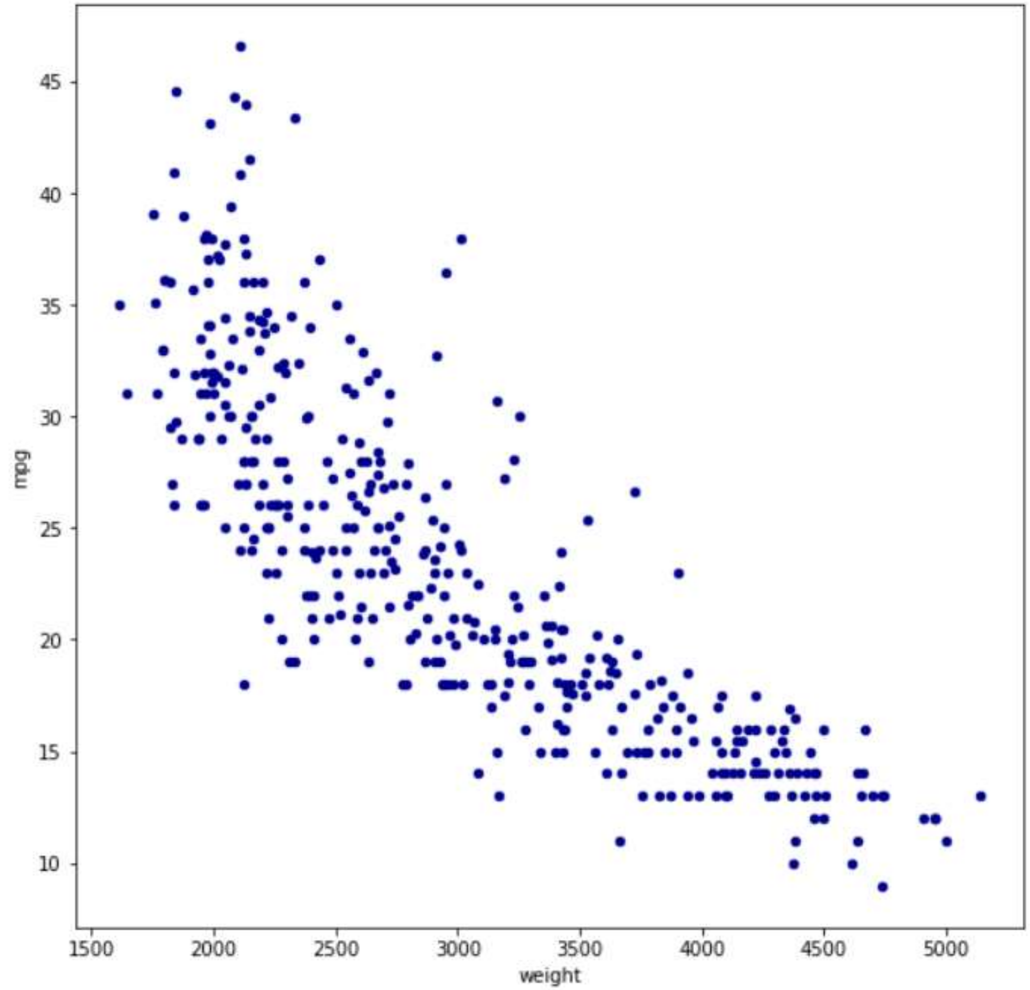
	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
mpg	1.000000	-0.775396	-0.804203	-0.771437	-0.831741	0.420289	0.579267	0.563450
cylinders	-0.775396	1.000000	0.950721	0.838939	0.896017	-0.505419	-0.348746	-0.562543
displacement	-0.804203	0.950721	1.000000	0.893646	0.932824	-0.543684	-0.370164	-0.609409
horsepower	-0.771437	0.838939	0.893646	1.000000	0.860574	-0.684259	-0.411651	-0.453669
weight	-0.831741	0.896017	0.932824	0.860574	1.000000	-0.417457	-0.306564	-0.581024
acceleration	0.420289	-0.505419	-0.543684	-0.684259	-0.417457	1.000000	0.288137	0.205873
model year	0.579267	-0.348746	-0.370164	-0.411651	-0.306564	0.288137	1.000000	0.180662
origin	0.563450	-0.562543	-0.609409	-0.453669	-0.581024	0.205873	0.180662	1.000000

We can see that there is maximum (negative) correlation between mpg and weight.

By the scatterplot between mpg and weight, we can see that they have almost linear, negative relationship. So we will try to fit a simple linear regression model to this.

```
data.plot.scatter('weight','mpg',c='DarkBlue', figsize= (9,9))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d5c8115b88>
```



Train – test split

- We are randomly splitting the data into two parts, in the ratio 80:20.
- The bigger par of he data will be used to train our linear regression model, while the smaller part will be used to test the performance of our model on an unknown dataset.

```
import random
random.seed(100)
```

```
msk = np.random.rand(len(data)) < 0.8
train = data[msk]
test = data[~msk]
```

```
train.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	Country_code
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	INDIA
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	INDIA
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	INDIA
5	15.0	8	429.0	198.0	4341.0	10.0	70	1	INDIA
6	14.0	8	454.0	220.0	4354.0	9.0	70	1	INDIA

```
test.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	Country_code
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	INDIA
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	INDIA
10	15.0	8	383.0	170.0	3563.0	10.0	70	1	INDIA
14	24.0	4	113.0	95.0	2372.0	15.0	70	3	CANADA
16	18.0	6	199.0	97.0	2774.0	15.5	70	1	INDIA

Extracting the variables we require – weight, and mpg, and transforming the pandas series to numpy and reshaping them.

```
x = train.iloc[:,4]
y = train.iloc[:,0]
xt = test.iloc[:,4]
yt = test.iloc[:,0]
```

```
train_x = np.asanyarray(x).reshape(-1,1)
train_y = np.asanyarray(y).reshape(-1,1)
test_x = np.asanyarray(xt).reshape(-1,1)
test_y = np.asanyarray(yt).reshape(-1,1)
```

What is regression?

- **Regression analysis** is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates', or 'features')
- Types of Regression:
  - Linear
  - Polynomial
  - Multiple linear
  - Logistic
  - Many more advanced versions

Simple Linear Regression

- Simple linear regression or SLR is a method to help us understand the relationship between two variables,
  - The predictor independent variable x
  - and the target dependent variable y.

$$y = \beta_0 + \beta_1 x + \varepsilon$$

- The equation for linear model is:

$$\hat{y} = b_0 + b_1 x$$

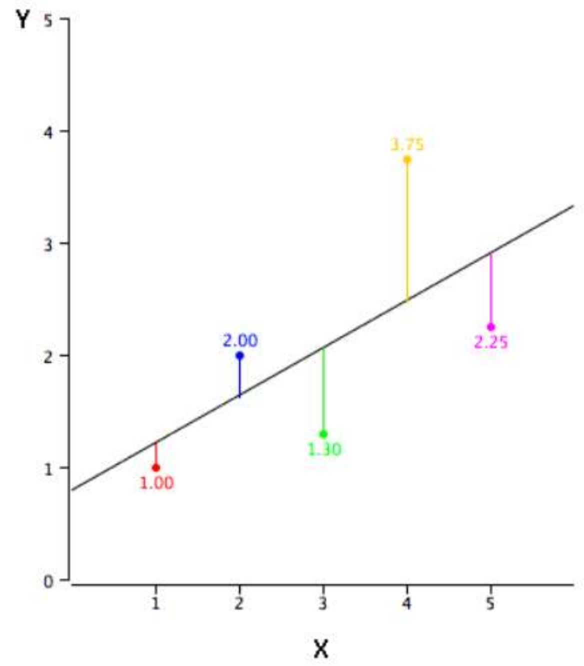


Figure 2. A scatter plot of the example data. The black line consists of the predictions, the points are the actual data, and the vertical lines between the points and the black line represent errors of prediction.

Simple Regression Linear regression

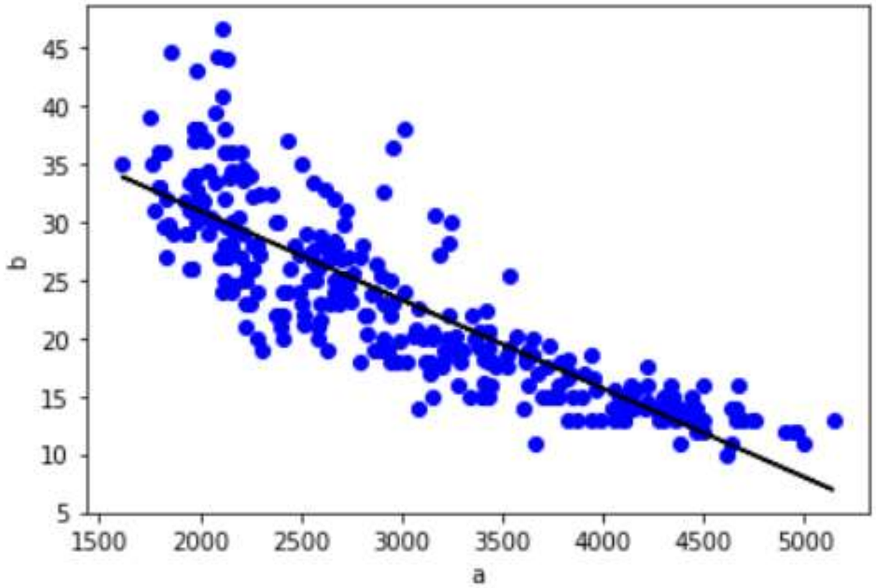
```
from sklearn import linear_model
regr = linear_model.LinearRegression()

regr.fit (train_x, train_y)
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[-0.00764876]]
Intercept:  [46.244169]
```

Plotting the regression line

```
plt.scatter(train_x, train_y,  color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], 'black')
plt.xlabel("a")
plt.ylabel("b")
```

```
Text(0, 0.5, 'b')
```

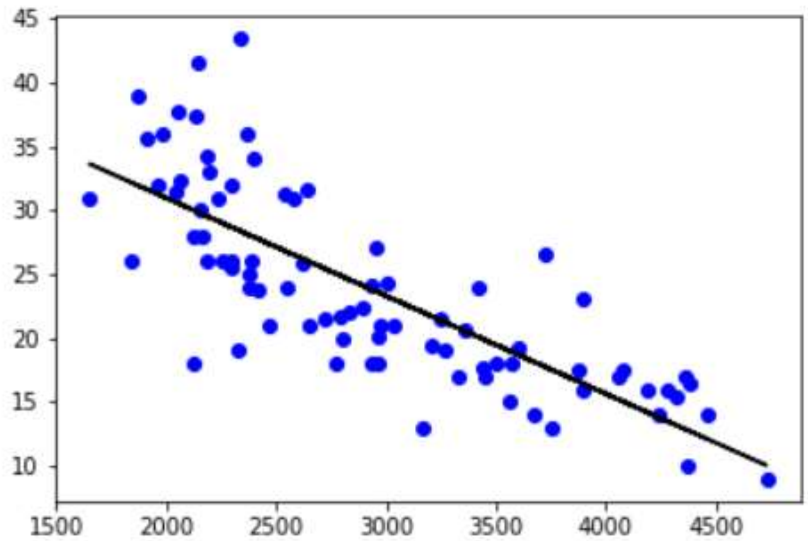


Predicting the values for the test set.

```
test_y_hat = regr.predict(test_x)
```

```
plt.scatter(test_x, test_y,  color='blue')
plt.plot(test_x, test_y_hat,  color='black')
```

```
[<matplotlib.lines.Line2D at 0x23586846d88>]
```



Evaluation metrics

Mean Absolute error

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Mean Absolute Error formula

Mean squared error

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Mean Square Error formula

R Squared

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i(y_i-\hat{y}_i)^2}{\sum_i(y_i-\bar{y})^2}$$

R square formula

Finding the evaluation metrics

```
from sklearn.metrics import r2_score
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_hat , test_y) )
```

```
Mean absolute error: 3.75
Residual sum of squares (MSE): 22.07
R2-score: 0.37
```

Polynomial Regression

- Quadratic – 2nd order

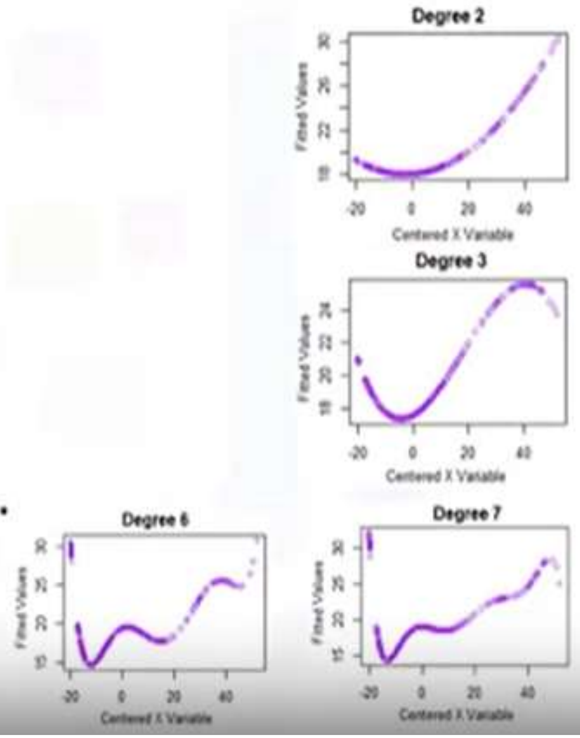
$$\hat{Y} = b_0 + b_1\, x_1 + b_2(\, x_1)^2$$

- Cubic – 3rd order

$$\hat{Y} = b_0 + b_1\, x_1 + b_2(\, x_1)^2 + b_3(\, x_1)^3$$

- Higher order

$$\hat{Y} = b_0 + b_1\, x_1 + b_2(\, x_1)^2 + b_3(\, x_1)^3 + ..$$



Transforming the data into polynomial matrix

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
```

```
train_x_poly
```

```
array([[1.0000000e+00, 3.6930000e+03, 1.3638249e+07],
       [1.0000000e+00, 3.4360000e+03, 1.1806096e+07],
       [1.0000000e+00, 3.4330000e+03, 1.1785489e+07],
       [1.0000000e+00, 4.3410000e+03, 1.8844281e+07],
       [1.0000000e+00, 4.3540000e+03, 1.8957316e+07],
       [1.0000000e+00, 4.3120000e+03, 1.8593344e+07],
       [1.0000000e+00, 4.4250000e+03, 1.9580625e+07],
       [1.0000000e+00, 3.8500000e+03, 1.4822500e+07],
       [1.0000000e+00, 3.6090000e+03, 1.3024881e+07],
       [1.0000000e+00, 3.7610000e+03, 1.4145121e+07],
       [1.0000000e+00, 3.0860000e+03, 9.5233960e+06],
       [1.0000000e+00, 2.8330000e+03, 8.0258890e+06],
       [1.0000000e+00, 2.5870000e+03, 6.6925690e+06],
       [1.0000000e+00, 2.1300000e+03, 4.5369000e+06],
       [1.0000000e+00, 2.6720000e+03, 7.1395840e+06],
       [1.0000000e+00, 2.4300000e+03, 5.9049000e+06],
       [1.0000000e+00, 2.2340000e+03, 4.9907560e+06],
       [1.0000000e+00, 4.6150000e+03, 2.1298225e+07],
       [1.0000000e+00, 4.3820000e+03, 1.9201924e+07],
       [1.0000000e+00, 2.1300000e+03, 4.5369000e+06]
```

Polynomial Regression of degree 2

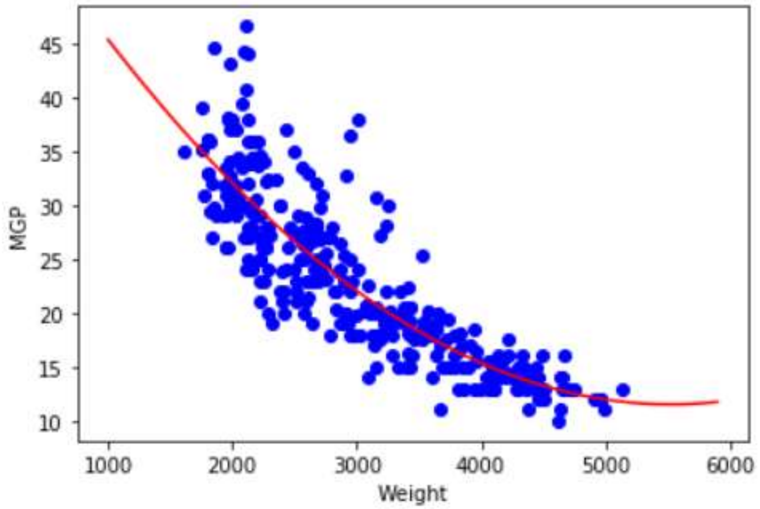
```
regr1 = linear_model.LinearRegression()
regr1.fit (train_x_poly, train_y)
# The coefficients
print ('Coefficients: ', regr1.coef_)
print ('Intercept: ',regr1.intercept_)
```

```
Coefficients:  [[ 0.00000000e+00 -1.82594373e-02  1.65272526e-06]]
Intercept:  [61.955339]
```

Plotting the regression curve

```
plt.scatter(train_x, train_y,  color='blue')
XX = np.arange(1000, 6000, 100)
yy = regr1.intercept_[0]+ regr1.coef_[0][1]*XX+ regr1.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Weight")
plt.ylabel("MGP")
```

```
Text(0, 0.5, 'MGP')
```

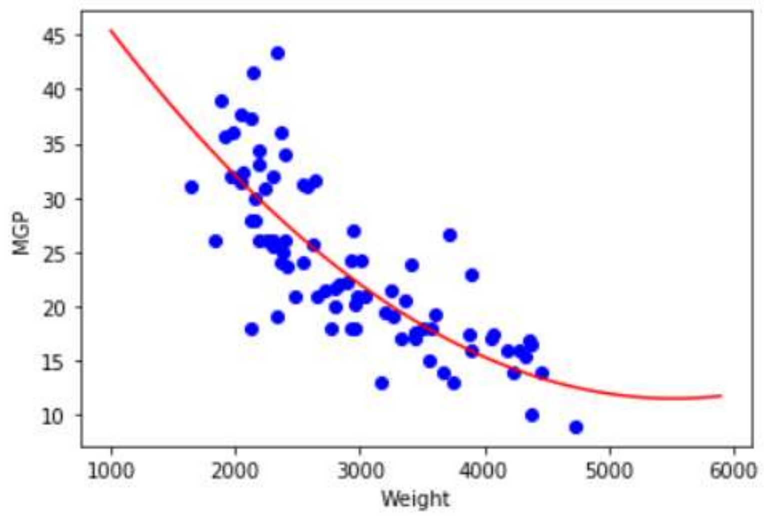


Predicting mpg value for test_y

```
test_x_poly = poly.fit_transform(test_x)
test_y_hat = regr1.predict(test_x_poly)
```

```
plt.scatter(test_x, test_y,  color='blue')
XX = np.arange(1000, 6000, 100)
yy = regr1.intercept_[0]+ regr1.coef_[0][1]*XX+ regr1.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Weight")
plt.ylabel("MGP")
```

```
Text(0, 0.5, 'MGP')
```



Evaluation

```
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_hat , test_y) )
```

```
Mean absolute error: 3.52
Residual sum of squares (MSE): 20.79
R2-score: 0.46
```

	Linear Regression	**Polynomial Regression**
MAE	3.75	3.52
MSE	22.07	20.79
R2_score	0.37	0.46

Can we do better?

- There are more complex regression techniques that have been developed.
- Multiple regression can also be used, but feature selection needs to be done before that, because as we could see from our correlation matrix, the correlation between the predictors themselves are high, and this would lead to error in the analysis.

Thank you