Regression algorithm project

Algorithms in this notebook

1) Linear Regression 2) Ridge Regression 3) Lasso Regression 4) ElasticNet Regression

In [1]:
```
#import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#ignore/ disable warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv(r"C:\Users\Sanjay Lohar\Downloads\automobile data.csv")
```

In [3]:
```
df
```

Out[3]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year	Origin	Car_Name
0	8.0	8	307.0	130	3504	12.0	2015	1	chevrolet
1	15.0	8	350.0	165	3693	11.5	2015	1	buick
2	18.0	8	318.0	150	3436	11.0	2015	1	plymouth
3	16.0	8	304.0	150	3433	12.0	2015	1	amc
4	17.0	8	302.0	140	3449	10.5	2015	1	ford
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	2003	1	ford
394	44.0	4	97.0	52	2130	24.6	2003	2	volkswagen
395	32.0	4	135.0	84	2295	11.6	2003	1	dodge
396	28.0	4	120.0	79	2625	18.6	2003	1	ford
397	31.0	4	119.0	82	2720	19.4	2003	1	chevrolet

398 rows × 9 columns

In [4]:
```
df.head(35)
```

Out[4]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year	Origin	Car_Name
0	8.0	8	307.0	130	3504	12.0	2015	1	chevrolet
1	15.0	8	350.0	165	3693	11.5	2015	1	buick
2	18.0	8	318.0	150	3436	11.0	2015	1	plymouth
3	16.0	8	304.0	150	3433	12.0	2015	1	amc
4	17.0	8	302.0	140	3449	10.5	2015	1	ford
5	15.0	8	429.0	198	4341	10.0	2015	1	ford
6	14.0	8	454.0	220	4354	9.0	2015	1	chevrolet
7	14.0	8	440.0	215	4312	8.5	2015	1	plymouth
8	14.0	8	455.0	225	4425	10.0	2015	1	pontiac
9	15.0	8	390.0	190	3850	8.5	2015	1	amc
10	15.0	8	383.0	170	3563	10.0	2015	1	dodge
11	14.0	8	340.0	160	3609	8.0	2015	1	plymouth
12	15.0	8	400.0	150	3761	9.5	2015	1	chevrolet
13	14.0	8	455.0	225	3086	10.0	2015	1	buick
14	24.0	4	113.0	95	2372	15.0	2015	3	toyota
15	22.0	6	198.0	95	2833	15.5	2015	1	plymouth
16	18.0	6	199.0	97	2774	15.5	2015	1	amc
17	21.0	6	200.0	85	2587	16.0	2015	1	ford
18	27.0	4	97.0	88	2130	14.5	2015	3	datsun
19	26.0	4	97.0	46	1835	20.5	2015	2	volkswagen
20	25.0	4	110.0	87	2672	17.5	2015	2	peugeot
21	24.0	4	107.0	90	2430	14.5	2015	2	audi
22	25.0	4	104.0	95	2375	17.5	2015	2	saab
23	26.0	4	121.0	113	2234	12.5	2015	2	bmw
24	21.0	6	199.0	90	2648	15.0	2015	1	amc
25	10.0	8	360.0	215	4615	14.0	2015	1	ford
26	10.0	8	307.0	200	4376	15.0	2015	1	chevrolet
27	11.0	8	318.0	210	4382	13.5	2015	1	dodge
28	9.0	8	304.0	193	4732	18.5	2015	1	honda
29	27.0	4	97.0	88	2130	14.5	2014	3	datsun
30	28.0	4	140.0	90	2264	15.5	2014	1	chevrolet
31	25.0	4	113.0	95	2228	14.0	2014	3	toyota
32	25.0	4	98.0	?	2046	19.0	2014	1	ford
33	19.0	6	232.0	100	2634	13.0	2014	1	amc
34	16.0	6	225.0	105	3439	15.5	2014	1	plymouth

In [5]:

```
#print the number of rows and columns
df.shape
```

Out[5]:

```
(398, 9)
```

In [6]:

```
#print descriptive statistics
df.describe()
```

Out[6]:

	MPG	Cylinders	Displacement	Weight	Acceleration	Model_year	Origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.489447	5.454774	193.425879	2970.424623	15.568090	2008.989950	1.572864
std	7.849757	1.701004	104.269838	846.841774	2.757689	3.697627	0.802055
min	8.000000	3.000000	68.000000	1613.000000	8.000000	2003.000000	1.000000
25%	17.125000	4.000000	104.250000	2223.750000	13.825000	2006.000000	1.000000
50%	23.000000	4.000000	148.500000	2803.500000	15.500000	2009.000000	1.000000
75%	29.000000	8.000000	262.000000	3608.000000	17.175000	2012.000000	2.000000
max	46.600000	8.000000	455.000000	5140.000000	24.800000	2015.000000	3.000000

In [7]:

```
#check data type
df.dtypes
```

Out[7]:
```
MPG             float64
Cylinders         int64
Displacement    float64
Horsepower       object
Weight            int64
Acceleration    float64
Model_year        int64
Origin            int64
Car_Name         object
dtype: object
```

In [8]:
```
#change the data type of Horsepower as it is a numeric data stored as
#object
# errors='coerce' means replace all non numeric values(e.g. "apple", ?, - or any other signs) with NaN
df['Horsepower']=pd.to_numeric(df['Horsepower'],errors='coerce')
df['Horsepower']
```

Out[8]:
```
0      130.0
1      165.0
2      150.0
3      150.0
4      140.0
       ...
393     86.0
394     52.0
395     84.0
396     79.0
397     82.0
Name: Horsepower, Length: 398, dtype: float64
```

In [9]:
```
df.head(35)
```

Out[9]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year	Origin	Car_Name
0	8.0	8	307.0	130.0	3504	12.0	2015	1	chevrolet
1	15.0	8	350.0	165.0	3693	11.5	2015	1	buick
2	18.0	8	318.0	150.0	3436	11.0	2015	1	plymouth
3	16.0	8	304.0	150.0	3433	12.0	2015	1	amc
4	17.0	8	302.0	140.0	3449	10.5	2015	1	ford
5	15.0	8	429.0	198.0	4341	10.0	2015	1	ford
6	14.0	8	454.0	220.0	4354	9.0	2015	1	chevrolet
7	14.0	8	440.0	215.0	4312	8.5	2015	1	plymouth
8	14.0	8	455.0	225.0	4425	10.0	2015	1	pontiac
9	15.0	8	390.0	190.0	3850	8.5	2015	1	amc
10	15.0	8	383.0	170.0	3563	10.0	2015	1	dodge
11	14.0	8	340.0	160.0	3609	8.0	2015	1	plymouth
12	15.0	8	400.0	150.0	3761	9.5	2015	1	chevrolet
13	14.0	8	455.0	225.0	3086	10.0	2015	1	buick
14	24.0	4	113.0	95.0	2372	15.0	2015	3	toyota
15	22.0	6	198.0	95.0	2833	15.5	2015	1	plymouth
16	18.0	6	199.0	97.0	2774	15.5	2015	1	amc
17	21.0	6	200.0	85.0	2587	16.0	2015	1	ford
18	27.0	4	97.0	88.0	2130	14.5	2015	3	datsun
19	26.0	4	97.0	46.0	1835	20.5	2015	2	volkswagen
20	25.0	4	110.0	87.0	2672	17.5	2015	2	peugeot
21	24.0	4	107.0	90.0	2430	14.5	2015	2	audi
22	25.0	4	104.0	95.0	2375	17.5	2015	2	saab
23	26.0	4	121.0	113.0	2234	12.5	2015	2	bmw
24	21.0	6	199.0	90.0	2648	15.0	2015	1	amc
25	10.0	8	360.0	215.0	4615	14.0	2015	1	ford
26	10.0	8	307.0	200.0	4376	15.0	2015	1	chevrolet
27	11.0	8	318.0	210.0	4382	13.5	2015	1	dodge
28	9.0	8	304.0	193.0	4732	18.5	2015	1	honda
29	27.0	4	97.0	88.0	2130	14.5	2014	3	datsun
30	28.0	4	140.0	90.0	2264	15.5	2014	1	chevrolet
31	25.0	4	113.0	95.0	2228	14.0	2014	3	toyota
32	25.0	4	98.0	NaN	2046	19.0	2014	1	ford
33	19.0	6	232.0	100.0	2634	13.0	2014	1	amc
34	16.0	6	225.0	105.0	3439	15.5	2014	1	plymouth

In [10]:

```
# recheck data type again to see horsepowers data type has been changed or not
df.dtypes
```

Out[10]:

```
MPG             float64
Cylinders         int64
Displacement    float64
Horsepower      float64
Weight            int64
Acceleration    float64
Model_year        int64
Origin            int64
Car_Name         object
dtype: object
```

In [11]:

```
# ----- Missing value imputation -----
```

In [12]:

```
#check missing values
df.isnull().sum()
```

Out[12]:
```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model_year      0
Origin          0
Car_Name        0
dtype: int64
```

In [13]:
```
# There are 6 missing values in Horsepower variable
```

In [14]:
```
#missing value treatment
#impute missing values
df['Horsepower']=df['Horsepower'].fillna(df['Horsepower'].median())
```

In [15]:
```
df.head(35)
```

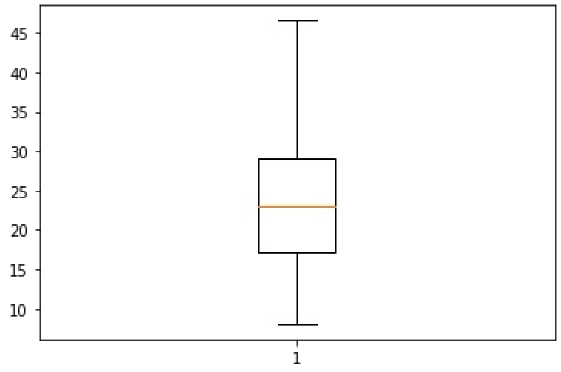
Out[15]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year	Origin	Car_Name
0	8.0	8	307.0	130.0	3504	12.0	2015	1	chevrolet
1	15.0	8	350.0	165.0	3693	11.5	2015	1	buick
2	18.0	8	318.0	150.0	3436	11.0	2015	1	plymouth
3	16.0	8	304.0	150.0	3433	12.0	2015	1	amc
4	17.0	8	302.0	140.0	3449	10.5	2015	1	ford
5	15.0	8	429.0	198.0	4341	10.0	2015	1	ford
6	14.0	8	454.0	220.0	4354	9.0	2015	1	chevrolet
7	14.0	8	440.0	215.0	4312	8.5	2015	1	plymouth
8	14.0	8	455.0	225.0	4425	10.0	2015	1	pontiac
9	15.0	8	390.0	190.0	3850	8.5	2015	1	amc
10	15.0	8	383.0	170.0	3563	10.0	2015	1	dodge
11	14.0	8	340.0	160.0	3609	8.0	2015	1	plymouth
12	15.0	8	400.0	150.0	3761	9.5	2015	1	chevrolet
13	14.0	8	455.0	225.0	3086	10.0	2015	1	buick
14	24.0	4	113.0	95.0	2372	15.0	2015	3	toyota
15	22.0	6	198.0	95.0	2833	15.5	2015	1	plymouth
16	18.0	6	199.0	97.0	2774	15.5	2015	1	amc
17	21.0	6	200.0	85.0	2587	16.0	2015	1	ford
18	27.0	4	97.0	88.0	2130	14.5	2015	3	datsun
19	26.0	4	97.0	46.0	1835	20.5	2015	2	volkswagen
20	25.0	4	110.0	87.0	2672	17.5	2015	2	peugeot
21	24.0	4	107.0	90.0	2430	14.5	2015	2	audi
22	25.0	4	104.0	95.0	2375	17.5	2015	2	saab
23	26.0	4	121.0	113.0	2234	12.5	2015	2	bmw
24	21.0	6	199.0	90.0	2648	15.0	2015	1	amc
25	10.0	8	360.0	215.0	4615	14.0	2015	1	ford
26	10.0	8	307.0	200.0	4376	15.0	2015	1	chevrolet
27	11.0	8	318.0	210.0	4382	13.5	2015	1	dodge
28	9.0	8	304.0	193.0	4732	18.5	2015	1	honda
29	27.0	4	97.0	88.0	2130	14.5	2014	3	datsun
30	28.0	4	140.0	90.0	2264	15.5	2014	1	chevrolet
31	25.0	4	113.0	95.0	2228	14.0	2014	3	toyota
32	25.0	4	98.0	93.5	2046	19.0	2014	1	ford
33	19.0	6	232.0	100.0	2634	13.0	2014	1	amc
34	16.0	6	225.0	105.0	3439	15.5	2014	1	plymouth

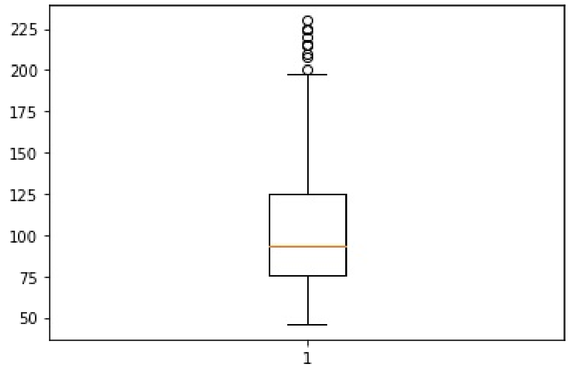
In [16]:
```
#check to ensure that missing values are replaced
df.isnull().sum()
```

Out[16]:
```
MPG             0
Cylinders       0
Displacement    0
Horsepower      0
Weight          0
Acceleration    0
Model_year      0
Origin          0
Car_Name        0
dtype: int64
```

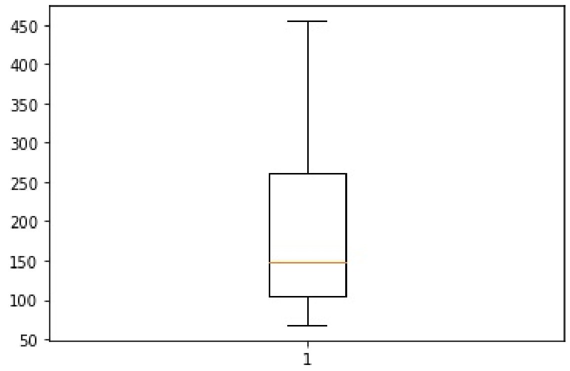
In [17]:
```
# create boxplots to check outliers
plt.boxplot(df['MPG']) #No outlier
plt.show()
```



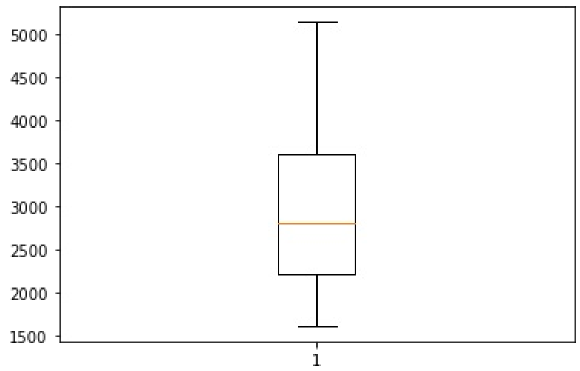
In [18]:
```
#check outliers
plt.boxplot(df['Horsepower']) # Has outlier
plt.show()
```



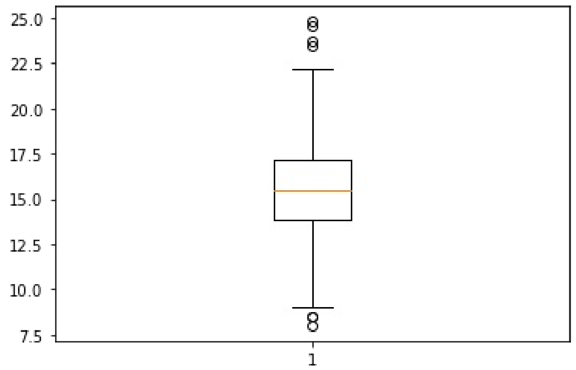
In [19]:
```
plt.boxplot(df['Displacement']) # No outlier
plt.show()
```



In [20]:
```
plt.boxplot(df['Weight']) # No outlier
plt.show()
```



In [21]:
```
plt.boxplot(df['Acceleration']) # Has outlier
plt.show()
```



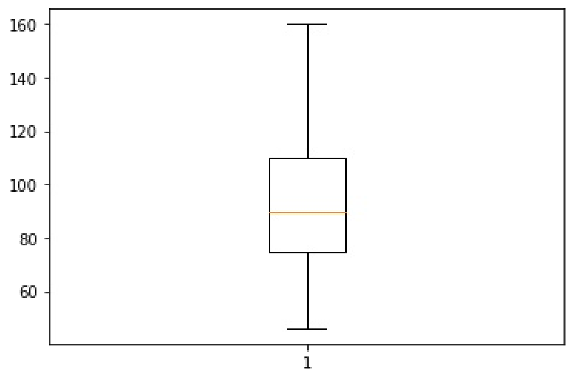
In [22]:
```
#Horsepower and acceleration has outliers
```

In [23]:
```
#user defined function for removing outliers
def remove_outlier(d,c):
    #find q1 and q3
    q1=d[c].quantile(0.25)
    q3=d[c].quantile(0.75)

    #find iqr (inter quartile range)
    iqr=q3-q1

    #find upper and lower bound
    ub=q3+1.5*iqr
    lb=q1-1.5*iqr

    final_data= d[(d[c]>lb) & (d[c]<ub)]
    return final_data
```

In [24]:
```
# to check sample size before removing outliers here & in next step we are removing outliers &
#in previous line we wrote just fn for removing outliers
df.shape # original observations (398rows*9columns)
```

Out[24]:
```
(398, 9)
```

In [29]:
```
#remove outlier from Horsepower
df=remove_outlier(df,'Horsepower')
plt.boxplot(df['Horsepower'])
plt.show()
```

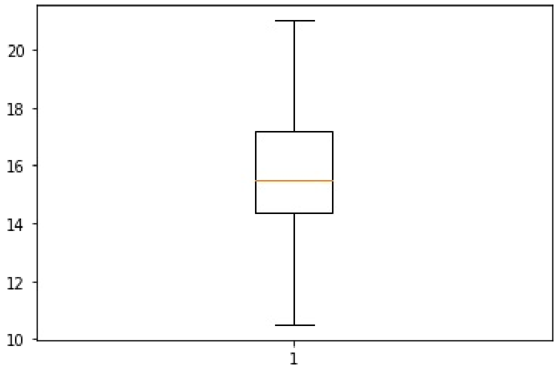


In [30]:
```
#to check data size
df.shape #Horsepower has 37 outlires and removed here therefore sample size is 398-37=361 rows (or obs) * 9 col
```

Out[30]:
```
(361, 9)
```

In [36]:
```
df=remove_outlier(df,'Acceleration')
```

```python
df=remove_outlier(df, 'Acceleration')
plt.boxplot(df['Acceleration'])
plt.show()
```



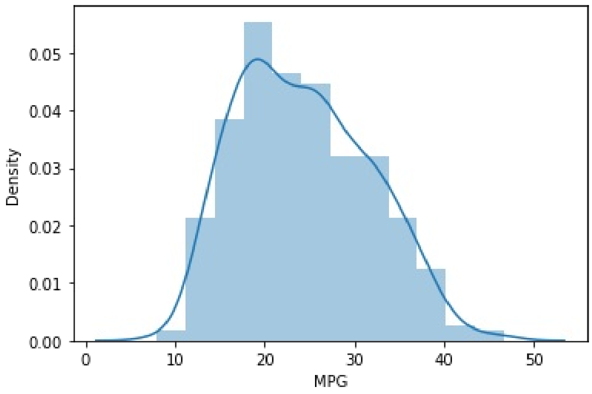
In [37]:
```python
#to check data size
df.shape #Acceleration has 13 outlires and removed here therefore sample size is 398-37-13=348 rows (or obs) *
```

Out[37]:
```
(348, 9)
```

In [38]:
```python
#------- Start EDA (Exploratory Data Analysis) --------------
#----- Data quality test ------------
#Check the distribution of MPG
#Check the distribution of Horsepower
#Check the distribution of Weight
#Check the distribution of Acceleration

#------ Correlation test -----------
#Scatter plot to find the correlation betweeen MPG and Acceleration
#Scatter plot to find the correlation betweeen MPG and Horsepower
#Scatter plot to find the correlation betweeen MPG and Weight
#Scatter plot to find the correlation betweeen MPG and Displacement

#---------- Understand data mix -------
#Barplots:
 #No. of cars by cylinders
 #No. of cars by Origin
 #No. of cars by brand

#------------------------End of EDA -----------------------
```

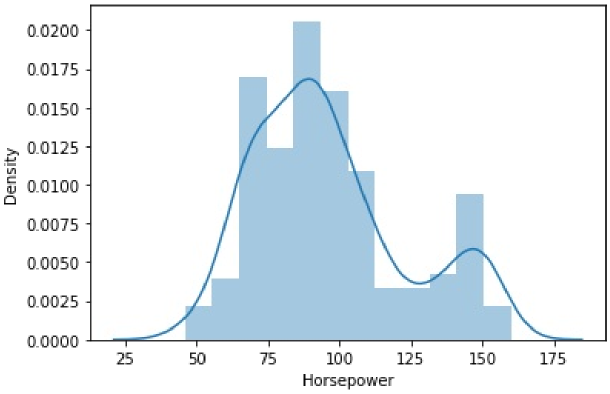
In [39]:
```python
#distribution of MPG
sns.distplot(df['MPG'])
```

Out[39]:
```
<AxesSubplot:xlabel='MPG', ylabel='Density'>
```



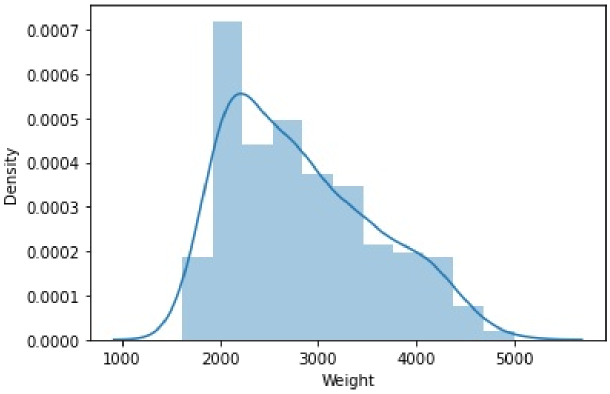
In [40]:
```python
#distribution of Horsepower
sns.distplot(df['Horsepower'])
```

Out[40]:
```
<AxesSubplot:xlabel='Horsepower', ylabel='Density'>
```



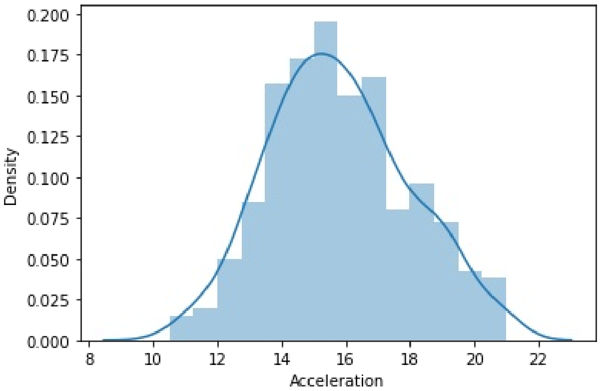
In [41]:
```
#distribution of Weight
sns.distplot(df['Weight'])
```

Out[41]: <AxesSubplot:xlabel='Weight', ylabel='Density'>



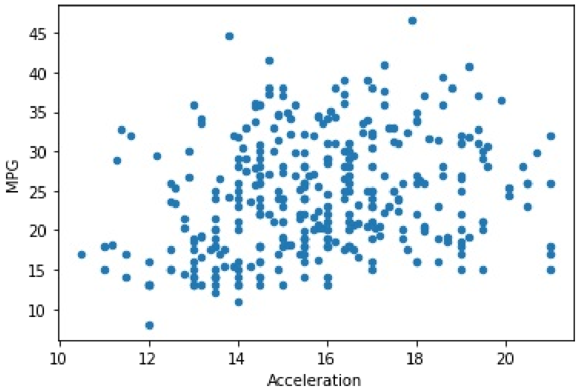
In [42]:
```
#distribution of Acceleration
sns.distplot(df['Acceleration'])
```

Out[42]: <AxesSubplot:xlabel='Acceleration', ylabel='Density'>



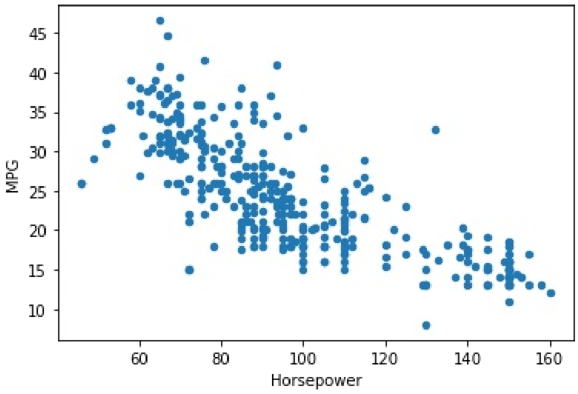
In [43]:
```
#Scatter plot to find the correlation betweeen MPG and Acceleration
df.plot(kind='scatter', x='Acceleration', y='MPG') # Medium to Weak +ve correlation
```

Out[43]: <AxesSubplot:xlabel='Acceleration', ylabel='MPG'>



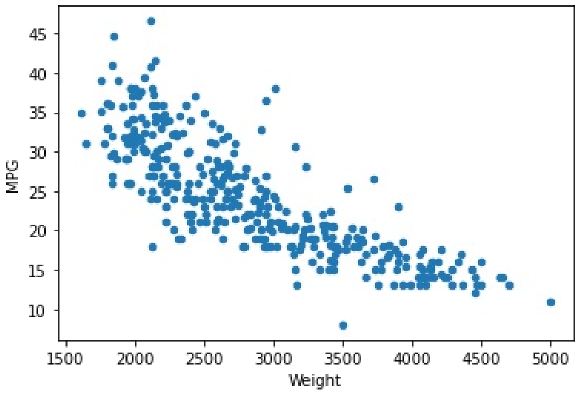
In [44]:
```
#Scatter plot to find the correlation betweeen MPG and Horsepower
df.plot(kind='scatter', x='Horsepower', y='MPG') #strong -ve correlation
```

Out[44]: <AxesSubplot:xlabel='Horsepower', ylabel='MPG'>



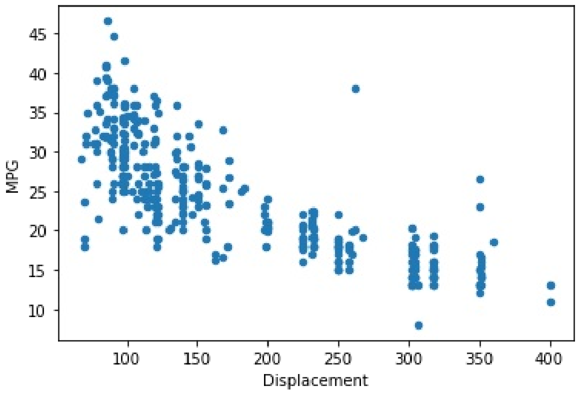
In [45]:
```
#Scatter plot to find the correlation betweeen MPG and Weight
df.plot(kind='scatter', x='Weight', y='MPG') #strong -ve correlation
```

Out[45]: <AxesSubplot:xlabel='Weight', ylabel='MPG'>



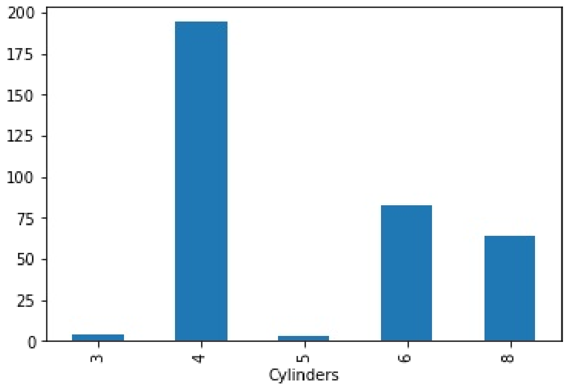
In [46]:
```
#Scatter plot to find the correlation betweeen MPG and Displacement
df.plot(kind='scatter', x='Displacement', y='MPG') #strong -ve correlation
```

Out[46]: <AxesSubplot:xlabel='Displacement', ylabel='MPG'>



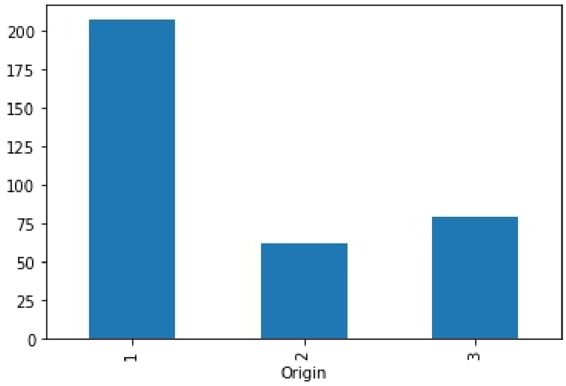
In [51]:
```
#No. of cars by cylinders
df.groupby('Cylinders')['Cylinders'].count().plot(kind='bar')
```

Out[51]: <AxesSubplot:xlabel='Cylinders'>



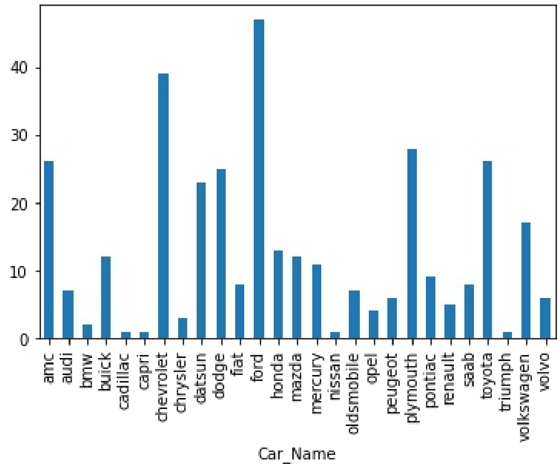
In [52]:
```
 #No. of cars by Origin
df.groupby('Origin')['Origin'].count().plot(kind='bar')
```

Out[52]: <AxesSubplot:xlabel='Origin'>



In [53]:
```
#No. of cars by brand
df.groupby('Car_Name')['Car_Name'].count().plot(kind='bar')
```

Out[53]: <AxesSubplot:xlabel='Car_Name'>



In [54]:
```
#------------------------ End of EDA -----------------
```

In [55]:
```
df.columns
```

Out[55]:
```
Index(['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
       'Acceleration', 'Model_year', 'Origin', 'Car_Name'],
      dtype='object')
```

In [56]:
```
#Check unique values in Origin variable
df['Origin'].unique()
```

Out[56]:
```
array([1, 3, 2], dtype=int64)
```

In [57]:
```
#Replace Origin 1-US, 2-Germany, 3-Japan
df['Origin']=df['Origin'].replace([1,2,3],["US","Germany","Japan"])
```

In [58]:
```
#Print unique entries from origin column again
df['Origin'].unique()
```

Out[58]:
```
array(['US', 'Japan', 'Germany'], dtype=object)
```

In [59]:
```
df.head()
```

Out[59]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year	Origin	Car_Name
0	8.0	8	307.0	130.0	3504	12.0	2015	US	chevrolet
2	18.0	8	318.0	150.0	3436	11.0	2015	US	plymouth
3	16.0	8	304.0	150.0	3433	12.0	2015	US	amc
4	17.0	8	302.0	140.0	3449	10.5	2015	US	ford
14	24.0	4	113.0	95.0	2372	15.0	2015	Japan	toyota

In [60]:
```
#Print correlation plot of numeric columns
#Check the correlation of numeric variables
df_numeric = df.select_dtypes(include=['float64', 'int64'])
df_numeric.head()
```

Out[60]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year
0	8.0	8	307.0	130.0	3504	12.0	2015
2	18.0	8	318.0	150.0	3436	11.0	2015
3	16.0	8	304.0	150.0	3433	12.0	2015
4	17.0	8	302.0	140.0	3449	10.5	2015
14	24.0	4	113.0	95.0	2372	15.0	2015

In [61]:
```
# drop cylinders variable because it is categorical
# & also remove model_year column as it is also a categorical feature
df_numeric = df_numeric.drop(['Cylinders','Model_year'], axis=1)
df_numeric.head()
```

Out[61]:

	MPG	Displacement	Horsepower	Weight	Acceleration
0	8.0	307.0	130.0	3504	12.0
2	18.0	318.0	150.0	3436	11.0
3	16.0	304.0	150.0	3433	12.0
4	17.0	302.0	140.0	3449	10.5
14	24.0	113.0	95.0	2372	15.0

In [62]:

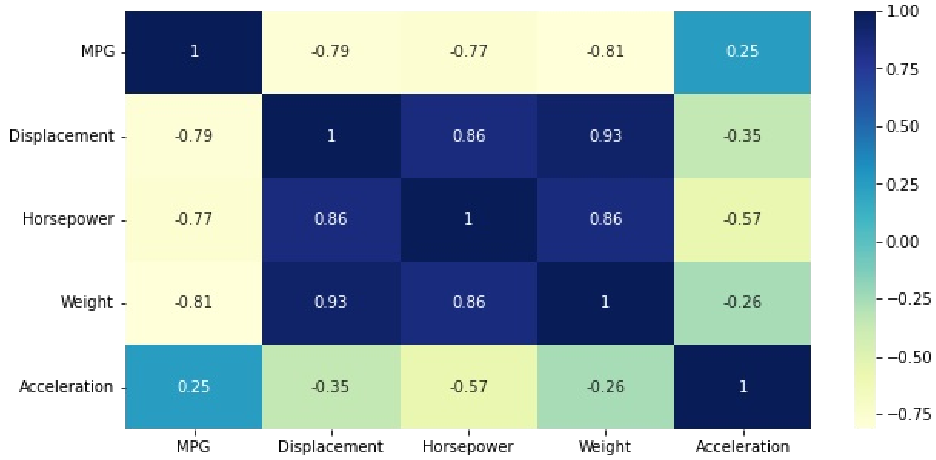
```
# correlation matrix
cor_mat = df_numeric.corr()
cor_mat
```

Out[62]:

	MPG	Displacement	Horsepower	Weight	Acceleration
MPG	1.000000	-0.787844	-0.765025	-0.814122	0.251711
Displacement	-0.787844	1.000000	0.856691	0.934799	-0.349404
Horsepower	-0.765025	0.856691	1.000000	0.864866	-0.568026
Weight	-0.814122	0.934799	0.864866	1.000000	-0.256231
Acceleration	0.251711	-0.349404	-0.568026	-0.256231	1.000000

In [63]:

```
# plot correlations on a heatmap

# figure size
plt.figure(figsize=(10,5))

# heatmap
sns.heatmap(cor_mat, cmap="YlGnBu", annot=True) #YlGnBu
plt.show()
```



In [64]:

```
#Displacement, Horsepower, and Weight are the key features (significant variables)
#to predict the mileage of the cars
```

In [65]:

```
#Cylinders is a categorical variable hence change Cylinders to Object
df['Cylinders']=df['Cylinders'].replace([3,4,5,6,8],
                                        ['3cyl','4cyl','5cyl','6cyl','8cyl'])
```

In [66]:

```
df['Cylinders'].unique()
```

Out[66]:

```
array(['8cyl', '4cyl', '6cyl', '3cyl', '5cyl'], dtype=object)
```

In [67]:

```
# One-hot encoding (dummy conversion)
# create dummy variables for categorical variables

# subset all categorical variables
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

Out[67]:

	Cylinders	Origin	Car_Name
0	8cyl	US	chevrolet
2	8cyl	US	plymouth
3	8cyl	US	amc
4	8cyl	US	ford
14	4cyl	Japan	toyota

In [68]:

```
# convert into dummies
df_dummies = pd.get_dummies(df_categorical)
```

```
df_dummies.head()
```

Out[68]:

	Cylinders_3cyl	Cylinders_4cyl	Cylinders_5cyl	Cylinders_6cyl	Cylinders_8cyl	Origin_Germany	Origin_Japan	Origin_US	Car_Name_amc
0	0	0	0	0	1	0	0	1	0
2	0	0	0	0	1	0	0	1	0
3	0	0	0	0	1	0	0	1	1
4	0	0	0	0	1	0	0	1	0
14	0	1	0	0	0	0	1	0	0

5 rows × 35 columns

In [69]:
```
#combine numeric columns and dummies to create master data
master=pd.concat([df_numeric,df_dummies], axis=1)
```

In [70]:
```
master.head()
```

Out[70]:

	MPG	Displacement	Horsepower	Weight	Acceleration	Cylinders_3cyl	Cylinders_4cyl	Cylinders_5cyl	Cylinders_6cyl	Cylinders_8cyl	...
0	8.0	307.0	130.0	3504	12.0	0	0	0	0	1	...
2	18.0	318.0	150.0	3436	11.0	0	0	0	0	1	...
3	16.0	304.0	150.0	3433	12.0	0	0	0	0	1	...
4	17.0	302.0	140.0	3449	10.5	0	0	0	0	1	...
14	24.0	113.0	95.0	2372	15.0	0	1	0	0	0	...

5 rows × 40 columns

In [71]:
```
#Export data to excel to check the final values data after preparation
#master.to_excel('C:/Users/Sanjay Lohar/Desktop/final_cars.xlsx')
```

In [72]:
```
#create x and y
y=master['MPG']

x=master.drop('MPG',axis=1)
```

In [73]:
```
# import the library to split the training-test sample
from sklearn.model_selection import train_test_split
```

In [74]:
```
#split data into training and test samples
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.3, random_state=0)
```

In [75]:
```
#check sample size
print(x_train.shape,y_train.shape, x_test.shape,y_test.shape)
```

```
(243, 39) (243,) (105, 39) (105,)
```

In [ ]:

In [76]:
```
# standardize or feature scaling dataset

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)

x_test_scaled = scaler.transform(x_test)
```

In [77]:
```
x_train_scaled
```

Out[77]:
```
array([[-0.90721285, -0.95930583, -1.37083325, ..., -0.06428243,
         4.2062225 , -0.1118034 ],
       [-0.61615541, -0.31961376, -0.44908641, ..., 15.55634919,
        -0.23774301, -0.1118034 ],
       [ 0.6644973 , -0.24435587,  0.52182026, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       ...,
       [-1.1167742 , -1.10982161, -1.13186184, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       [-1.22155488,  0.01904675, -0.68123006, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       [-0.2203173 ,  0.43296515,  0.13810046, ..., -0.06428243,
        -0.23774301, -0.1118034 ]])
```

In [78]:
```
x_test_scaled
```

Out[78]:
```
array([[ 0.28030148,  0.01904675, -0.07492547, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       [-0.62779771,  0.58348093,  0.14219712, ..., -0.06428243,
        -0.23774301,  8.94427191],
       [-0.27852879, -0.24435587,  0.23778568, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       ...,
       [-0.89557055, -1.1850795 , -0.61295251, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       [-0.90721285, -0.69590321, -0.87513828, ..., -0.06428243,
        -0.23774301, -0.1118034 ],
       [-0.46480555, -0.47012954, -0.46274192, ..., -0.06428243,
        -0.23774301, -0.1118034 ]])
```

In [ ]:

In [ ]:

Linear regression

In [79]:
```
#import library for linear regression
from sklearn.linear_model import LinearRegression
```

In [80]:
```
#create a model object
model=LinearRegression()
```

In [81]:
```
# fit the model
model.fit(x_train_scaled,y_train)
```

Out[81]:
```
LinearRegression()
```

In [82]:
```
#Goodness of fit test: check the accuracy ot training model
model.score(x_train_scaled,y_train)
```

Out[82]:
```
0.7907794309128342
```

In [83]:
```
#check the accuracy ot testing model
model.score(x_test_scaled,y_test)
```

Out[83]:
```
0.7284320305884564
```

In [84]:
```
#predict y
linreg_y_pred=model.predict(x_test_scaled)
linreg_y_pred
```

Out[84]:
```
array([20.59482181, 24.51458588, 21.85695933, 14.71368373, 32.97108143,
       16.2695135 , 28.16051905, 28.02296528, 21.2756276 , 29.61801286,
       32.89676057, 20.05819741, 30.35195322, 30.43438349, 19.84076454,
       27.86335528, 25.22506659, 21.76532554, 17.43812594, 33.2450913 ,
       17.83448035, 32.7527569 , 28.69736617, 14.56504336, 28.77133613,
       18.68262228, 32.33532374, 24.43912589, 26.63494828, 15.32312412,
       16.13641519, 28.40683891, 24.59676879, 30.48105791, 31.98559539,
       15.14918655, 22.75303521, 10.0335068 , 29.76409721, 20.38157385,
       22.76284406, 33.82051338, 34.60824657, 16.43949419, 16.85383954,
       14.32467638, 19.57152681, 29.65801222, 24.71612975, 30.1997423 ,
       19.04266479, 25.02553609, 32.6801491 , 21.92664453, 12.72020693,
       34.94383707, 15.40892293, 27.53409899, 10.91711679, 30.44693484,
       22.18057384, 32.82165018, 26.06706799, 35.38212063, 26.41835695,
       11.99868274, 21.03288328, 19.70526021, 15.85046907, 14.51413261,
       28.08407171, 24.10124836, 29.65676154, 19.26187747, 28.22754849,
       22.21126496, 16.47124569, 29.51890595, 29.55033761, 15.56749768,
       37.31725482, 32.49092518, 20.34654859, 20.09117441, 25.85707952,
       21.18655843, 29.72224062, 31.2396371 , 11.26409308, 16.10207709,
       24.89322333, 17.17357816, 33.37108228, 30.45716121, 30.2045559 ,
       28.58549478, 33.59074954, 13.18892731, 30.62367615, 22.06432275,
       19.43242075, 22.91522651, 26.45469706, 25.60474274, 28.40230894])
```

In [ ]:

In [85]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# we cant calculate confusion_matrix, accuracy score, precision score, recall score , f1 score for Regression m
#like Linear Regression, Random forest Regressor, etc. so for regression models we can calculate r2_score, mean

#but we can calculate confusion_matrix, accuracy score, precision score, recall score , f1 score for Classifica
#like LogisicRegression, knn, svm, Decision tree, Random forest Classifier, naive-bayes, etc.

# confusion_matrix(ytest, linreg_y_pred)
```

In [86]:
```
#model=LinearRegression()
#model.fit(xtrain,ytrain)

#linreg_y_pred=model.predict(xtest)
```

```python
from sklearn.metrics import r2_score, mean_squared_error

linreg_r2_scr = r2_score(y_test, linreg_y_pred)
print("linreg_r2_scr: ", linreg_r2_scr )

mse= mean_squared_error(y_test, linreg_y_pred)
rmse=np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
linreg_r2_scr:  0.7284320305884564
Root Mean Squared Error: 4.04161886584362
```

In [ ]:

Ridge regression

In [87]:
```python
from sklearn.linear_model import Ridge

ridge=Ridge()
```

In [88]:
```python
# fit the model
ridge.fit(x_train_scaled,y_train)
```

Out[88]:
```
Ridge()
```

In [89]:
```python
#check the accuracy ot training model
ridge.score(x_train_scaled,y_train)
```

Out[89]:
```
0.7907059483561818
```

In [90]:
```python
#check the accuracy ot testing model
ridge.score(x_test_scaled,y_test)
```

Out[90]:
```
0.7279478999875066
```

In [91]:
```python
#predict y
ridgereg_y_pred=ridge.predict(x_test_scaled)
ridgereg_y_pred
```

Out[91]:
```
array([20.58718046, 24.57818279, 21.92998307, 14.69869048, 32.89956391,
       16.34630287, 28.2076056 , 28.13390811, 21.31815795, 29.59361007,
       32.74874564, 20.13241094, 30.41185972, 30.39519244, 19.86416875,
       27.84837419, 25.24190429, 21.76670582, 17.49995023, 33.232296  ,
       17.73296901, 32.71490573, 28.771418  , 14.50821144, 28.7647569 ,
       18.77478453, 32.32398413, 24.36412674, 26.74461407, 15.28619157,
       16.18755529, 28.47893483, 24.60792602, 30.46650093, 31.92952017,
       15.2390462 , 22.9245395 , 10.29191936, 29.82080828, 20.36242679,
       22.60631712, 33.7527488 , 34.50823995, 16.4017143 , 16.88105197,
       14.26714064, 19.54681924, 29.78633664, 24.72551053, 30.2067636 ,
       19.10442617, 25.03125519, 32.5199236 , 21.92471853, 12.77330552,
       34.92832247, 15.3917375 , 27.55155022, 10.92189842, 30.38553854,
       22.18431019, 32.65873546, 26.08770709, 35.34647374, 26.39091051,
       12.00169715, 20.96726215, 19.70657858, 15.96294248, 14.47222133,
       28.0853002 , 24.10810116, 29.57427606, 19.26926339, 28.22079103,
       22.18863388, 16.35023231, 29.48854661, 29.55230927, 15.61490863,
       37.19635998, 32.47511591, 20.30088005, 20.04650913, 25.77354846,
       21.17992062, 29.69102912, 31.13214847, 11.34125498, 16.11518656,
       24.85065707, 17.21551642, 33.344151  , 30.50524198, 30.13980274,
       28.63356704, 33.54394695, 13.17730746, 30.56332879, 21.99746453,
       19.42584375, 22.93915513, 26.43726378, 25.61149784, 28.40131974])
```

In [92]:
```python
from sklearn.metrics import r2_score, mean_squared_error

ridgereg_r2_scr = r2_score(y_test, ridgereg_y_pred)
print("ridgereg_r2_scr: ", ridgereg_r2_scr )

mse= mean_squared_error(y_test, ridgereg_y_pred)
rmse=np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
ridgereg_r2_scr:  0.7279478999875066
Root Mean Squared Error: 4.045219806153347
```

In [ ]:

Lasso regression

In [93]:
```python
from sklearn.linear_model import Lasso
```

In [94]:
```python
lasso = Lasso()
```

In [95]:
```python
# fit the model
```

```
lasso.fit(x_train_scaled,y_train)
```

Out[95]:
```
Lasso()
```

In [96]:
```
#check the accuracy ot training model
lasso.score(x_train_scaled,y_train)
```

Out[96]:
```
0.6771283863648423
```

In [97]:
```
#check the accuracy ot testing model
lasso.score(x_test_scaled,y_test)
```

Out[97]:
```
0.6748960442061311
```

In [98]:
```
#predict y
lassoreg_y_pred=lasso.predict(x_test_scaled)
lassoreg_y_pred
```

Out[98]:
```
array([22.47525382, 24.4236182 , 25.15703188, 14.71948041, 31.47212284,
       20.14601392, 28.90932643, 28.03018155, 22.43836536, 29.25615209,
       28.99149146, 21.07472063, 28.13511379, 29.47432924, 22.31836769,
       26.9543565 , 26.5662938 , 21.39538926, 19.340534  , 29.88508372,
       18.74084988, 30.09025899, 29.20784836, 14.33488296, 27.90737081,
       21.34363735, 29.03984144, 25.34073933, 28.58625117, 17.71901307,
       17.14166302, 29.13116834, 25.35916291, 28.10686157, 28.53852171,
       19.30400324, 26.34347368, 16.64545353, 28.02368914, 20.10665549,
       23.23016449, 30.11987632, 30.36728392, 16.64949305, 18.24507352,
       16.46379502, 19.57695947, 28.56211142, 24.19325201, 28.52696531,
       21.9199921 , 26.36273434, 28.71751076, 23.56080289, 15.05027701,
       31.24908615, 15.88439603, 26.23692563, 13.42770242, 28.14137257,
       21.03427937, 28.46201201, 27.18001641, 31.37085535, 24.98941184,
       15.68043262, 22.43909783, 21.54772977, 19.54959059, 16.8942433 ,
       28.37434986, 25.58707621, 29.14898352, 20.65663665, 26.92448847,
       22.2457539 , 17.02492863, 28.92941207, 28.28323952, 16.64545353,
       30.50123004, 29.99691233, 21.99181259, 21.25510892, 25.52339313,
       22.3339782 , 29.2626445 , 28.15556925, 15.01355887, 18.71239815,
       26.85369246, 19.35621747, 29.37082542, 29.8976646 , 27.3421685 ,
       28.4403447 , 30.24045073, 15.33845439, 29.18817162, 22.0904398 ,
       21.40144853, 25.88258309, 28.12916646, 28.15435739, 26.9768609 ])
```

In [99]:
```
from sklearn.metrics import r2_score, mean_squared_error

lassoreg_r2_scr = r2_score(y_test, lassoreg_y_pred)
print("lassoreg_r2_scr: ", lassoreg_r2_scr )

mse= mean_squared_error(y_test, lassoreg_y_pred)
rmse=np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
lassoreg_r2_scr:  0.6748960442061311
Root Mean Squared Error: 4.422086266693381
```

In [ ]:
```

```

Elastic Net regression

In [100…
```
from sklearn.linear_model import ElasticNet
```

In [101…
```
ElsNet = ElasticNet()
```

In [102…
```
# fit the model
ElsNet.fit(x_train_scaled,y_train)
```

Out[102]:
```
ElasticNet()
```

In [103…
```
#check the accuracy ot training model
ElsNet.score(x_train_scaled,y_train)
```

Out[103]:
```
0.6814251179097743
```

In [104…
```
#check the accuracy ot testing model
ElsNet.score(x_test_scaled,y_test)
```

Out[104]:
```
0.669389703736148
```

In [105…
```
#predict y
ElsNetreg_y_pred=ElsNet.predict(x_test_scaled)
ElsNetreg_y_pred
```

Out[105]:
```
array([21.92905031, 25.05170871, 24.96466669, 15.2276841 , 31.26970272,
       20.04606535, 28.47708309, 28.35791502, 23.26283829, 28.89085085,
       28.44650151, 21.04793429, 28.48874534, 29.63208379, 21.87011621,
       27.42670178, 25.97617886, 21.58690043, 19.97863542, 30.61327491,
       18.92724103, 30.56073468, 28.57668838, 15.10954814, 27.59988005,
       20.42926662, 29.39944153, 25.60465486, 28.88112367, 17.53562149,
       17.01467295, 29.1865128 , 26.07342086, 27.88860563, 28.45497948,
       19.28618999, 25.86821094, 16.30068039, 28.59890677, 20.50539941,
       24.6391605 , 29.73194819, 30.70560978, 16.93365472, 18.01340781,
       16.63734721, 20.2085407 , 28.10508071, 25.02400644, 28.39388864,
       21.21692739, 26.45782274, 28.29295391, 22.73200513, 14.9915485 ,
       31.60620366, 16.31006005, 25.80849798, 14.08220545, 27.7115412 ,
       22.57023351, 28.01339741, 26.44850049, 31.77823294, 25.64153123,
       16.0115737 , 22.18128936, 21.696431  , 19.22901537, 16.93702719,
       28.20584834, 25.32998895, 28.21015953, 20.65156664, 26.63262132,
       22.04909605, 17.45303507, 29.21253017, 27.73260511, 16.73658118,
       30.51656623, 30.41101537, 21.79424954, 20.86244758, 25.70701993,
       22.40274687, 28.90359178, 27.74425218, 14.96537985, 19.46454795,
       26.92121305, 18.27783485, 30.09529392, 30.02518229, 26.80170981,
       28.49692877, 30.54750213, 15.76938672, 29.35754736, 22.26626011,
       21.68531754, 26.2628076 , 27.63862121, 28.05397413, 26.59697967])
```

In [106…
```python
from sklearn.metrics import r2_score, mean_squared_error

ElsNetreg_r2_scr = r2_score(y_test, ElsNetreg_y_pred)
print("ElsNetreg_r2_scr: ", ElsNetreg_r2_scr )

mse= mean_squared_error(y_test, ElsNetreg_y_pred)
rmse=np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
ElsNetreg_r2_scr:  0.669389703736148
Root Mean Squared Error: 4.459377836347032
```

In [ ]:

Compairing accuracy scores of all models

In [107…
```python
print(linreg_r2_scr, ridgereg_r2_scr, lassoreg_r2_scr, ElsNetreg_r2_scr)
```

```
0.7284320305884564 0.7279478999875066 0.6748960442061311 0.669389703736148
```

In [108…
```python
accuracy_scores=pd.DataFrame( [[linreg_r2_scr, ridgereg_r2_scr, lassoreg_r2_scr, ElsNetreg_r2_scr]],
                                  columns=["linscr","ridgescr","lasscr","ElsNetscr"] )
accuracy_scores
```

Out[108]:

	linscr	ridgescr	lasscr	ElsNetscr
0	0.728432	0.727948	0.674896	0.66939

In [ ]:

In [ ]:

In [ ]: