Faculty of Engineering & Technology
Electrical & Computer Engineering Department

Machine Learning - ENCS5341
Assignment One Report

Prepared by :
Basheer Arouri - 1201141

Instructor : Dr. Yazan Abu Farha
Section : 2

Part 1:

Code:

```python
import pandas as pd
contents = pd.read_csv("cars.csv")
print(contents.to_string())
```

```python
#This is just to print the number of features and examples in the dataset
print(contents)
```

Output:

```
"C:\Users\bashe\Desktop\My University\Forth Year-Second Semester\AI Lab\pythonProject6\Scripts\pytho
     mpg  cylinders  displacement  horsepower  weight  acceleration  model_year  origin
0   18.0          8         307.0       130.0    3504          12.0          70     USA
1   15.0          8         350.0       165.0    3693          11.5          70     USA
2   18.0          8         318.0       150.0    3436          11.0          70     USA
3   16.0          8         304.0       150.0    3433          12.0          70     USA
4   17.0          8         302.0       140.0    3449          10.5          70     USA
5   15.0          8         429.0       198.0    4341          10.0          70     USA
6   14.0          8         454.0       220.0    4354           9.0          70     USA
7   14.0          8         440.0       215.0    4312           8.5          70     USA
8   14.0          8         455.0       225.0    4425          10.0          70     USA
9   15.0          8         390.0       190.0    3850           8.5          70     USA
10  15.0          8         383.0       170.0    3563          10.0          70     USA
11  14.0          8         340.0       160.0    3609           8.0          70     USA
12  15.0          8         400.0       150.0    3761           9.5          70     USA
13  14.0          8         455.0       225.0    3086          10.0          70     USA
14  24.0          4         113.0        95.0    2372          15.0          70    Asia
15  22.0          6         198.0        95.0    2833          15.5          70     USA
16  18.0          6         199.0        97.0    2774          15.5          70     USA
17  21.0          6         200.0        85.0    2587          16.0          70     USA
18  27.0          4          97.0        88.0    2130          14.5          70    Asia
19  26.0          4          97.0        46.0    1835          20.5          70  Europe
20  25.0          4         110.0        87.0    2672          17.5          70  Europe
21  24.0          4         107.0        90.0    2430          14.5          70  Europe
22  25.0          4         104.0        95.0    2375          17.5          70  Europe
23  26.0          4         121.0       113.0    2234          12.5          70  Europe
24  21.0          6         199.0        90.0    2648          15.0          70     USA
25  10.0          8         360.0       215.0    4615          14.0          70     USA
26  10.0          8         307.0       200.0    4376          15.0          70     USA
27  11.0          8         318.0       210.0    4382          13.5          70     USA
```

and so on for the entire data.

```
ML_1
"C:\Users\bashe\Desktop\My University\Forth Year-Second Semester\AI Lab\pythonProject6\Scripts\python.exe" C:\Users\bashe\pythonProject6\ML_1.py
      mpg  cylinders  displacement  ...  acceleration  model_year  origin
0    18.0          8         307.0  ...          12.0          70     USA
1    15.0          8         350.0  ...          11.5          70     USA
2    18.0          8         318.0  ...          11.0          70     USA
3    16.0          8         304.0  ...          12.0          70     USA
4    17.0          8         302.0  ...          10.5          70     USA
..    ...        ...           ...  ...           ...         ...     ...
393  27.0          4         140.0  ...          15.6          82     USA
394  44.0          4          97.0  ...          24.6          82  Europe
395  32.0          4         135.0  ...          11.6          82     USA
396  28.0          4         120.0  ...          18.6          82     USA
397  31.0          4         119.0  ...          19.4          82     USA

[398 rows x 8 columns]

Process finished with exit code 0
```

Explanation:

I just got the data from the file 'cars.csv' and converted it to a dataframe using the pandas package in python. In the first picture, I printed the entire dataframe which contains the features in the next picture. In the second one, we can notice that we have 398 samples (examples) to learn the model, and we have 7 features (8 = 7 features + the target values (mpg)).

Part 2:

Code:

```
ML_2_3.py   ML_2.py   cars.csv

# ----------------------------This is the second part-----------------------------
dataframe = pd.read_csv("cars.csv")
for feature in dataframe:
    # Series type for missing values
    missing_values = pd.isnull(dataframe[feature])
    # I want to convert it string type
    number_of_missing_values = missing_values.to_string().count('True')
    print("For Feature {}".format(feature) + ", number of missing values are {}".format(number_of_missing_values))
```

Output:

```
ML_2
"C:\Users\bashe\Desktop\My University\Forth Year-Second Semester
For Feature mpg, number of missing values are 0
For Feature cylinders, number of missing values are 0
For Feature displacement, number of missing values are 0
For Feature horsepower, number of missing values are 6
For Feature weight, number of missing values are 0
For Feature acceleration, number of missing values are 0
For Feature model_year, number of missing values are 0
For Feature origin, number of missing values are 2

Process finished with exit code 0
```

Explanation:

After I read the file and converted it to a dataframe, I iterated for each feature to check if it has missing values by the assist 'isnull' method in pandas. After that, I checked the number of missing values by the 'count' method, if it is null, the answer will be true. We can notice that we have 6 missing values in horsepower and 2 in origin features.

NOTE: I know that the 'mpg' is not a feature, but I also checked it worrying if it has missing values.

Part 3:

Code:

```python
import pandas as pd

# ------------------------------This is the third part------------------------------
dataframe = pd.read_csv("cars.csv")
# I calculated the mode for the columns and replaced each missing value with this mode value
modes_dataframe = dataframe.mode()  # returns a dataframe contains mode for each column
for mode_for_feature in modes_dataframe:
    current_mode = modes_dataframe[mode_for_feature][0]  # Getting the mode for each column for this dataframe
    # fill the missing value with mode for this feature
    dataframe[mode_for_feature] = dataframe[mode_for_feature].fillna(current_mode)

# This is just to confirm that the missing data has been completed#
for feature in dataframe:
    # Series type for missing values
    missing_values = pd.isnull(dataframe[feature])
    # I want to convert it string type
    number_of_missing_values = missing_values.to_string().count('True')
    print("For Feature {}".format(feature) + ", number of missing values are {}".format(number_of_missing_values))

```

Output:

```
"C:\Users\bashe\Desktop\My University\Forth Year-Second Semester\AI Lab\pythonProject6\Scripts\python.exe" C:\Users\bashe\pythonProject6\ML_3.py
For Feature mpg, number of missing values are 0
For Feature cylinders, number of missing values are 0
For Feature displacement, number of missing values are 0
For Feature horsepower, number of missing values are 0
For Feature weight, number of missing values are 0
For Feature acceleration, number of missing values are 0
For Feature model_year, number of missing values are 0
For Feature origin, number of missing values are 0

Process finished with exit code 0
```

Explanation:

After I read the file and converted it to a dataframe, I took the mode for the entire dataframe which returns a new dataframe containing the origin features, each corresponding with its mode value. I iterated for each feature ,got its mode and filled all the missing values for this feature by this mode. We noticed that the 'horsepower' and 'origin' features had missing values and I replaced them by the mode.

NOTE: I know that the 'mpg' is not a feature, but i also checked it worrying if it has missing values.

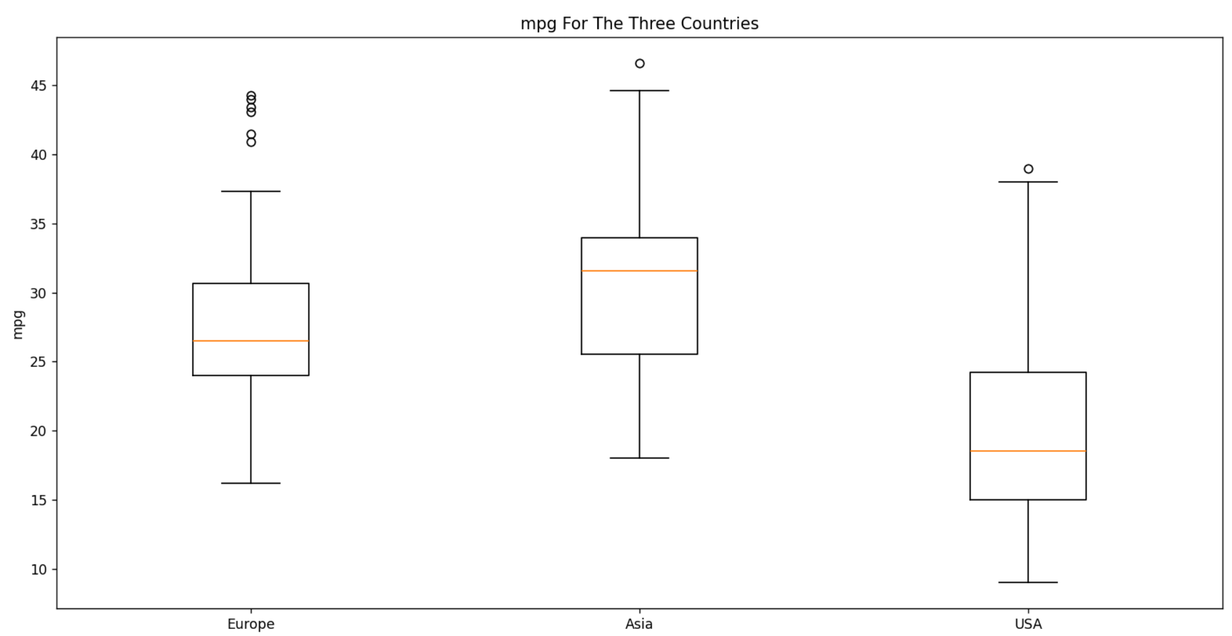
Part 4:

Code:

ML_3_UP_TO_10.py ML_3.py

```python
# ------------------------------This is the forth part------------------------------
# First, make the Europe dataframe that contains just the europe origin with corresponding mpgs
condition = dataframe['origin'] == 'Europe'
europe_dataframe = dataframe[condition]

# Second, make the Asia dataframe that contains just the Asia origin with corresponding mpgs
condition = dataframe['origin'] == 'Asia'
Asia_dataframe = dataframe[condition]

# Third, make the USA dataframe that contains just the USA origin with corresponding mpgs
condition = dataframe['origin'] == 'USA'
USA_dataframe = dataframe[condition]

fig, ax = plt.subplots()
# Plot box plot for each DataFrame on the same graph
ax.boxplot([europe_dataframe['mpg'], Asia_dataframe['mpg'], USA_dataframe['mpg']], labels=['Europe', 'Asia', 'USA'])
plt.ylabel('mpg')
# Setting the title
plt.title('mpg For The Three Countries')
# Show the plot
plt.show()
```

Output:



Explanation:

I first created a condition for each country to get three dataframes: europe, Asia and USA which contain europe, Asia and USA origin features respectively. After that I plotted the BoxPlot for each country with mpg as an output. We can notice that Asia produces cars with better fuel economy than the other countries. And this is for two reasons :
1- The median(which represents 50%less and 50% higher than the values) is higher than the medians for the USA and europe.
2-The noisy data (outlier mpg) is just one.

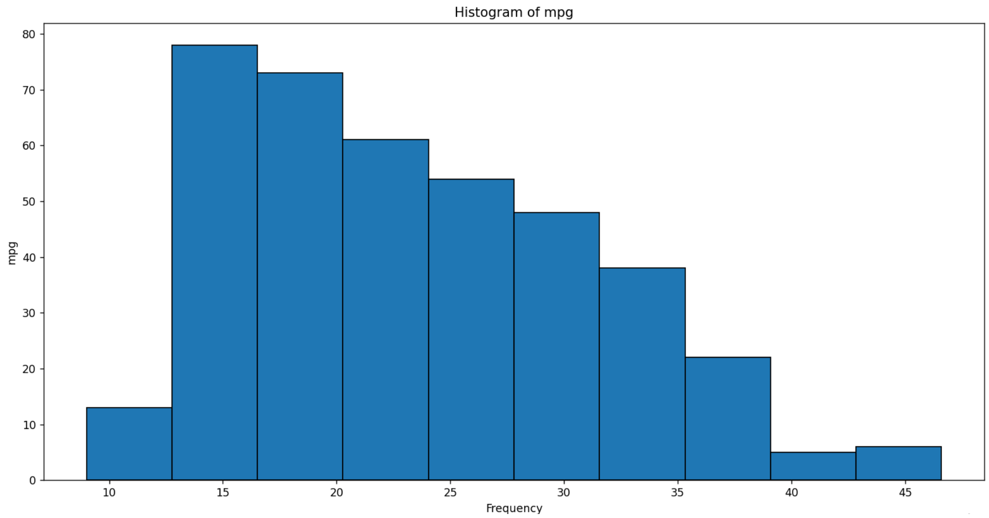
NOTE: The median for europe, Asia and USA respectively are : 26.5, 31.55 and 18.55 mpg.

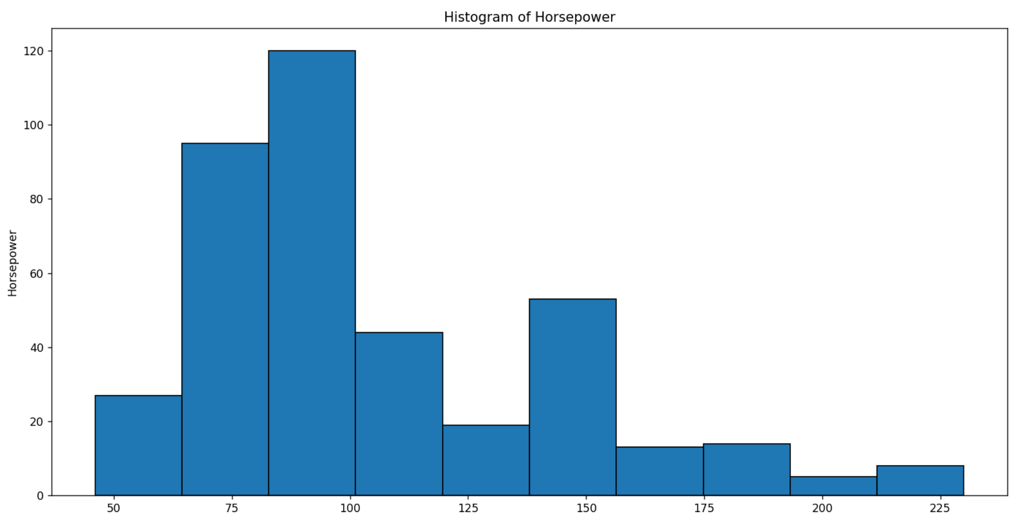
Part 5:

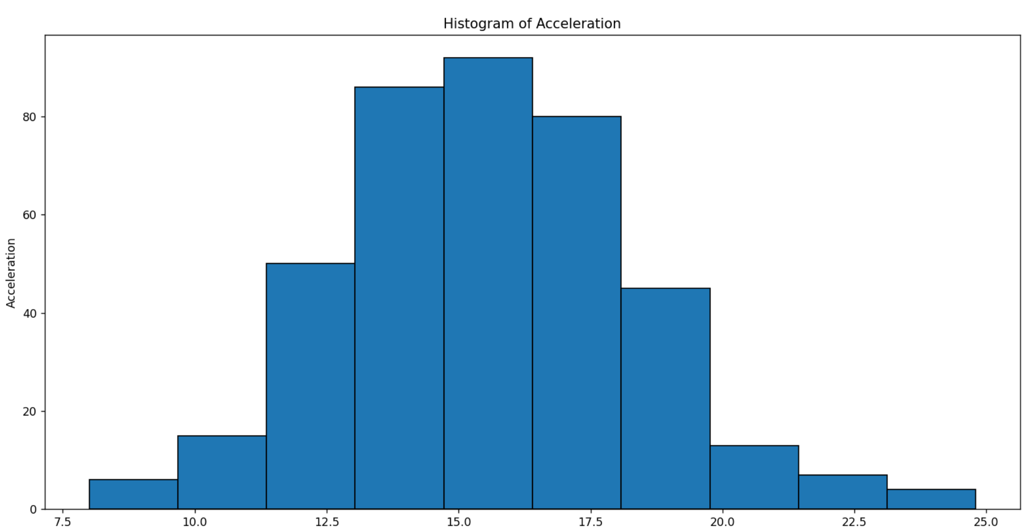
Code:

```
ML_3_UP_TO_10.py    ML_3.py

# ----------------------------This is the fifth part-----------------------------

# First, the histogram for the acceleration
plt.hist(dataframe['acceleration'], bins=10, edgecolor='black')  # Adjust the number of bins as needed
plt.xlabel('Frequency')
plt.ylabel('Acceleration')
plt.title('Histogram of Acceleration')
plt.show()

# #Second, the histogram for the horsepower
plt.hist(dataframe['horsepower'], bins=10, edgecolor='black')  # Adjust the number of bins as needed
plt.xlabel('Frequency')
plt.ylabel('Horsepower')
plt.title('Histogram of Horsepower')
plt.show()

#Third, the histogram for the mpg
plt.hist(dataframe['mpg'], bins=10, edgecolor='black')  # Adjust the number of bins as needed
plt.xlabel('Frequency')
plt.ylabel('mpg')
plt.title('Histogram of mpg')
plt.show()

```

Output:


Histogram of mpg
mpg
Frequency
0
10
20
30
40
50
60
70
80
10
15
20
25
30
35
40
45


Histogram of Horsepower
Horsepower
0
20
40
60
80
100
120
50
75
100
125
150
175
200
225


Histogram of Acceleration
Acceleration
Frequency
0
20
40
60
80
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0

Explanation:

I first plotted the histogram for features :acceleration, horsepower and mpg by the .hist method in matplotlib package in python. We can notice that the Histogram of Acceleration is the most similar to the Gaussian distribution.

Part 6:

Code:

```python
# I found the skewness for acceleration, horsepower and mpg features
skewness_series = dataframe[['acceleration', 'horsepower', 'mpg']].skew()
print("\n\nThe Skewness for acceleration is {}".format(skewness_series['acceleration']))
print("The Skewness for horsepower is {}".format(skewness_series['horsepower']))
print("The Skewness for mpg is {}".format(skewness_series['mpg']))
```

Output:

```
The Skewness for acceleration is 0.27877684462588986
The Skewness for horsepower is 1.0330029083003485
The Skewness for mpg is 0.45706634399491913
```

Explanation:

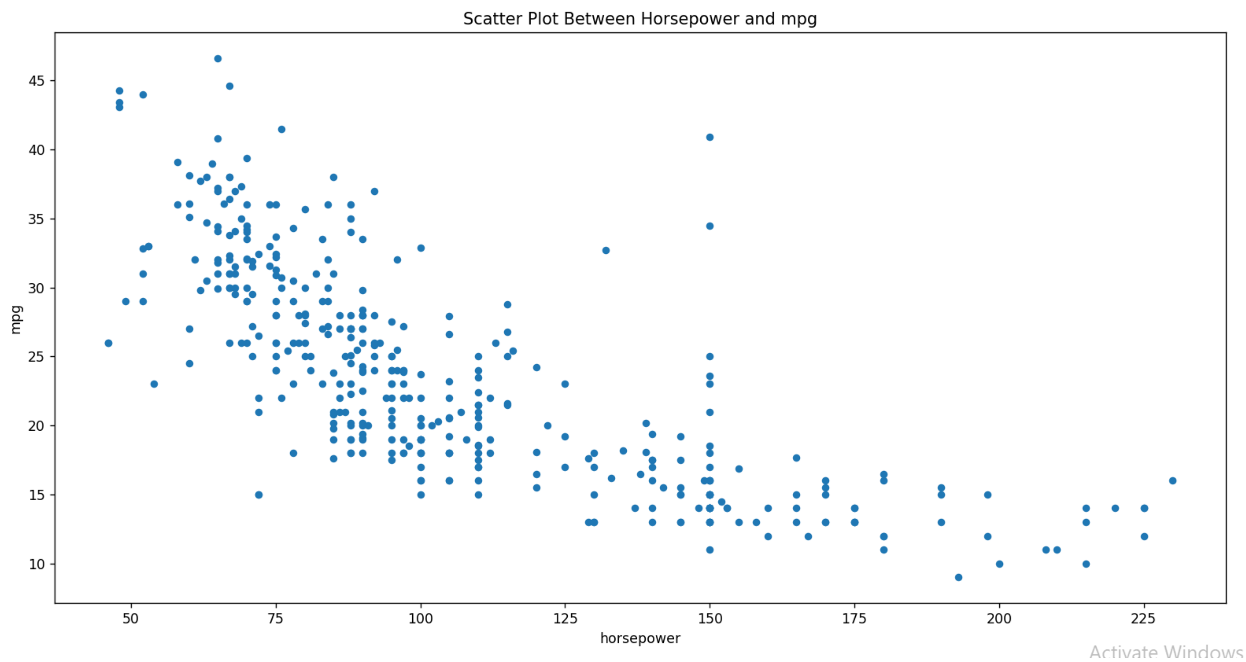
I used skewness as a parameter to determine if the distribution is most similar to the gaussian distribution or not. The method .skew returns a series with the feature as an index and the skew is the value for this index. We can notice that the skewness for acceleration feature is the least value between them, and according to the relationship with skewness which calls that the smallest value to 0 is nearest to the normal distribution.

Part 7:

Code:

```python
# ------------------------------This is the Seventh part------------------------------

dataframe.plot(kind='scatter', x='horsepower', y='mpg', title='Scatter Plot Between Horsepower and mpg')
plt.show()
correlation = dataframe['horsepower'].corr(dataframe['mpg'])
print("\n\nCorrelation Coefficient between mpg and horsepower features is {}".format(correlation))
```

Output:




```
Correlation Coefficient between mpg and horsepower features is -0.7531769820344798
```

Explanation:

I plotted in a scatter plot the relationship between mpg (target values) and the values for the horsepower feature. We can notice that the correlation is negative, and the reason for that is when the horsepower has small values, the mpg has high values and exists Intensely and vice versa for big values of the horsepower feature.

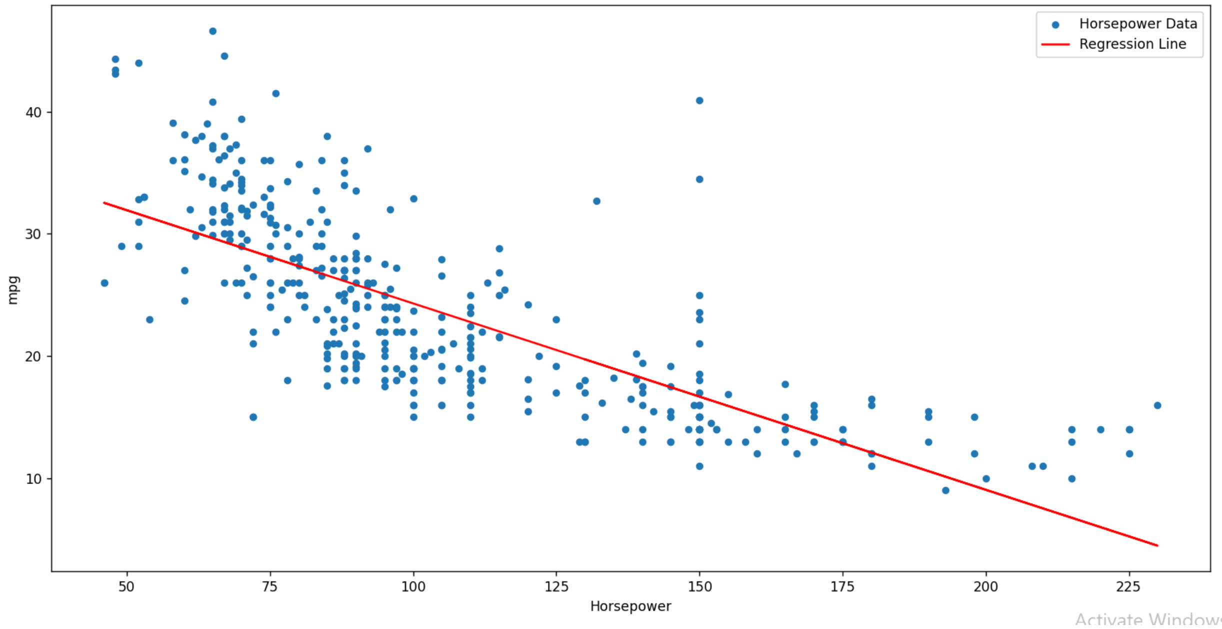
NOTE: The value for the correlation between the horsepower feature and the mpg is -0.7, which confirms our previous speech.

Part 8:

Code:

```python
# ------------------------------This is the Eighth part------------------------------
# Create the closed form solution
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
dataframe['X0'] = [1 for _ in range(398)]  # make a feature with ones for the intercept
X = dataframe[['X0', 'horsepower']].values  # make the data matrix from X0 and Horsepower features
Y = dataframe['mpg'].values  # make an array from the target values
XT = np.transpose(X)  # make a transpose matrix
XT_X = np.dot(XT, X)  # make a dot product between X and X transpose
XT_X_inv = np.linalg.inv(XT_X)
XT_X_inv_XT = np.dot(XT_X_inv, XT)
W = np.dot(XT_X_inv_XT, Y)  # make the weights from the closed from solution (slope and the intercept)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Create the predicted output
predicted_output = W[0] + W[1]*dataframe['horsepower']

# Plot the scatter plot
dataframe.plot(kind='scatter', x='horsepower', y='mpg', label='Horsepower Data')

# Plot the regression line
plt.plot(dataframe['horsepower'], predicted_output, color='red', label='Regression Line')

# Add labels and a legend
plt.xlabel('Horsepower')
plt.ylabel('mpg')
plt.legend()
plt.show()
```

Output:




```
W[0] = 39.551289806757794
W[1] = -0.15250438074858036
```

Explanation:

I implemented the simple linear regression for this question from scratch, and the idea for that is to use the closed form solution $W = (X^TX)^{-1}X^TY$. After that I just printed the scatter plot and printed the linear line on it. We can notice that the values for the weights are in the picture.

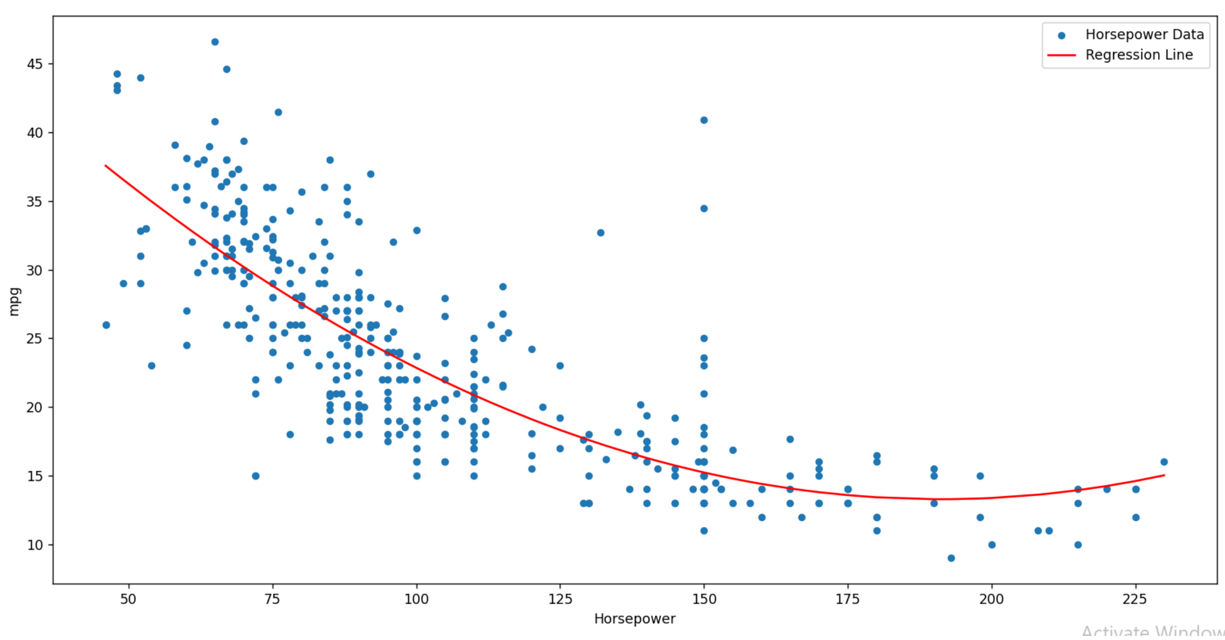
Part 9:

Code:

ML_3_UP_TO_10.py ML_3.py

```python
# ----------------------------This is the Ninth part----------------------------
# Create the closed form solution
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
dataframe['X_square'] = dataframe['horsepower'] ** 2  # make X^2 to learn the model
X = dataframe[['X0', 'horsepower', 'X_square']].values  # make the data matrix from X0, X and X^2 features
Y = dataframe['mpg'].values  # make an array from the target values
XT = np.transpose(X)  # make a transpose matrix
XT_X = np.dot(XT, X)  # make a dot product between X and X transpose
XT_X_inv = np.linalg.inv(XT_X)
XT_X_inv_XT = np.dot(XT_X_inv, XT)
W = np.dot(XT_X_inv_XT, Y)  # make the weights from the closed from solution (slope and the intercept)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create the predicted output after sorting
X = dataframe['horsepower'].sort_values()
X_square = X ** 2
predicted_output = W[0] + W[1]*X + W[2]*X_square

# Plot the scatter plot
dataframe.plot(kind='scatter', x='horsepower', y='mpg', label='Horsepower Data')

# Plot the regression line
plt.plot(X, predicted_output, color='red', label='Regression Line')

# Add labels and a legend
plt.xlabel('Horsepower')
plt.ylabel('mpg')
plt.legend()
plt.show()
```

Output:



```
For degree = 2 ------> W[0] = 55.396212698135095, W[1] = -0.4407150894966449, W[2] = 0.0011529068807191753
```

Explanation:

I created a new column and added it to the dataframe which contains the squared values for the 'horsepower' feature in order to create the learning from degree 2 for the following equation: f(X) = W0 + W1X + W2X^2. After that I created the closed form solution. Then, I sorted the data in order to get one plot and printed the scatter plot and the polynomial graph. We can notice that the line is the same as X^2 and the values for the weights are in the next picture.

Part 10:

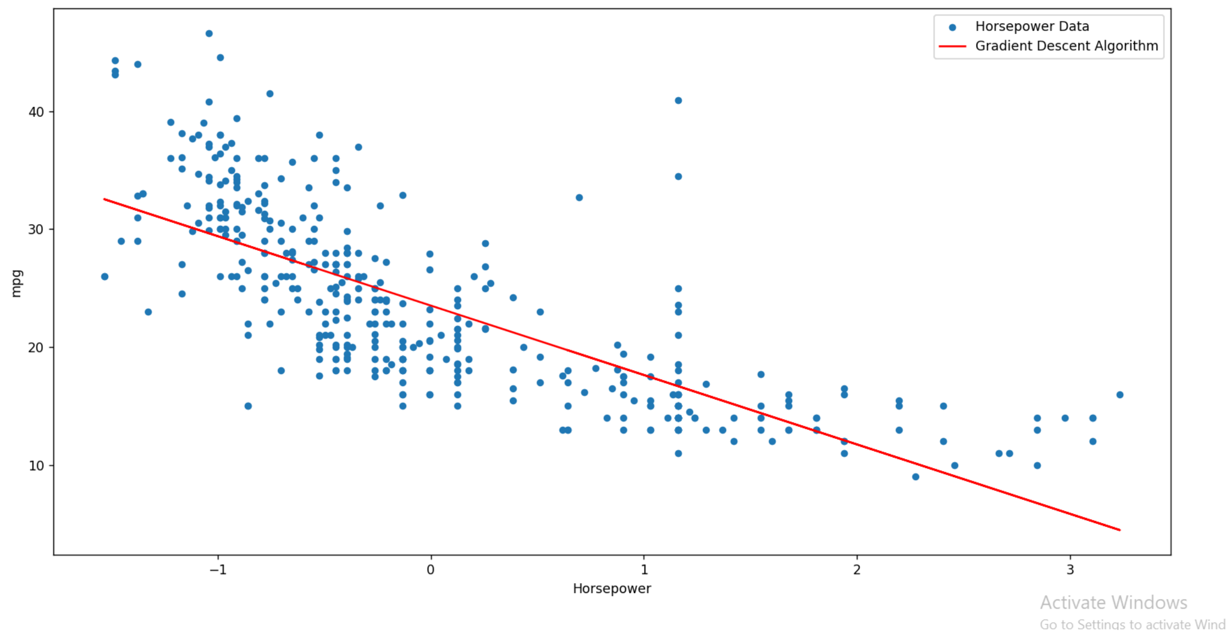
Code:

```
def get_cost(scaled_values_for_X0, scaled_values_for_horsepower, target_values, n, W):
    cost = 0
    for i in range(n):
        f_x = W[0] * scaled_values_for_X0[i] + W[1] * scaled_values_for_horsepower[i]
        cost = cost + (f_x - target_values[i]) ** 2
    cost = cost / n
    return cost


# Prepare the data
# I made a scaling by Z-score for the values to converge faster to the weights
mean = dataframe['horsepower'].mean()
std = dataframe['horsepower'].std()
dataframe['horsepower'] = (dataframe['horsepower'] - mean) / std

scaled_values_for_X0 = dataframe['X0'].values
scaled_values_for_horsepower = dataframe['horsepower'].values
target_values = dataframe['mpg'].values

W = [5, 3]  # initial weights
alpha = 0.1  # Learning rate
n = 398  # Number of examples
prev_cost = get_cost(scaled_values_for_X0, scaled_values_for_horsepower, target_values, n, W)
```

```python
# Making infinity loop until the data converge

while True:
    gradients = []
    X0_summation = 0
    horsepower_summation = 0

    # Make a gradient
    for i in range(0, n):
        # Make the summation for horsepower and for X0
        f_x = W[0] * scaled_values_for_X0[i] + W[1] * scaled_values_for_horsepower[i]

        X0_summation = X0_summation + (f_x - target_values[i]) * scaled_values_for_X0[i]
        horsepower_summation = horsepower_summation + \
                               (f_x - target_values[i]) * scaled_values_for_horsepower[i]

    gradients.append((2 / n) * X0_summation)
    gradients.append((2 / n) * horsepower_summation)

    #Change the value of W to the new ones
    W[0] = W[0] - alpha * gradients[0]
    W[1] = W[1] - alpha * gradients[1]

    current_cost = get_cost(scaled_values_for_X0, scaled_values_for_horsepower, target_values, n, W)
    final_cost = abs(current_cost - prev_cost)
```

```python
    if final_cost < 1e-12:
        break
    else:
        prev_cost = current_cost
print("\n\nBy the Gradient Descent -----> W[0] = {}, W[1] = {}".format(W[0], W[1]))

predicted_output = W[0] * dataframe['X0'] + W[1] * dataframe['horsepower']
# Plot the scatter plot
dataframe.plot(kind='scatter', x='horsepower', y='mpg', label='Horsepower Data')
# Plot the regression line
plt.plot(dataframe['horsepower'], predicted_output, color='red', label='Gradient Descent Algorithm')
plt.xlabel('Horsepower')
plt.ylabel('mpg')
plt.legend()
plt.show()
```

Output:




```
By the Gradient Descent -----> W[0] = 23.514571866017814, W[1] = -5.886818973883748
```

Explanation:

I first implemented the cost (mean square error) in the function get_cost(). I did the normalization for the 'horsepower' feature, created the initial points (W, alpha). I iterated for ever until the error difference between the consecutive iterations is less than <10^-12 , (in the loop I calculated the gradients and found the parameters W for the following formula:

$$\hat{w}_i^{(t+1)} = \hat{w}_i^{(t)} - \alpha \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)\, xij$$

) . We notice that the gradient descent algorithm graph is almost the same for the linear regression case. The values for W and the graph are in the pictures.