In [1]:
```
#import pandas, numpy, matplotlib, seaborn
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]:
```
#import data
df=pd.read_csv(r"C:\Users\Sanjay Lohar\Downloads\automobile data.csv")
```

In [3]:
```
#print head, tail
df.head()
```

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

In [4]:
```
df.tail()
```

Out[4]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model_year	Origin	Car_Name
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

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 types
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]:
```
#Horsepower is a numeric variable that is stored as object. So we need to
#change the data type of horsepower to number
# 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')
```

In [9]:
```
df.dtypes
```

Out[9]:
```
MPG             float64
Cylinders         int64
Displacement    float64
Horsepower      float64
Weight            int64
Acceleration    float64
Model_year        int64
Origin            int64
Car_Name         object
dtype: object
```

In [10]:
```
# ----- Missing value imputation -----
```

In [11]:
```
#check if there are missing values
df.isnull().sum()
```

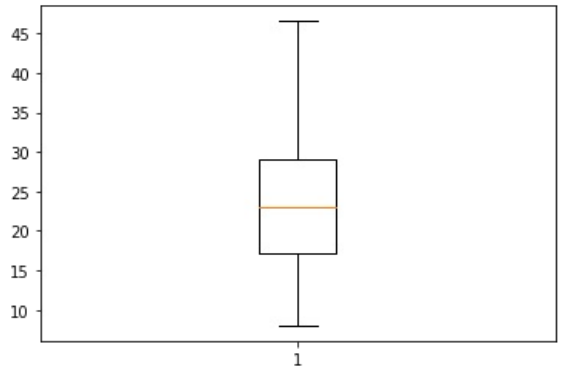
Out[11]:
```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model_year      0
Origin          0
Car_Name        0
dtype: int64
```

In [12]:
```
#impute missing values
df['Horsepower']=df['Horsepower'].fillna(df['Horsepower'].median())
```

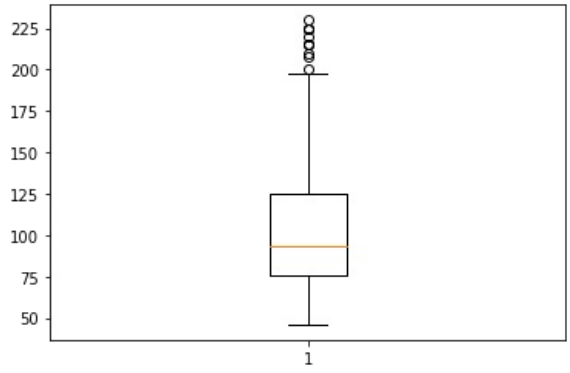
In [13]:
```
#check missing values again to ensure that there are no missing values in the data
df.isnull().sum()
```

Out[13]:
```
MPG             0
Cylinders       0
Displacement    0
Horsepower      0
Weight          0
Acceleration    0
Model_year      0
Origin          0
Car_Name        0
dtype: int64
```

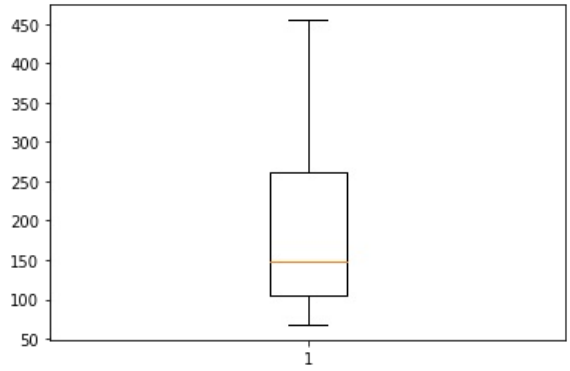
In [14]:
```
# create boxplots to check outliers
plt.boxplot(df['MPG']) #No outlier
plt.show()
```



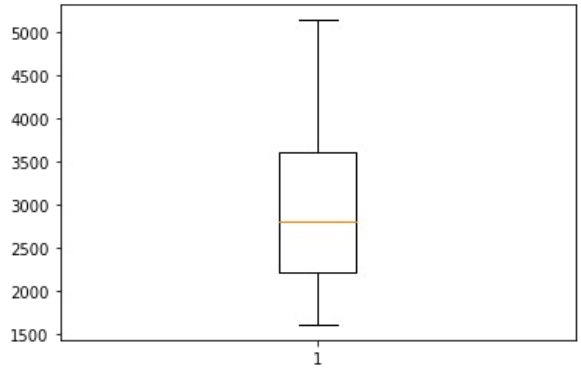
In [15]:
```
plt.boxplot(df['Horsepower']) #Has outlier
plt.show()
```



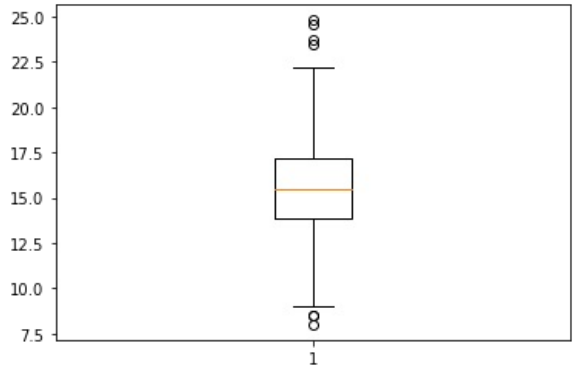
In [16]:
```
plt.boxplot(df['Displacement']) #No outlier
plt.show()
```



In [17]:
```
plt.boxplot(df['Weight']) #No outlier
plt.show()
```



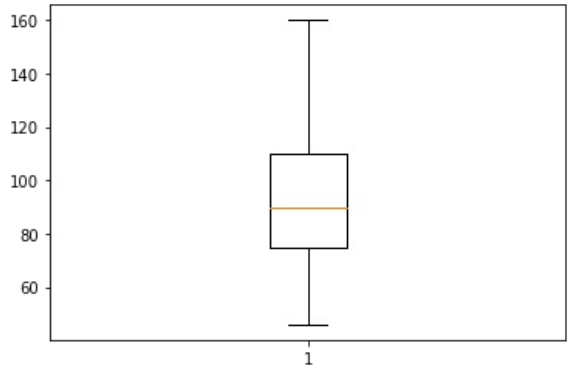
In [18]:
```
plt.boxplot(df['Acceleration']) #Has outlier
plt.show()
```



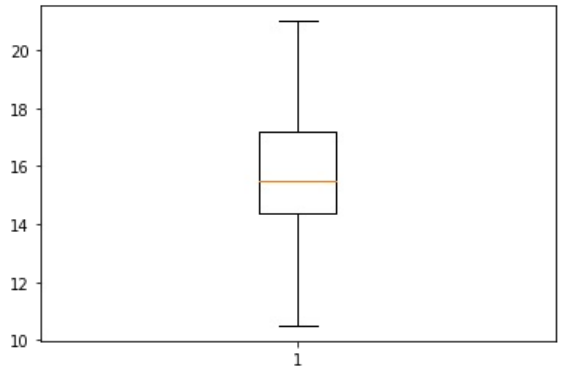
In [19]:
```
#Horsepower and Acceleration have outliers
```

In [20]:
```
#UDF to remove outliers
def remove_outlier(d,c):
    #find q1 and q3
    q1=d[c].quantile(0.25)
    q3=d[c].quantile(0.75)

    #iqr
    iqr=q3-q1

    #ub and lb
    ub=q3+1.5*iqr
    lb=q1-1.5*iqr

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

In [27]:
```
# Remove outlier Horsepower
df=remove_outlier(df,'Horsepower')
plt.boxplot(df['Horsepower'])
plt.show()
```



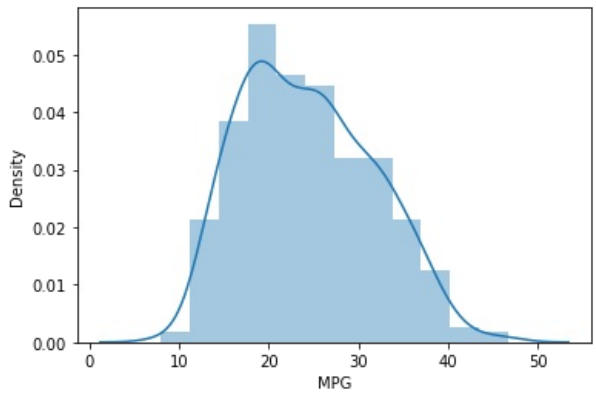
In [36]:
```
# Remove outlier Horsepower
df=remove_outlier(df,'Acceleration')
plt.boxplot(df['Acceleration'])
plt.show()
```



In [37]:
```
#------- 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 -----------------------
```

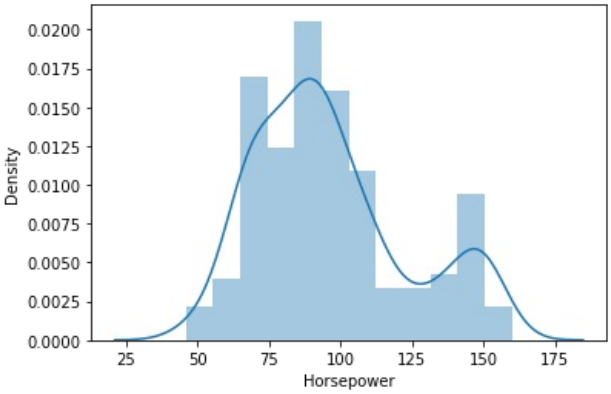
In [40]:
```
#distribution of MPG
sns.distplot(df['MPG'])
```

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



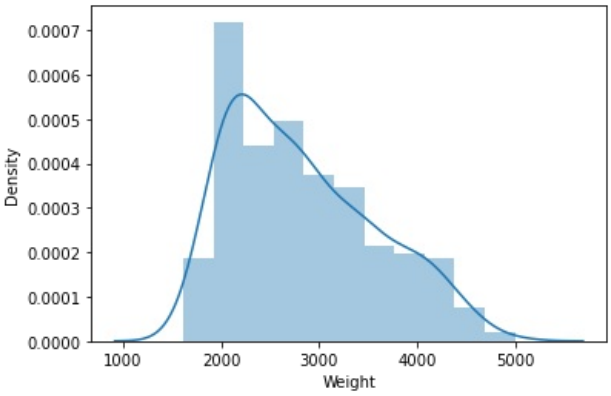
In [41]:
```
#distribution of Horsepower
sns.distplot(df['Horsepower'])
```

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



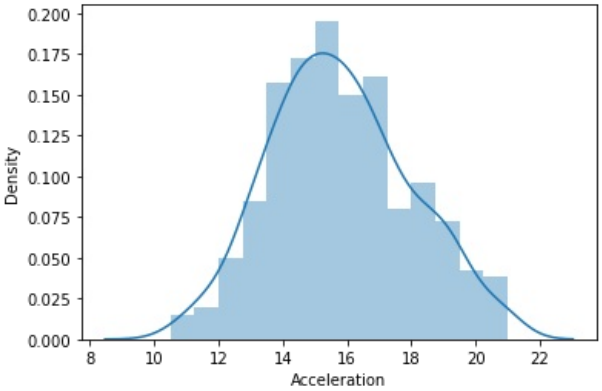
In [42]:
```
#distribution of Weight
sns.distplot(df['Weight'])
```

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



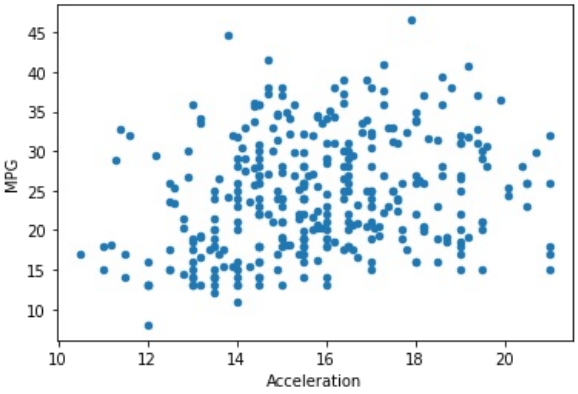
In [43]:
```
#distribution of Acceleration
sns.distplot(df['Acceleration'])
```

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



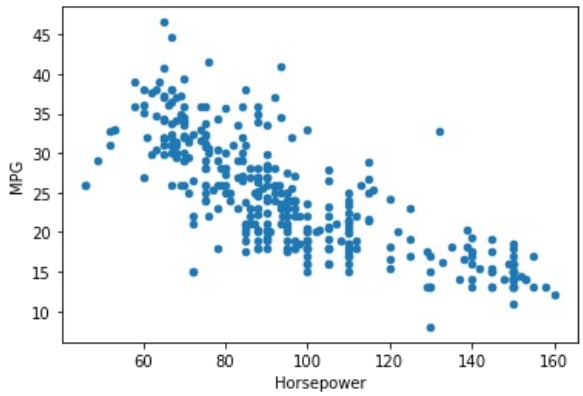
In [44]:
```
#Scatter plot to find the correlation betweeen MPG and Acceleration
df.plot(kind='scatter', x='Acceleration', y='MPG') # Medium to Weak +ve correlation
```

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



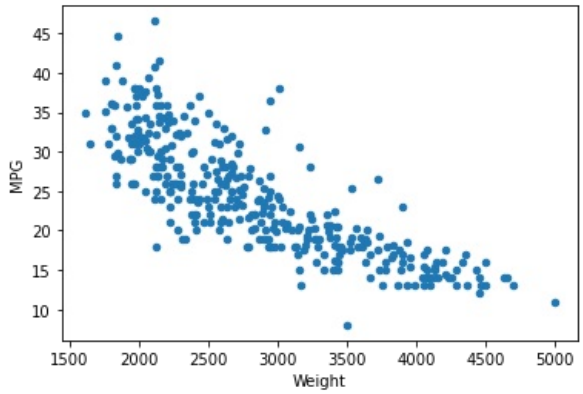
In [45]:
```
#Scatter plot to find the correlation betweeen MPG and Horsepower
df.plot(kind='scatter', x='Horsepower', y='MPG') #strong -ve correlation
```

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



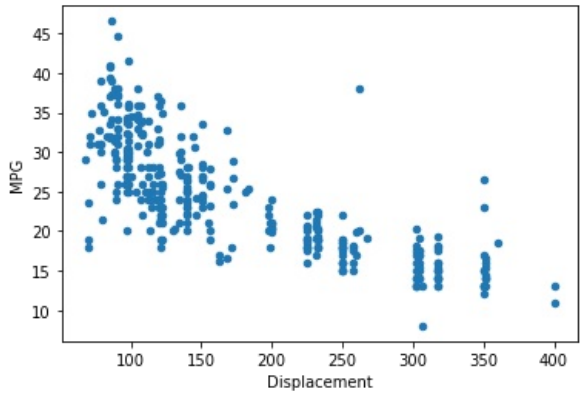
In [46]:
```
#Scatter plot to find the correlation betweeen MPG and Weight
df.plot(kind='scatter', x='Weight', y='MPG') #strong -ve correlation
```

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



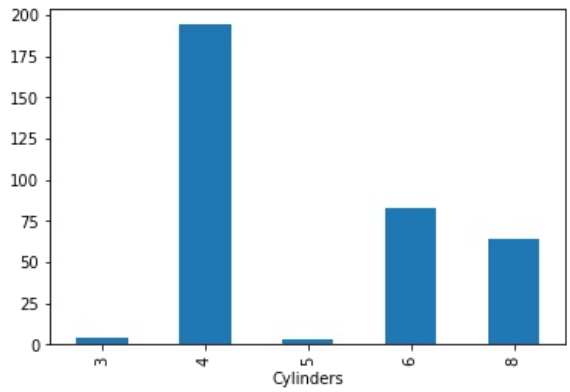
In [47]:
```
#Scatter plot to find the correlation betweeen MPG and Displacement
df.plot(kind='scatter', x='Displacement', y='MPG') #strong -ve correlation
```

Out[47]:
```
<AxesSubplot:xlabel='Displacement', ylabel='MPG'>
```



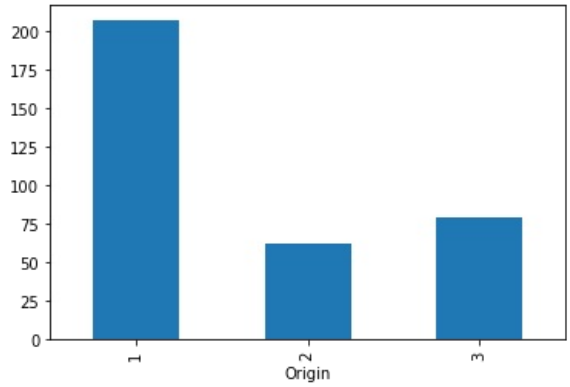
In [48]:
```
#No. of cars by cylinders
df.groupby('Cylinders')['Cylinders'].count().plot(kind='bar')
```

Out[48]:
```
<AxesSubplot:xlabel='Cylinders'>
```



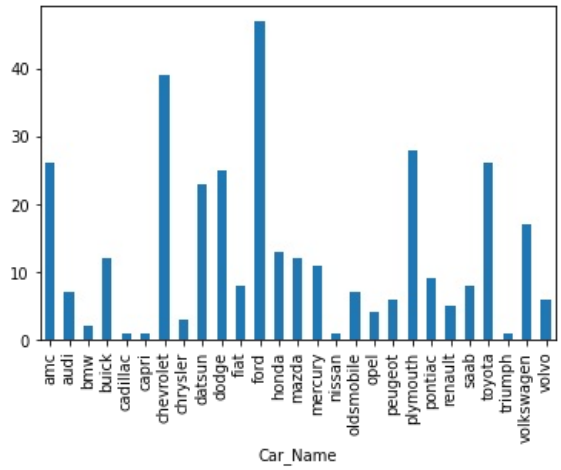
In [49]:
```
 #No. of cars by Origin
df.groupby('Origin')['Origin'].count().plot(kind='bar')
```

Out[49]:
```
<AxesSubplot:xlabel='Origin'>
```



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

Out[52]:
```
<AxesSubplot:xlabel='Car_Name'>
```



In [53]:
```
#------------------------- End of EDA -----------------
```

In [54]:
```
#Check unique values in Origin variable
df['Origin'].unique()
```

Out[54]:
```
array([1, 3, 2], dtype=int64)
```

In [55]:
```
#Replace 1 with US, 2 with Germany, 3 with Japan
df['Origin']=df['Origin'].replace([1,2,3],['US','Germany','Japan'])
```

In [56]:
```
#Print unique entries from origin column again
df['Origin'].unique()
```

Out[56]:
```
array(['US', 'Japan', 'Germany'], dtype=object)
```

In [57]:
```
df['Cylinders'].unique()
```

Out[57]:
```
array([8, 4, 6, 3, 5], dtype=int64)
```

In [58]:
```
#Cylinders is a categorical variable hence change Cylinders to Object
df['Cylinders']=df['Cylinders'].replace([8, 4, 6, 3, 5],
                                        ['8cyl','4cyl','6cyl','3cyl','5cyl'])
```

In [59]:
```
df['Cylinders'].unique()
```

Out[59]:
```
array(['8cyl', '4cyl', '6cyl', '3cyl', '5cyl'], dtype=object)
```

In [60]:
```
# Feature Selection: Print Correlation heatmap
```

In [61]:
```
#Check the correlation of numeric variables
df_numeric = df.select_dtypes(include=['float64', 'int64'])
df_numeric.head()
```

Out[61]:

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

In [62]:
```
#remove model_year column as it is a categorical feature
df_numeric=df_numeric.drop('Model_year', axis=1)
```

In [63]:
```
df_numeric.head()
```

Out[63]:

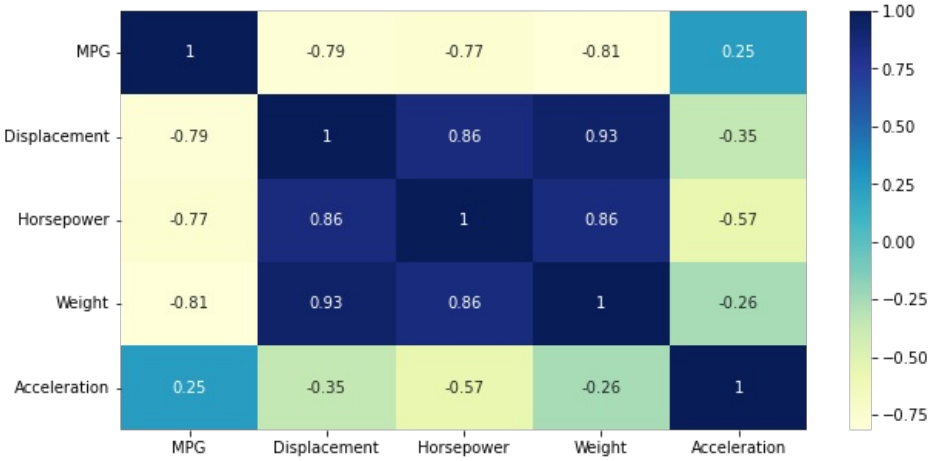
	MPG	Displacement	Horsepower	Weight	Acceleration
0	8.0	307.0	130.0	3504	12.0
2	18.0	318.0	150.0	3436	11.0
3	16.0	304.0	150.0	3433	12.0
4	17.0	302.0	140.0	3449	10.5
14	24.0	113.0	95.0	2372	15.0

In [64]:
```
# correlation matrix
cor_mat = df_numeric.corr()
cor_mat
```

Out[64]:

	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 [65]:
```
# print correlation heatmap

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

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



In [66]:
```
#Displacement, Horsepower, and Weight are the key features (significant variables)
#to predict the mileage of the cars
```

In [67]:
```
# One-hot encoding (dummy conversion)
df_categorical = df.select_dtypes(include=['object'])
df_categorical
```

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
...	...	...	...
392	4cyl	US	chevrolet
393	4cyl	US	ford
395	4cyl	US	dodge
396	4cyl	US	ford
397	4cyl	US	chevrolet

348 rows × 3 columns

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]:
```
#Create master data by combining numeric columns and dummies
master=pd.concat([df_numeric,df_dummies], axis=1)
master.head()
```

Out[69]:

	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 [70]:
```
#Export data to excel to check the values
#master.to_excel(r'C:\Users\Admin\Desktop\final_data.xlsx')
```

In [71]:
```
# Create X and Y
y=master['MPG']

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

In [72]:
```
# import library to split the training-test sample
from sklearn.model_selection import train_test_split
```

In [73]:
```
# Split the data into training and test sample [Random Sampling]
xtrain,xtest,ytrain,ytest=train_test_split(x,y, test_size=0.3, random_state=0)
```

In [74]:
```
#Print the sample size
print(xtrain.shape,ytrain.shape, xtest.shape,ytest.shape)
```

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

In [75]:
```
#import library for linear regression
from sklearn.linear_model import LinearRegression
```

In [76]:
```
#create a model object
model=LinearRegression()
```

In [77]:
```
# fit the model
```

```
model.fit(xtrain,ytrain)
```

Out[77]:
```
LinearRegression()
```

In [78]:
```
#Goodness of fit test: check the accuracy ot training model
model.score(xtrain,ytrain)
```

Out[78]:
```
0.7907794309128342
```

In [79]:
```
#predict y
ypred=model.predict(xtest)
```

In [80]:
```
#check prediction accuracy based on test sample
model.score(xtest,ytest)
```

Out[80]:
```
0.7284320305884562
```

In [ ]:

In [81]:
```
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 mo
#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 Classificat
#like LogisicRegression, knn, svm, Decision tree, Random forest Classifier, naive-bayes, etc.

# confusion_matrix(ytest, ypred)
```

In [82]:
```
model=LinearRegression()
model.fit(xtrain,ytrain)

ypred=model.predict(xtest)


from sklearn.metrics import r2_score, mean_squared_error

r2_scr = r2_score(ytest, ypred)
print("r2_score: ", r2_scr )

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

```
r2_score:  0.7284320305884562
Root Mean Squared Error: 4.0416188658436205
```

In [ ]:

In [ ]: