Electrical and Computer Engineering Department

Machine Learning and Data Science

ENCS5341

Assignment #1

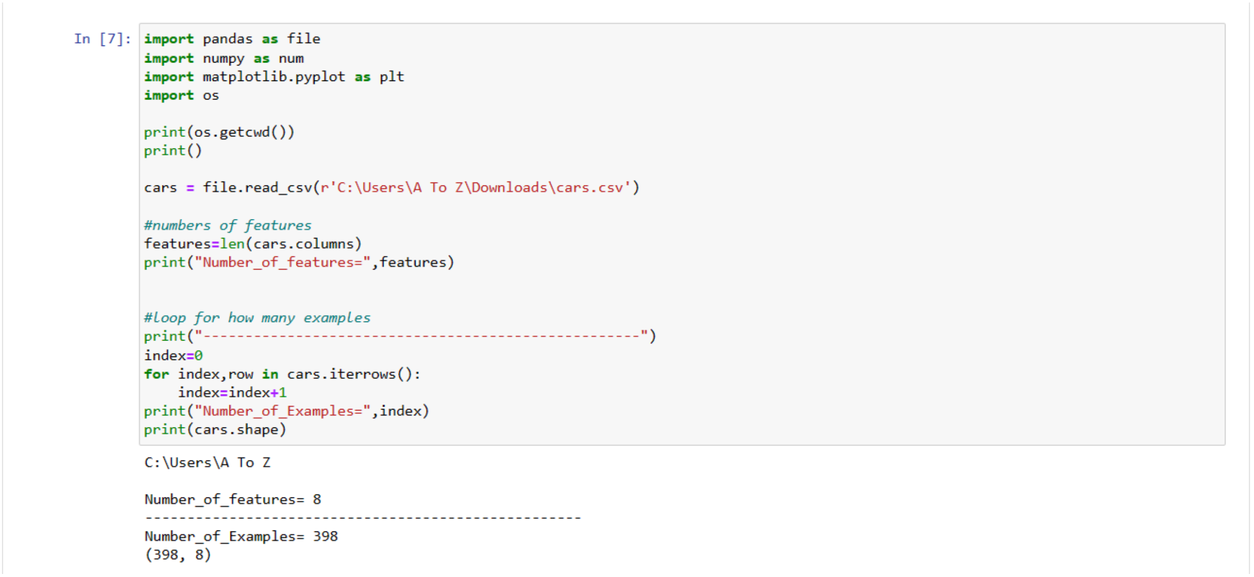
Dr: Yazan Abu Farha

Prepared by: Raneem Daqa -1202093

Sec:2

Date:29/11/202

1. Read the dataset and examine how many features and examples does it have?

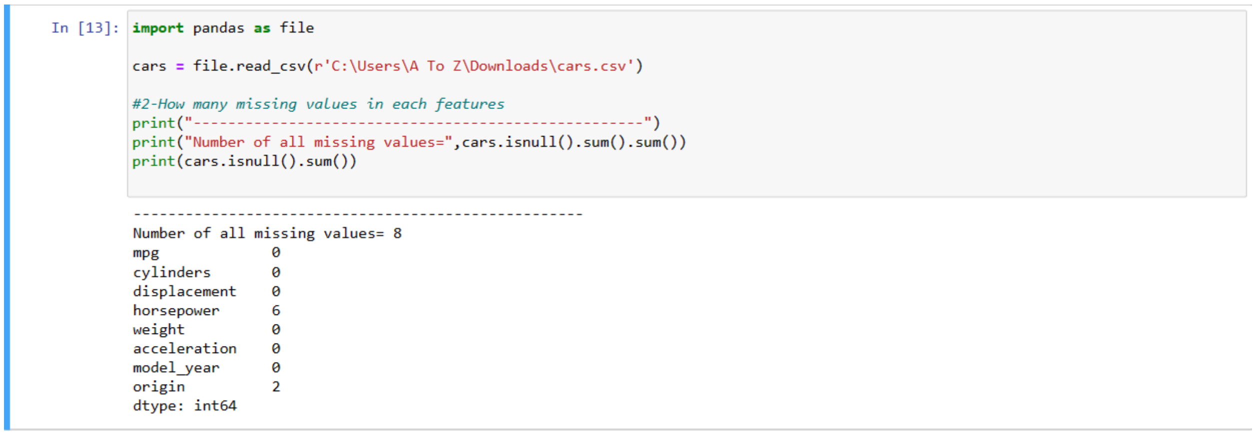


```
In [7]: import pandas as file
        import numpy as num
        import matplotlib.pyplot as plt
        import os

        print(os.getcwd())
        print()

        cars = file.read_csv(r'C:\Users\A To Z\Downloads\cars.csv')

        #numbers of features
        features=len(cars.columns)
        print("Number_of_features=",features)


        #loop for how many examples
        print("----------------------------------------------------")
        index=0
        for index,row in cars.iterrows():
            index=index+1
        print("Number_of_Examples=",index)
        print(cars.shape)
```

```
C:\Users\A To Z

Number_of_features= 8
----------------------------------------------------
Number_of_Examples= 398
(398, 8)
```

In this code, the dataset is read into a Pandas DataFrame named **cars**. The number of features (columns) in the dataset is then determined and printed, providing an overview of the many kinds of data that are accessible for every car. In addition, each row in the dataset is iterated through using a loop, which increments a counter to determine the total number of examples (rows). The last line **print(cars.shape)** shows the dataset's shape and its row and column counts.

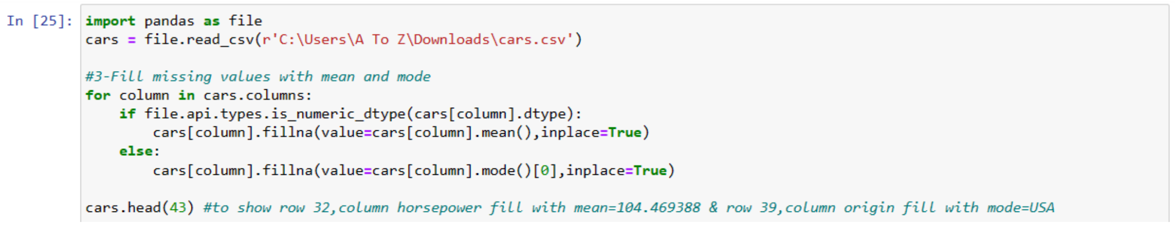
2. Are there features with missing values? How many missing values are there in each one?



```
In [13]: import pandas as file

         cars = file.read_csv(r'C:\Users\A To Z\Downloads\cars.csv')

         #2-How many missing values in each features
         print("----------------------------------------------------")
         print("Number of all missing values=",cars.isnull().sum().sum())
         print(cars.isnull().sum())

         ----------------------------------------------------
         Number of all missing values= 8
         mpg             0
         cylinders       0
         displacement    0
         horsepower      6
         weight          0
         acceleration    0
         model_year      0
         origin          2
         dtype: int64
```

This code to determine whether any values are missing. The total number of missing values for all characteristics is printed by the code. It performs this by using the DataFrame **isnull().sum().sum()** method, which essentially counts the number of instances of missing values. Using the **isnull().sum()** method, also provides a performs breakdown of the number of missing values for each characteristic.

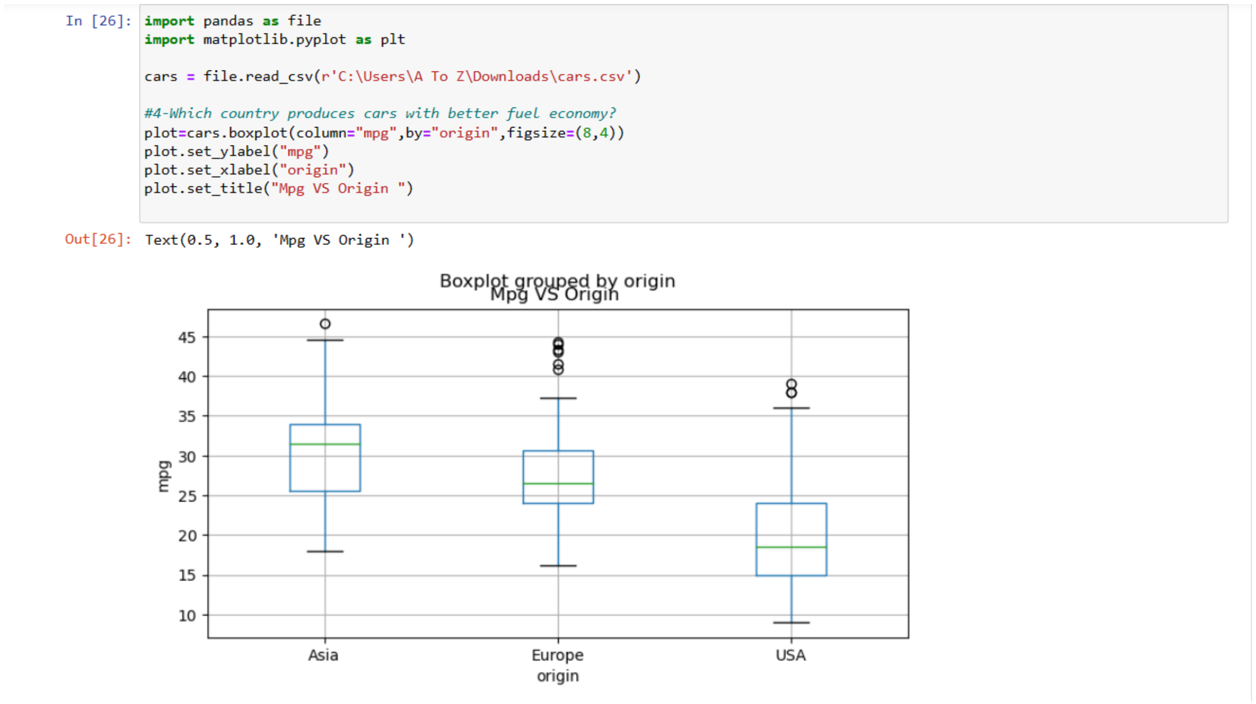
3. Fill the missing values in each feature using a proper imputation method (for example: fill with mean, median, or mode)



```
In [25]: import pandas as file
         cars = file.read_csv(r'C:\Users\A To Z\Downloads\cars.csv')

         #3-Fill missing values with mean and mode
         for column in cars.columns:
             if file.api.types.is_numeric_dtype(cars[column].dtype):
                 cars[column].fillna(value=cars[column].mean(),inplace=True)
             else:
                 cars[column].fillna(value=cars[column].mode()[0],inplace=True)

         cars.head(43) #to show row 32,column horsepower fill with mean=104.469388 & row 39,column origin fill with mode=USA
```

The missing values in the dataset are filled in with the proper replacements using this code. Each column in the dataset is iterated over by the algorithm to determine whether it includes numeric or non-numeric data. When it comes to numerical columns such as horsepower, the **fillna(value=cars[column].mean(), inplace=True)** method is used to fill in missing values with the mean of the corresponding column.The mode, or value that occurs the most frequently, is used to fill in missing values for non-numeric columns like origin by using the formula **fillna(value=cars[column].mode()[0], inplace=True)**.
To guarantee that the modifications are implemented to the original dataset, use the inplace=True argument. This procedure contributes to the dataset's increased completeness for ensuing analysis.

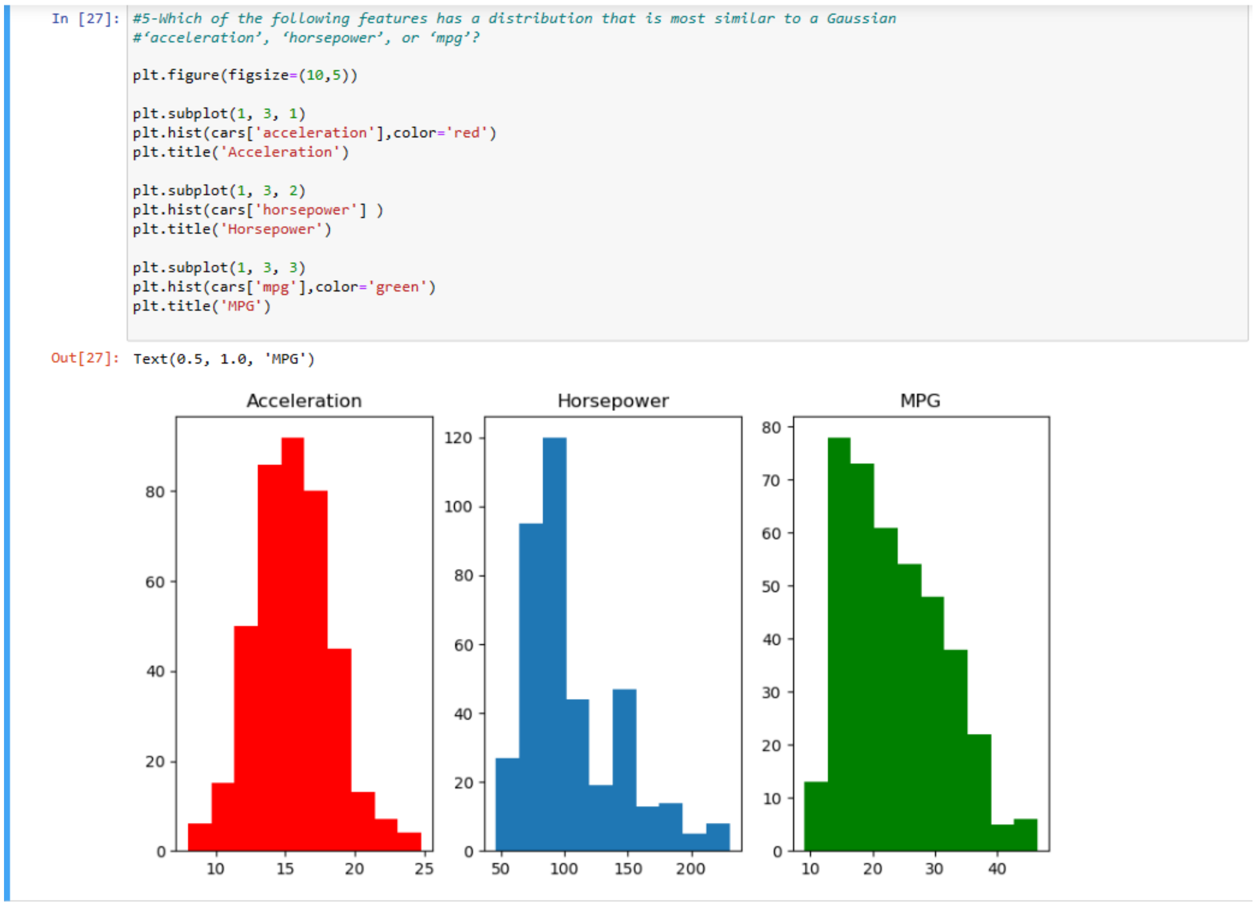
4. Which country produces cars with better fuel economy?



```
In [26]: import pandas as file
         import matplotlib.pyplot as plt

         cars = file.read_csv(r'C:\Users\A To Z\Downloads\cars.csv')

         #4-Which country produces cars with better fuel economy?
         plot=cars.boxplot(column="mpg",by="origin",figsize=(8,4))
         plot.set_ylabel("mpg")
         plot.set_xlabel("origin")
         plot.set_title("Mpg VS Origin ")

Out[26]: Text(0.5, 1.0, 'Mpg VS Origin ')
```

This code uses a boxplot to graphically compare the fuel efficiency of automobiles made in various nations. The (mpg) column is the specific focus of the code. Matplotlib is used to construct the boxplot, which shows the fuel efficiency distribution for each country of origin, indicated by the (origin) column. The (mpg) values are shown by the y-axis, which lets us see the distribution and central tendency of fuel efficiency for every nation. Through a visual comparison of these boxplots, it is possible to rapidly determine which nation produces cars with higher fuel efficiency and see any possible differences in fuel efficiency between different sources.

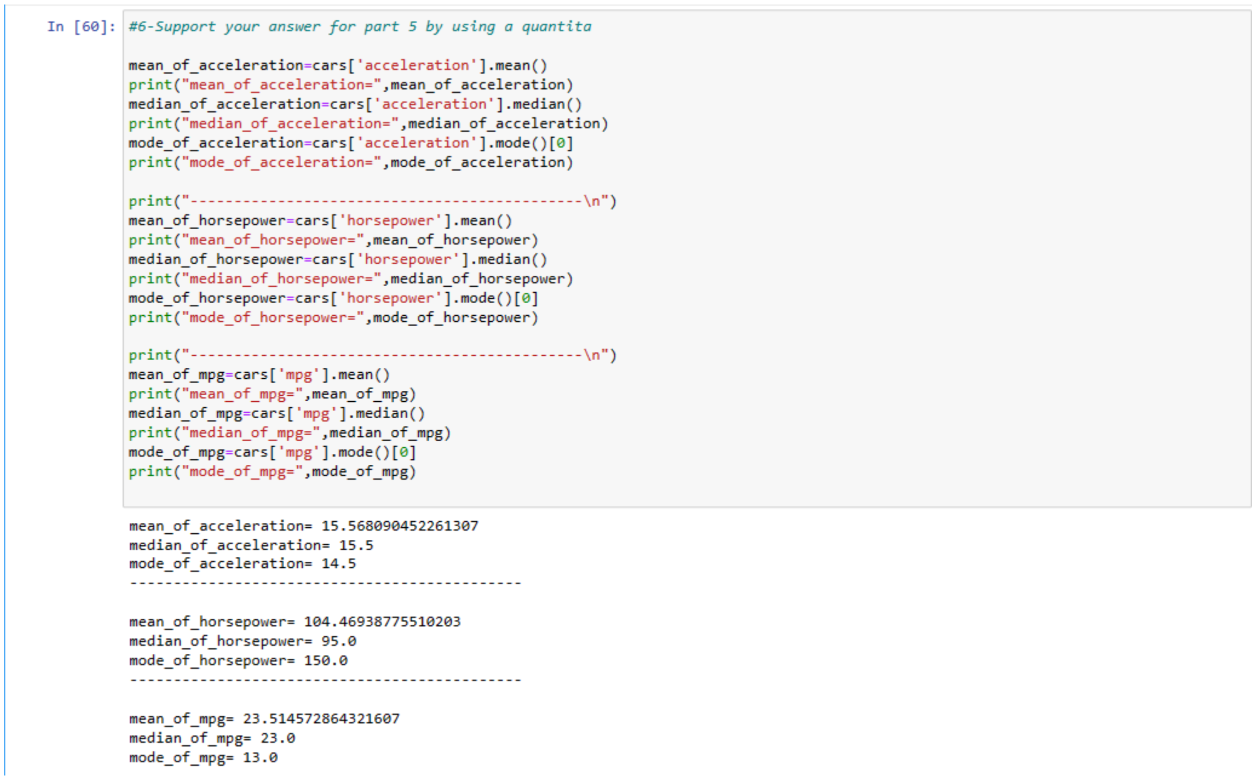
5. Which of the following features has a distribution that is most similar to a Gaussian: 'acceleration', 'horsepower', or 'mpg'?



```
In [27]: #5-Which of the following features has a distribution that is most similar to a Gaussian
         #'acceleration', 'horsepower', or 'mpg'?

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

         plt.subplot(1, 3, 1)
         plt.hist(cars['acceleration'],color='red')
         plt.title('Acceleration')

         plt.subplot(1, 3, 2)
         plt.hist(cars['horsepower'] )
         plt.title('Horsepower')

         plt.subplot(1, 3, 3)
         plt.hist(cars['mpg'],color='green')
         plt.title('MPG')

Out[27]: Text(0.5, 1.0, 'MPG')
```

The code is to determine which of the three feature distributions (acceleration),(horsepower) and (mpg) most closely match a Gaussian (normal) distribution. Histograms offer a graphic depiction of the data distribution. The 'acceleration' subplot is displayed in red, (horsepower) in the center, and (mpg) in green. These histograms' forms can provide information about the distribution properties of each item. The feature with the most Gaussian-like distribution is acceleration the histogram form is closest to a bell curve, which is characteristic of a Gaussian distribution.

6. Support your answer for part 5 by using a quantitative measure.



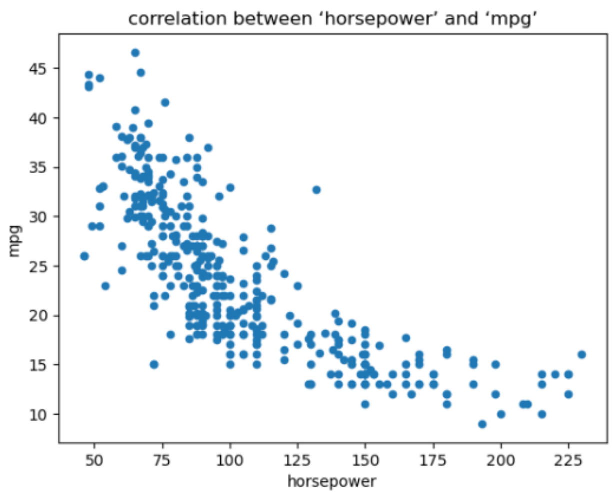
```
In [60]: #6-Support your answer for part 5 by using a quantita

         mean_of_acceleration=cars['acceleration'].mean()
         print("mean_of_acceleration=",mean_of_acceleration)
         median_of_acceleration=cars['acceleration'].median()
         print("median_of_acceleration=",median_of_acceleration)
         mode_of_acceleration=cars['acceleration'].mode()[0]
         print("mode_of_acceleration=",mode_of_acceleration)

         print("---------------------------------------------\n")
         mean_of_horsepower=cars['horsepower'].mean()
         print("mean_of_horsepower=",mean_of_horsepower)
         median_of_horsepower=cars['horsepower'].median()
         print("median_of_horsepower=",median_of_horsepower)
         mode_of_horsepower=cars['horsepower'].mode()[0]
         print("mode_of_horsepower=",mode_of_horsepower)

         print("---------------------------------------------\n")
         mean_of_mpg=cars['mpg'].mean()
         print("mean_of_mpg=",mean_of_mpg)
         median_of_mpg=cars['mpg'].median()
         print("median_of_mpg=",median_of_mpg)
         mode_of_mpg=cars['mpg'].mode()[0]
         print("mode_of_mpg=",mode_of_mpg)
```

```
mean_of_acceleration= 15.568090452261307
median_of_acceleration= 15.5
mode_of_acceleration= 14.5
---------------------------------------------

mean_of_horsepower= 104.46938775510203
median_of_horsepower= 95.0
mode_of_horsepower= 150.0
---------------------------------------------

mean_of_mpg= 23.514572864321607
median_of_mpg= 23.0
mode_of_mpg= 13.0
```

This code computes mean, median, and mode for the characteristics distributions (acceleration), (horsepower) and (mpg) in the dataset .The code offers a qualitative conclusion that the distribution of (acceleration) is Gaussian-like. This conclusion is based on quantitative measurements, where a symmetric distribution is suggested by the mean, median, and mode values for (acceleration) which are probably near in magnitude. As so, the code validates the histogram analysis and provides quantitative evidence that, among the attributes analyzed, 'acceleration' has a distribution that is most akin to a Gaussian.

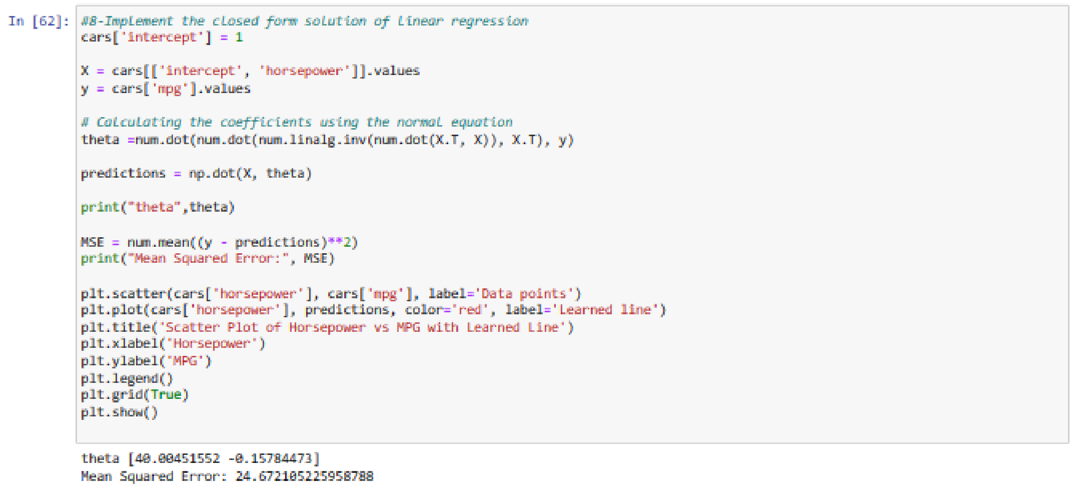
7. Plot a scatter plot that shows the ‘horsepower’ on the x-axis and ‘mpg’ on the y-axis. Is there a correlation between them? Positive or negative?

```
In [9]: #scatter plot that shows the correlation between ‘horsepower’ and ‘mpg’ oPositive or negative
        cars.plot.scatter(x='horsepower',y='mpg')
        print("correlation_coefficient=",cars['horsepower'].corr(cars['mpg']))
        plt.title("correlation between ‘horsepower’ and ‘mpg’")

        correlation_coefficient= -0.7784267838977759

Out[9]: Text(0.5, 1.0, 'correlation between ‘horsepower’ and ‘mpg’')
```



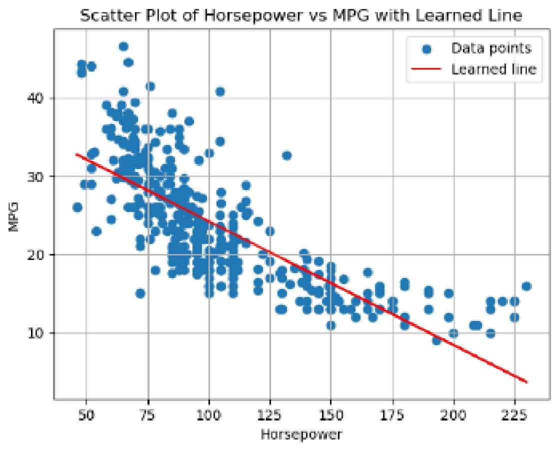
with this code, the association between the two characteristics in the data set(horsepower) and (mpg) is visually represented as a scatter plot. One can determine the kind and degree of the relationship between these two factors by looking at the overall pattern of scores. A negative association is indicated by a descending slope, which means that more (horsepower) is correlated with higher (mpg). That means there is a negative link between them.

8. Implement the closed form solution of linear regression and use it to learn a linear model to predict the ‘mpg’ from the ‘horsepower’. Plot the learned line on the same scatter plot you got in part 7



```
In [62]: #8-Implement the closed form solution of linear regression
cars['intercept'] = 1

X = cars[['intercept', 'horsepower']].values
y = cars['mpg'].values

# Calculating the coefficients using the normal equation
theta =num.dot(num.dot(num.linalg.inv(num.dot(X.T, X)), X.T), y)

predictions = np.dot(X, theta)

print("theta",theta)

MSE = num.mean((y - predictions)**2)
print("Mean Squared Error:", MSE)

plt.scatter(cars['horsepower'], cars['mpg'], label='Data points')
plt.plot(cars['horsepower'], predictions, color='red', label='Learned line')
plt.title('Scatter Plot of Horsepower vs MPG with Learned Line')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.legend()
plt.grid(True)
plt.show()
```

```
theta [40.00451552 -0.15784473]
Mean Squared Error: 24.672105225958788
```

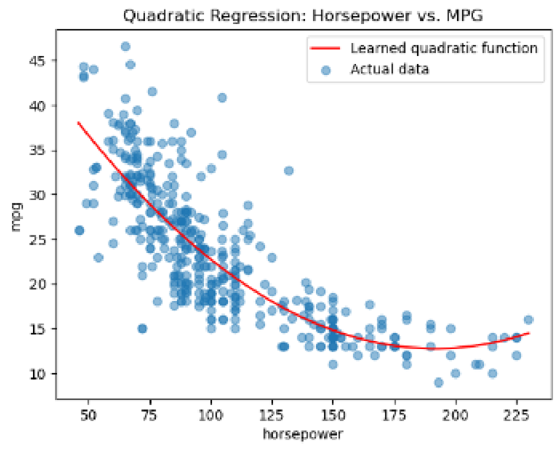


This code produces a visual depiction of a linear regression model that uses the (horsepower) to estimate the (mpg).The code creates matrices X and Y that represent the features and target variable, respectively, and adds a column to the dataset for the intercept term. The optimal coefficients (theta) for the linear equation are then determined by applying the closed form solution. The red line, which follows the trend of the data points, represents the best-fit line that minimizes the difference between the expected and actual (mpg) values based on horsepower. This suggests that the linear regression model fits the observed relationship between (horsepower) and (mpg) well.

9. Repeat part 8 but now learn a quadratic function of the form $f = w_0 + w_1 x + w_2 x^2$

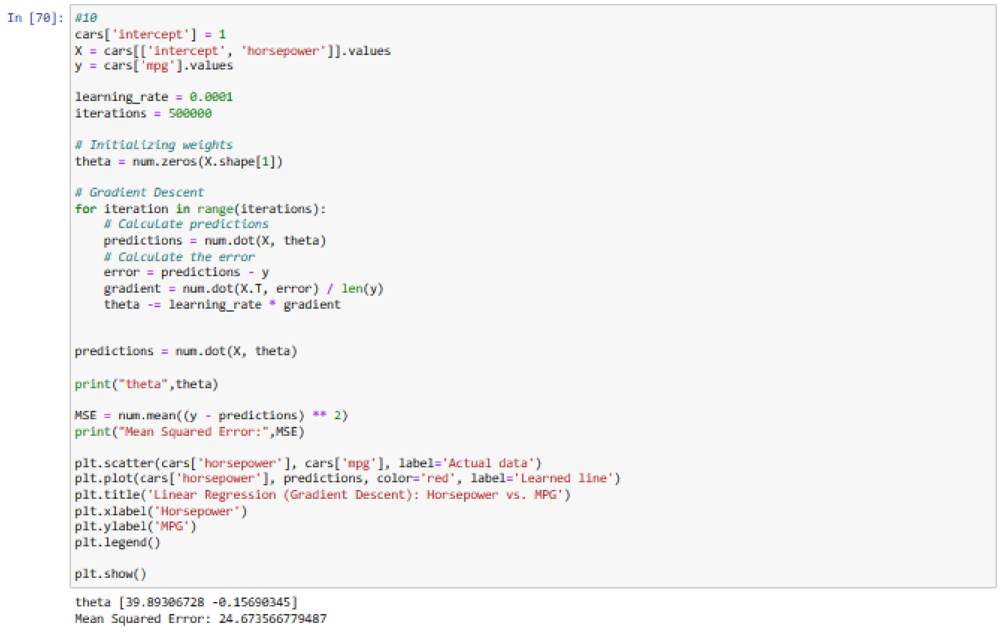
```
In [64]: #9- quadratic function
cars['intercept'] = 1
cars['horsepower_squared'] = cars['horsepower'] ** 2

X = cars[['intercept', 'horsepower', 'horsepower_squared']].values
y = cars['mpg'].values

theta = num.dot(num.dot(num.linalg.inv(num.dot(X.T, X)), X.T), y)

predictions = num.dot(X, theta)

sorted_indices = num.argsort(cars['horsepower'])
sorted_horsepower = cars['horsepower'].iloc[sorted_indices]
sorted_predictions = predictions[sorted_indices]

print("theta",theta)

MSE = np.mean((y - predictions) ** 2)
print("Mean Squared Error:",MSE)

plt.plot(sorted_horsepower, sorted_predictions, color='red', label='Learned quadratic function')
plt.scatter(cars['horsepower'], cars['mpg'], label='Actual data', alpha=0.5)
plt.title('Quadratic Regression: Horsepower vs. MPG')
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.legend()

plt.show()
```

```
theta [ 5.64035222e+01 -4.55434972e-01  1.18761665e-03]
Mean Squared Error: 20.07699602485639
```



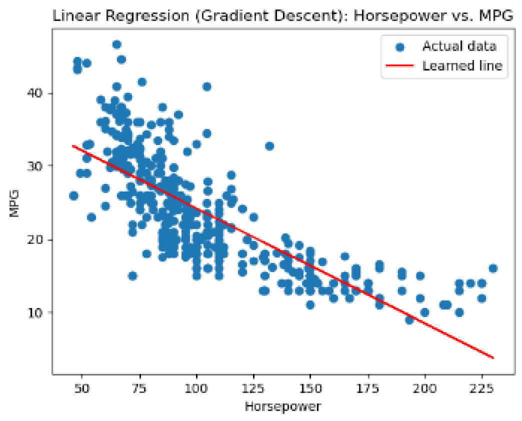
The output of this code is a quadratic regression model visualization, which forecasts (mpg) by using the (horsepower). Firstly, the code expands the dataset by adding a column for the intercept term and a squared term of the (horsepower) feature. The best coefficients (Theta) for the quadratic equation are then determined using the closed form solution. The quadratic function's shape is represented by the red curve, which demonstrates how(mpg)varies curvedly with (horsepower) The curve's good alignment with the data point distribution suggests that the quadratic model offers a good fit match to the noted correlation between (mpg) and (horsepower).

10. Repeat part 8 (simple linear regression case) but now by implementing the gradient descent algorithm instead of the closed form solution.



```
In [70]: #10
cars['intercept'] = 1
X = cars[['intercept', 'horsepower']].values
y = cars['mpg'].values

learning_rate = 0.0001
iterations = 500000

# Initializing weights
theta = num.zeros(X.shape[1])

# Gradient Descent
for iteration in range(iterations):
    # Calculate predictions
    predictions = num.dot(X, theta)
    # Calculate the error
    error = predictions - y
    gradient = num.dot(X.T, error) / len(y)
    theta -= learning_rate * gradient


predictions = num.dot(X, theta)

print("theta",theta)

MSE = num.mean((y - predictions) ** 2)
print("Mean Squared Error:",MSE)

plt.scatter(cars['horsepower'], cars['mpg'], label='Actual data')
plt.plot(cars['horsepower'], predictions, color='red', label='Learned line')
plt.title('Linear Regression (Gradient Descent): Horsepower vs. MPG')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.legend()

plt.show()
```

```
theta [39.89306728 -0.15690345]
Mean Squared Error: 24.673566779487
```



This code produces a figure that shows how to predict mpg based on the (horsepower) feature using a linear regression model that was trained using the gradient descent approach. To reduce the discrepancy between the predicted and actual (mpg) values in the dataset, the program iteratively changed the linear model's weights, or coefficients. How well the learned line fits the distribution of data points is clearly displayed in the plot. A satisfactory fit is indicated by a red line that closely matches the trend of the data points, indicating that the linear regression model accurately depicts the relationship between (mpg) and (horsepower).

I Calculated Mean Squared Error (MSE) for Linear Regression, quadratic regression and Linear Regression (Gradient Descent), When it reached the same value of MSE for part8 and part10, I adopted the number of iterations in part10