Faculty of Engineering & Technology
Electrical & Computer Engineering Department

MACHINE LEARNING AND DATA SCIENCE - ENCS5341

Assignment #1

Prepared by:
Yazeed Hamdan 1201133
Instructor: Dr. Yazan Abu Farha

Section: 2
Date: 27/11/2023 Birzeit

Part 1:

```
# Yazeed Hamdan 1201133
# part 1
file_path = r'C:\Users\hp\Desktop\ML\cars.csv'  #Load a CSV file containing cars data
Cars_DataFrame = pd.read_csv(file_path)
# Checking the number of features and examples
total_features  = Cars_DataFrame.shape[1]
total_examples  = Cars_DataFrame.shape[0]
print("\n////////////////// Part 1 ////////////////////////////\n")
# Prints the number of features and examples in the dataset
print("Number of Features:", total_features)
print("Number of Examples:", total_examples )
```

```
////////////////// Part 1 ////////////////////////////


Number of Features: 8
Number of Examples: 398
```

In this part, a CSV file containing cars data is loaded into a Data Frame called Cars_DataFrame using the pandas library. The path to the CSV file is specified in the file_path variable. Once loaded, the code calculates and prints the number of features (columns) and examples (rows) in the DataFrame. Features are different attributes of information about the cars, like model, horsepower, mpg, etc.

Part 2:

```
# part 2
print("\n////////////////// Part 2 ////////////////////////////\n")
missing_values = Cars_DataFrame.isnull().sum() # Checking for missing values in each feature
print(missing_values) # print the number of missing values in each feature
```

```
////////////////// Part 2 ////////////////////////////


mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          2
dtype: int64
```

In part 2 of the code, the number of missing values in each column of the Cars_DataFrame was calculated then printed. This is done using the isnull().sum() method.
isnull() checks each cell in the Data Frame and returns True if the cell is missing a value (null) and False otherwise.
sum () method then adds up these True values for each column, giving the total count of missing values. The output shows that most columns have no missing values, but the 'horsepower' column has 6 missing values, and the 'origin' column has 2 missing values.

Part 3:

```
# part 3
print("\n////////////////// Part 3 ////////////////////////////\n")
horsepower_median = Cars_DataFrame['horsepower'].median() # For horsepower the median was used
origin_mode = Cars_DataFrame['origin'].mode()[0] # For origin the mode was used
#fill missing values
for index, row in Cars_DataFrame.iterrows():
    if pd.isnull(row['horsepower']):
        Cars_DataFrame.at[index, 'horsepower'] = horsepower_median
    if pd.isnull(row['origin']):
        Cars_DataFrame.at[index, 'origin'] = origin_mode

# Calculate the number of missing values
updated_missing_values = Cars_DataFrame.isnull().sum()
print(updated_missing_values)
print("\n")
print("Median:",horsepower_median)
print("Mode:",origin_mode)
```

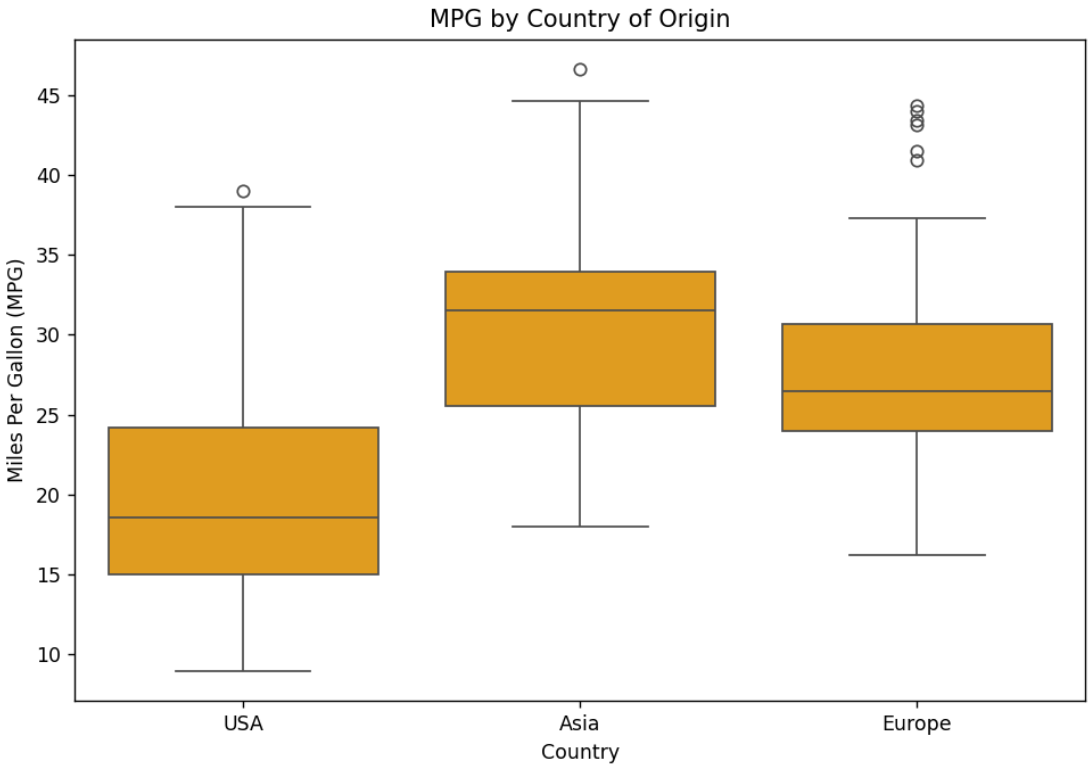
```
////////////////// Part 3 ////////////////////////////


mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
origin          0
dtype: int64


Median: 93.5
Mode: USA
```

In Part 3 of the code, missing values in the Cars_DataFrame were filled. For the horsepower column, missing values were filled with the median value of the existing horsepower data, while missing values in the origin column were filled with the mode of the origin data.
This is done using a loop that iterates over each row and checks for missing values in these columns. After filling the missing values, the number of missing values for each column was calculated to confirm that there are no longer any missing values. The final output shows zero missing values in all columns, indicating that all missing data

have been successfully filled. The median value used for horsepower is 93.5, and the mode of origin (most common) filled in is USA.

Part 4:

```
# Part 4
plt.figure(figsize=(8, 5)) # size of the figure
sns.boxplot(x='origin', y='mpg', data=Cars_DataFrame, color='orange')
# Set the title and x,y labels in the figure
plt.title('MPG by Country of Origin')
plt.xlabel('Country')
plt.ylabel('Miles Per Gallon (MPG)')
plt.show()
```

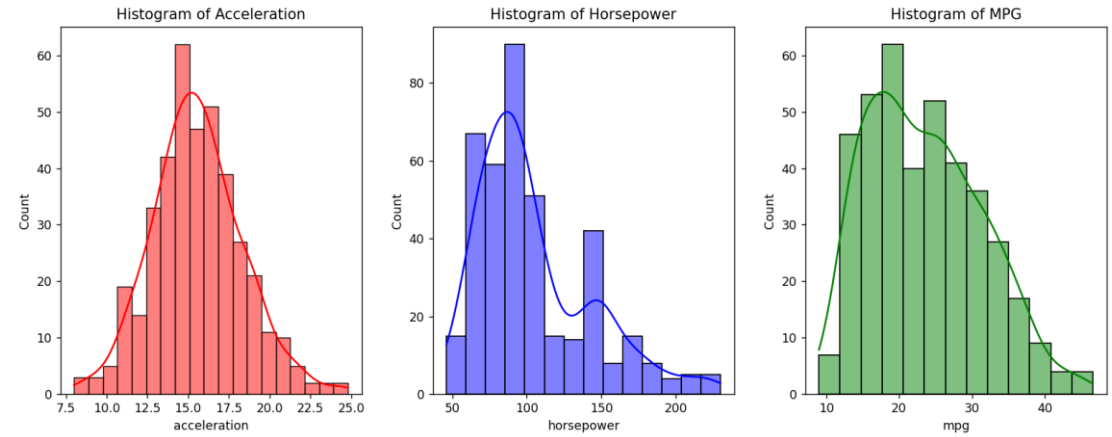


In Part 4 a plot is created to compare the fuel efficiency, measured in miles per gallon (MPG) of cars from different countries of origin. Seaborn library was used to draw the box plot with MPG values on the y-axis and the country of origin on the x-axis. In the output plot the central tendency and spread of MPG values are shown for cars from the USA, Asia, and Europe. Each box shows the median MPG, and any outliers observed.

From the plot it can be noticed that Asian cars generally go the furthest on a gallon of gas, making them the most fuel-efficient, followed by European cars then, USA cars being the least efficient on average. Also, it can be noticed that Europe has several outliers indicating variation in fuel efficiency.

Part 5:

```
# part 5
plt.figure(figsize=(10, 5)) # figure size
# Histogram for acceleration
plt.subplot(1, 3, 1)
# creates a histogram with a Kernel Density Estimate (kde)
# To provide a smooth curve that gives a visualization of the shape of the data distribution
sns.histplot(Cars_DataFrame['acceleration'], kde=True, color='red')
plt.title('Histogram of Acceleration')
# Histogram for horsepower
plt.subplot(1, 3, 2)
sns.histplot(Cars_DataFrame['horsepower'], kde=True, color='blue')
plt.title('Histogram of Horsepower')
# Histogram for mpg
plt.subplot(1, 3, 3)
sns.histplot(Cars_DataFrame['mpg'], kde=True, color='green')
plt.title('Histogram of MPG')
plt.tight_layout() # to adjusts the layout
plt.show()
```



Part 5 code plots three histograms of acceleration, horsepower, and mpg with overlaid smooth curves that estimate the pattern of the data, to examine their distributions. From the graphs it can be noticed that the acceleration is mostly around the middle range, horsepower has a big peak at lower values, and mpg is more spread out with peaks at lower and higher values.

This tells that most cars have moderate acceleration, there are lots of less horsepower cars, and mpg varies widely among cars. The acceleration histogram appears quite symmetric and bell-shaped, so it can be considered as the most similar to a Gaussian distribution.

Part 6:

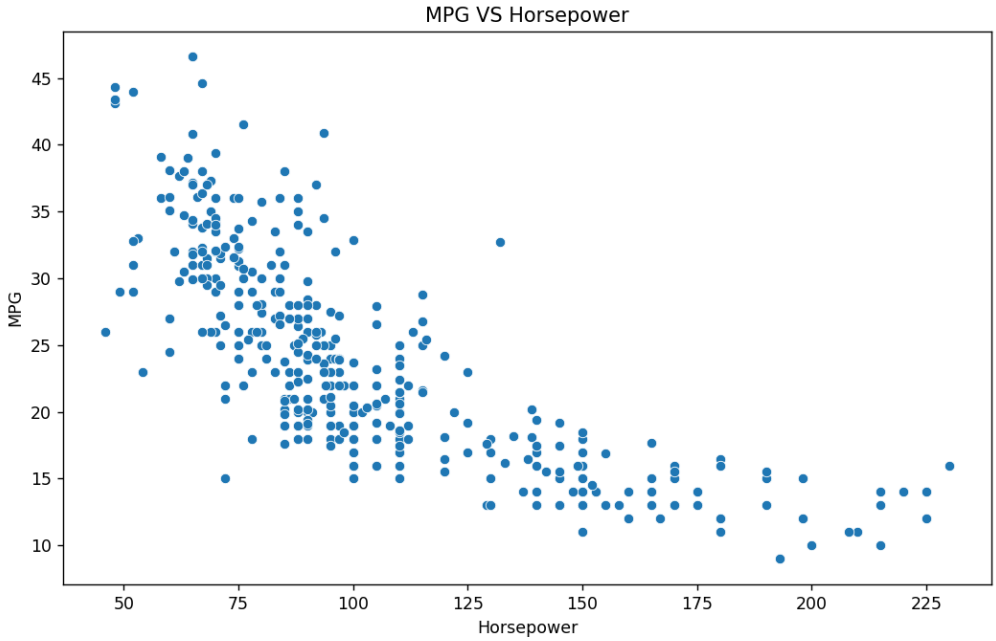
```
# part 6
print("\n////////////////// Part 6 ////////////////////////////\n")
# Calculate skewness for each feature using skew()
skewness_acceleration = Cars_DataFrame['acceleration'].skew()
skewness_horsepower = Cars_DataFrame['horsepower'].skew()
skewness_mpg = Cars_DataFrame['mpg'].skew()
# Compare absolute skewness values to find the closest to zero
if abs(skewness_acceleration) < abs(skewness_horsepower) and abs(skewness_acceleration) < abs(skewness_mpg):
    closest_to_gaussian = 'acceleration'
    skew_value = skewness_acceleration
elif abs(skewness_horsepower) < abs(skewness_acceleration) and abs(skewness_horsepower) < abs(skewness_mpg):
    closest_to_gaussian = 'horsepower'
    skew_value = skewness_horsepower
else:
    closest_to_gaussian = 'mpg'
    skew_value = skewness_mpg

# Print the feature with skewness value closest to zero
print(f"\nThe feature with a distribution most similar to a Gaussian is: {closest_to_gaussian} with a skewness of {skew_value}")
print("skewness of horsepower =",skewness_horsepower)
print("skewness of mpg =",skewness_mpg)
```

```
////////////////// Part 6 ////////////////////////////


The feature with a distribution most similar to a Gaussian is: acceleration with a skewness of 0.27877684462588986
skewness of horsepower = 1.1062242930872614
skewness of mpg = 0.4570663439491913
```

The previous code compares the distribution of acceleration, horsepower, and mpg to see which one is shaped most like a Gaussian (normal) curve, which is a common. A number called skewness was calculated for each feature. The feature with the skewness number closest to zero is the one that looks most like a Gaussian curve. According to the results, acceleration has a skewness closest to zero, which means its distribution is the most symmetrical and bell-shaped compared to horsepower and mpg.

Part 7:

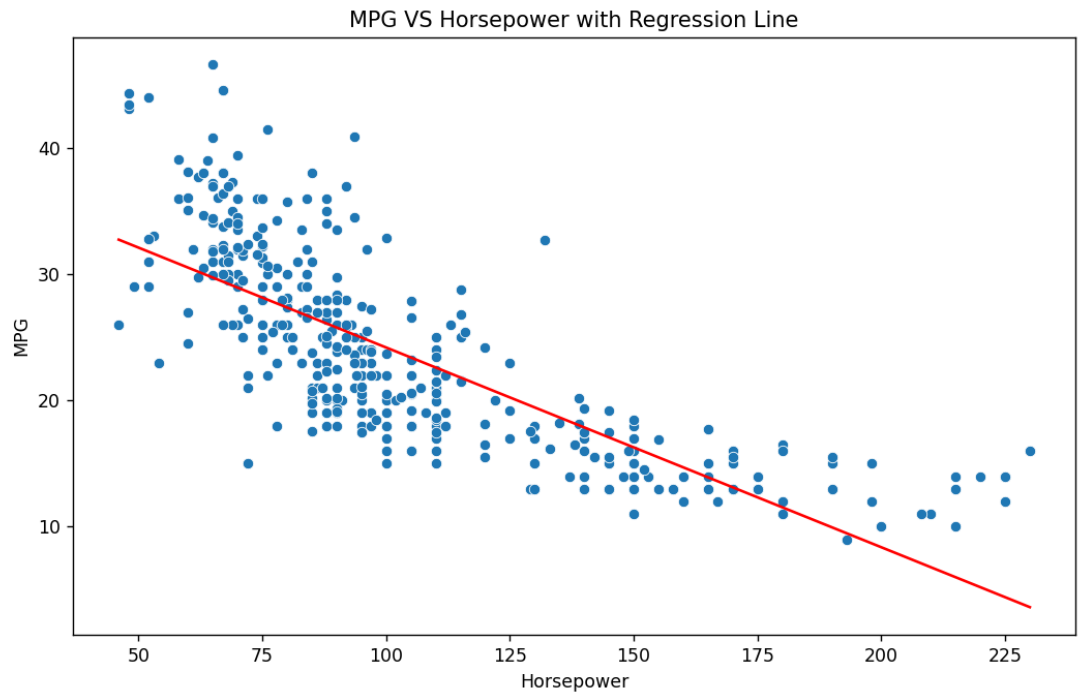
```
# part 7
plt.figure(figsize=(10, 6)) # figure size
sns.scatterplot(x='horsepower', y='mpg', data=Cars_DataFrame)
# Set the title and x,y labels for the figure
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.title('MPG VS Horsepower')
plt.show()
```



The code in Part 7 plots a scatter plot to show the relationship between a car's horsepower and the MPG. Each point on the plot represents a car, with its position determined by its horsepower and its MPG. The plot shows that as horsepower increases, MPG tends to decrease. This indicates a negative correlation between horsepower and MPG.

Part 8:

```
# part 8
# Adding x0 = 1 to the dataset for the intercept
X = Cars_DataFrame['horsepower']
X = np.c_[np.ones(X.shape[0]), X]  # Adding the intercept term
y = Cars_DataFrame['mpg'] # Target variable
VectorOfW = np.linalg.inv(X.T @ X) @ X.T @ y # Closed form solution
# Plotting the scatter plot and the learned regression line
plt.figure(figsize=(10, 6))
sns.scatterplot(x='horsepower', y='mpg', data=Cars_DataFrame)
# Plotting the regression line
x_values = np.array([Cars_DataFrame['horsepower'].min(), Cars_DataFrame['horsepower'].max()])
y_values = VectorOfW[0] + VectorOfW[1] * x_values
plt.plot(x_values, y_values, color="red")
plt.title('MPG VS Horsepower with Regression Line')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.show()
print("\n////////////////// Part 8 ////////////////////////////\n")
print ("parameters values:", VectorOfW)
```

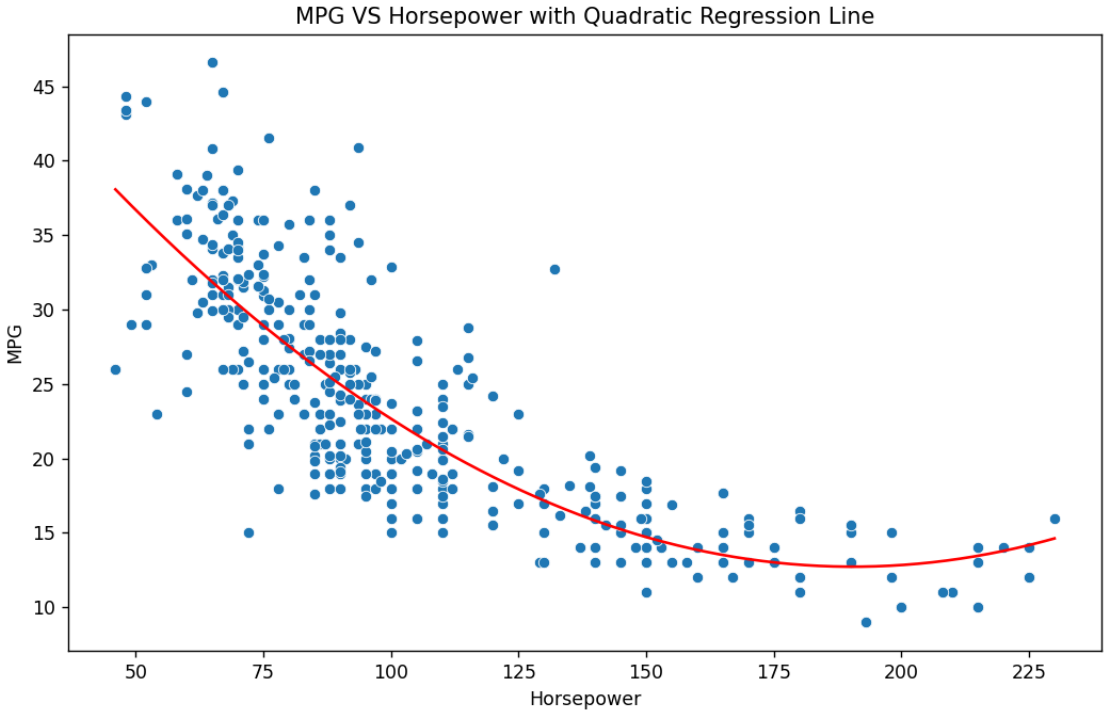



```
////////////////// Part 8 ////////////////////////////

parameters values: [40.01131787 -0.1581602 ]
```

In Part 8 code, closed form solution of the linear regression was used to draw a linear line that shown above through a scatter plot, this line helps to understand the general relation between horsepower increases and MPG decreases, in the code, closed form solution was used to find the values of parameters (W0, W1) that gives the best line that fits through the points. The values of the parameters from this model are 40.01131787 for the intercept and -0.1581602 for the slope.

Part 9:

```python
# Part 9
X = Cars_DataFrame['horsepower'].values
X_quad = np.column_stack((np.ones(X.shape), X, X**2)) # prepare the feature matrix for a quadratic regression model
VectorOfW_quad = np.linalg.inv(X_quad.T @ X_quad) @ X_quad.T @ y # Simplified Quadratic Regression Model
plt.figure(figsize=(10, 6))
sns.scatterplot(x='horsepower', y='mpg', data=Cars_DataFrame)
# Quadratic Regression Line calculations
x_values_quad = np.linspace(X.min(), X.max(), 400)
# calculate the y coordinates for the quadratic regression
y_values_quad = VectorOfW_quad[0] + VectorOfW_quad[1] * x_values_quad + VectorOfW_quad[2] * x_values_quad**2
plt.plot(x_values_quad, y_values_quad, color = 'red') # plot quadratic regression line
plt.title('MPG VS Horsepower with Quadratic Regression Line')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.show()
print("\n////////////////// Part 9 ////////////////////////////\n")
print("parameters values:",VectorOfW_quad)
```

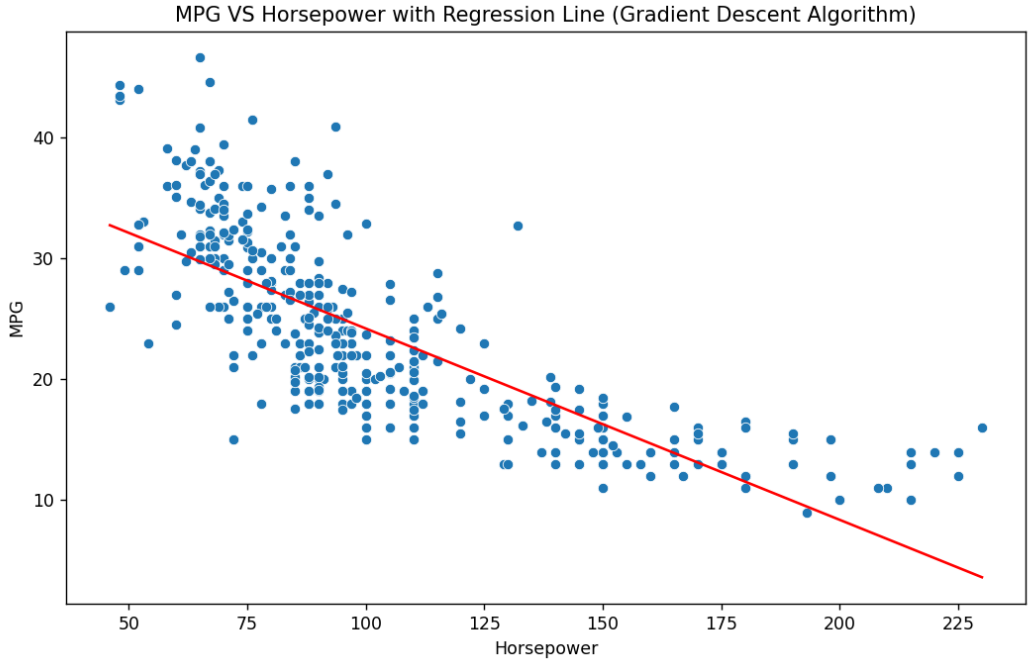



```
////////////////// Part 9 ////////////////////////////

parameters values: [ 5.67812082e+01 -4.62563379e-01  1.21431773e-03]
```

In this part a quadratic regression was used to understand the relationship between a horsepower and MPG, which allows for a curved line to fit the data. The code calculates the best fitting curve and plots it on the scatter plot.
This curve fits better than a linear line for the dataset. Parameters at the end are represent the coefficients of the quadratic regression equation fitted to the relationship between horsepower and MPG in the dataset. They can be denoted as W0, W1, and W2 respectively and the values are shown above.

Part 10:

```python
# part 10
# Normalize the horsepower data using Z-score normalization
HP_mean = Cars_DataFrame['horsepower'].mean() #calculate the mean
HP_std = Cars_DataFrame['horsepower'].std() #calculate standard deviation
X = (Cars_DataFrame['horsepower'].values - HP_mean) / HP_std
X = np.c_[np.ones(X.shape[0]), X]  # Adding intercept term
y = Cars_DataFrame['mpg'].values
# Initialize Parameters for Gradient Descent
Vector_W = np.array([0.0, 0.0])  # Both w0 and w1 set to 0
learningRate = 0.1
iterations = 10000  # Number of iterations

# Gradient Descent Algorithm
for _ in range(iterations):
    predictions = X @ Vector_W
    Errors = predictions - y
    gradients = X.T @ Errors / len(X)
    Vector_W -= learningRate * gradients
plt.figure(figsize=(10, 6)) # Plotting the Results
sns.scatterplot(x='horsepower', y='mpg', data=Cars_DataFrame)
# Regression Line
x_values = np.linspace(Cars_DataFrame['horsepower'].min(), Cars_DataFrame['horsepower'].max(), 100)
x_standardized = (x_values - HP_mean) / HP_std
x_standardized = np.c_[np.ones(x_standardized.shape[0]), x_standardized]
y_values = x_standardized @ Vector_W
plt.plot(x_values, y_values, color='red')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.title('MPG VS Horsepower with Regression Line (Gradient Descent Algorithm)')
plt.show()
print("\n////////////////// Part 10 ////////////////////////////\n")
print("parameters values :", Vector_W)
print("\n")
```




```
////////////////// Part 10 ////////////////////////////

parameters values : [23.51457286 -6.04529811]
```

Part 10 code performs a linear regression using gradient descent algorithm on the relationship between horsepower and MPG.
It was start by normalize the horsepower using Z-score normalization, this involves subtracting the mean from the feature and dividing by the standard deviation.
This step was done to make the gradient descent can work more effectively.

In gradient descent algorithm, within a loop, the algorithm repeatedly calculates the predicted MPG values using the current Vector_W, then, computes the errors between these predictions and the actual MPG values, and then updates Vector_W to reduce these errors to get the best linear line that represents how MPG changes with horsepower, and to be more similar to the plot of simple linear regression case. The number of iteration and learning rate were changed until there is no change on the values of the parameters (Vector_W).
The plot was observed as shown above, and it can be noticed that its very similar to part 8 plot, so it can be concluded that the plot is correct.