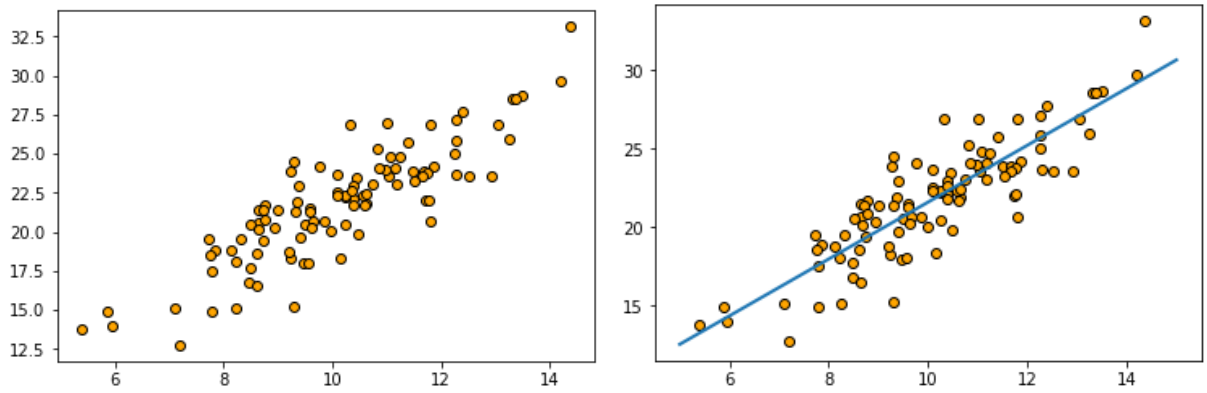
COSC 130 – Project 01 Instructions

Introduction to Linear Regression

Linear regression is a machine learning technique used to predict or estimate the value of a continuous variable, called the **response** variable, based on the values of one or more input variables, which are called **predictors**. In this project, we will be walking through an example of creating a linear regression model to predict the miles per gallon obtained by a vehicle based on a single predictor, which will be the weight of the vehicle.

A linear regression model with only one predictor is known as a **simple linear regression** model. When creating a simple linear regression model, it is typical to denote the response variable as Y and the predictor variable as X. Suppose that we are provided with n paired observations of X and Y values. We will denote the observed values of X by $x_1, x_2, \dots, x_n$ and will denote the observed values of Y by $y_1, y_2, \dots, y_n$. We can combine the paired values of X and Y into points of the following form: (x_i, y_i). We can plot these points as a scatter plot to get an idea of the relationship between X and Y. Suppose that our plot looks like this:



If the points seem to fall near a line, as is the case in the plot above, then we might try to use the equation for this line to explain the relationship between the variables X and Y. In other words, we might explain the relationship between the two variables using a **linear model** of the following form:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot X$$

In the equation above, $\hat{Y}$ is used to denote an predicted, or estimated value of Y (as opposed to a true, observed value). The symbols $\hat{\beta}_0$ and $\hat{\beta}_1$ represent the intercept and slope of the line. These are "learned" from the data in order to select the line that provides the "best" fit for the data. We will not explain in this project exactly how any one particular linear model is compared to another, but will instead provide formulas for calculating values for $\hat{\beta}_0$ and $\hat{\beta}_1$ that are considered to be "best" according to one commonly used metric.

Description of the Dataset

The dataset that you will be working with in this project contains information about 398 vehicle models manufactured in 1983. For each vehicle, we have the following pieces of information:

- The weight of the vehicle, measured in pounds.
- The average miles per gallon (MPG) for the model.
- The natural logarithm of average MPG for the model.

Our goal will be to create a linear model that can be used to estimate the natural logarithm of average MPG for a vehicle based on the vehicle's weight. In other words, log-MPG will be used as the response variable in our model, and weight will be the predictor. We will discuss later why it is that we are using the natural logarithm of MPG as the response as opposed to the MPG itself.

General Instructions

Create a new notebook named **`Project_01_YourLastName.ipynb`**. Instructions for adding cells to this notebook are provided throughout these instructions. Download the data file **`auto_data.txt`** from either Canvas or CoCalc, and place it into the same folder as the notebook you created.

Any set of instructions you see in this document with an orange bar to the left will indicate a place where you should create a markdown cell. If no instructions are provided regarding formatting, then the text should be unformatted.

Any set of instructions you see with a blue bar to the left will provide instructions for creating a single code cell.

Read the instructions carefully.

Assignment Header and Introduction

Create a markdown cell with a level 1 header that reads: "COSC 130 – Project 01". Add your name below that as a level 3 header.

Add another markdown cell containing a brief description of the goal for this project (No more than 2 or 3 lines are required). You may base this off of the information provided on the previous page, but please use your own words. The text in this cell should be unformatted.

We will need to use external tools (called libraries) for importing and visualizing data. We will discuss how to load and use libraries later in this course. But for now, you will be provided with the code required to load two libraries, Pandas and Matplotlib.

Copy the code below (without the indentation) into a code cell and then execute that cell.

```
import pandas as pd
import matplotlib.pyplot as plt
```

Part 1: Importing and Viewing the Data

Create a markdown cell with a level 2 header that reads "**Part 1: Importing and Viewing the Data**".

In the same cell, add unformatted text explaining that the first tasks will be to import and view the data.

We will now use the Pandas library to import the dataset.

Copy the code below (without the indentation) into a code cell and then execute that cell.

```
df = pd.read_table(filepath_or_buffer='auto_data.txt', sep='\t')
weight = list(df.wt)
mpg = list(df.mpg)
ln_mpg = list(df.ln_mpg)
```

The code cell above created three lists named **`weight`**, **`mpg`**, and **`ln_mpg`**. Each of these lists contains the indicated information for each of the 398 vehicle models. The lists are "parallel" in the sense that the values stored at the same index in each of the three lists correspond to the same vehicle model.

Create a markdown cell with unformatted text explaining that you will now confirm each list contains 398 values.

Print the lengths of the three lists **`weight`**, **`mpg`**, and **`ln_mpg`**.

We will now view the information for the first 10 vehicles in the dataset. Create a markdown cell with unformatted text to explain this.

Use a code cell to print the first 10 values of each of the lists **`weight`**, **`mpg`**, and **`ln_mpg`**. Format the output as follows:

- The list values should be arranged in columns, with each row of output corresponding to one vehicle model.
- The output should include column headers and a dividing line, as shown below.
- The number of characters reserved for each column, in order, should be 6, 8, and 10.
- The columns should be right aligned.

The first few rows of your output should look exactly as shown below:

```
Weight     MPG    LN_MPG
-----------------------
  3190    27.2    3.3032
  1985    32.8    3.4904
```

We will now create two scatter plots. The first will explore the relationship between MPG and vehicle weight. The second will explore the relationship between natural log of MPG and vehicle weight. Create a markdown cell with unformatted text to explain this.

We will discuss how to use the Matplotlib library to create data visualizations later in this course. For now, the necessary code will be provided to you.

Copy the code below (without the indentation) into a code cell and then execute that cell.

```
plt.figure(figsize=[12,4])
plt.subplot(1,2,1)
plt.scatter(weight, mpg, c='skyblue', edgecolor='k')
plt.xlabel('Weight (in Pounds)')
plt.ylabel('MPG')
plt.title('Plot of MPG against Weight')

plt.subplot(1,2,2)
plt.scatter(weight, ln_mpg, c='skyblue', edgecolor='k')
plt.xlabel('Weight (in Pounds)')
plt.ylabel('Natural Logarithm of MPG')
plt.title('Plot of Log-MPG against Weight')
plt.show()
```

Notice that the relationship between MPG and weight in the first scatter plot seems to have a slight bend or curve, whereas the relationship between log-MPG and weight appears to be mostly linear. Since we will be constructing a linear model, we will use log-MPG as the response variable in our model.

Create a markdown cell with unformatted text to paraphrase the observations made in the preceding above.

Part 2: Splitting the Data

When training a machine learning model, it is common practice to split the dataset into two separate sets, referred to as the **training set** and the **test** set. The model is created using the data from the training set. After the model is created, its performance will be evaluated on the test set. The reason for splitting the data in this way is that machine learning models tend to perform better on the data sets on which they were trained than they do when exposed to new data. If we trained the model on a data set, and then evaluated it on the same data, we would likely have an overly optimistic view of how well the model would perform on new observations. By splitting the data, we can hold out a set of observations that are not seen during training. The held-out test data can be used to give as a less biased assessment of how well the model will perform on new data.

Create a markdown cell with a level 2 header that reads "**Part 2: Splitting the Data**".

In the same cell, add unformatted text explaining that we will now be splitting the data into training and test sets.

When splitting data into training and test sets, it is important to either randomly select the rows of the data set that are to be used in each set, or to randomly shuffle the rows before splitting the data set. The reason for this is that data is sometimes recorded in a systematic way. For example, the observations might be sorted according to the values in one of the columns. If you don't randomly select rows and instead just take the first several rows for the training set and the last several rows for the test set, then your two sets might contain records of very different types, neither of which will be representative of the dataset as a whole.

While this is an important consideration, and one we will discuss again later in the course, the records in our dataset have fortunately already been shuffled, so this will not be a concern for us in this project. We will simply use the first 300 records for our training set, and will use the last 98 records for the test set.

Create six new lists as follows:

- Use slicing to split the list **`weight`** into two lists, **`x_train`** and **`x_test`**.
- Use slicing to split the list **`ln_mpg`** into two lists, **`y_train`** and **`y_test`**.
- Use slicing to split the list **`mpg`** into two lists, **`mpg_train`** and **`mpg_test`**.

In each case, use the first 300 values in the original list for the training set and use the last 98 values for the test set.

Create variables **`n_train`** and **`n_test`** by setting them equal to the lengths of **`x_train`** and **`x_test`**, respectively. Use the variables **`n_train`** and **`n_test`** to display the training and test set sizes, along with text output as follows:

```
Training Set Size: xxxx
Test Set Size:     xxxx
```

Create a markdown cell, explaining that we will now create scatter plots to visualize the data in the training and test sets. As before, the code for this will be provided to you.

Copy the code below (without the indentation) into a code cell and then execute that cell.

```
plt.figure(figsize=[12,4])
plt.subplot(1,2,1)
plt.scatter(x_train, y_train, c='skyblue', edgecolor='k')
plt.xlabel('Natural Log of Weight')
plt.ylabel('MPG')
plt.title('Training Set')

plt.subplot(1,2,2)
plt.scatter(x_test, y_test, c='skyblue', edgecolor='k')
plt.xlabel('Natural Log of Weight')
plt.ylabel('MPG')
plt.title('Test Set')
plt.show()
```

Part 3: Descriptive Statistics

In this section, we will calculate the mean and variance for our response variable and our predictor variable. We will use the values calculated in this section to calculate the coefficients $\widehat{\beta_0}$ and $\widehat{\beta_1}$ for our regression model.

Create a markdown cell with a level 2 header that reads "**Part 3: Descriptive Statistics**".

In the same cell, add unformatted text explaining the goal for this part of the project, and that we will start by calculating the mean of the X values (which represent weight), and the mean of the Y values (which represent log-MPG).

The **mean** or **average** of a collection of n numbers is the sum of those numbers divided by n. Let $\bar{x}$ denote the mean of the values $x_1, x_2, \ldots, x_n$ stored in **`x_train`**, and let $\bar{y}$ denote the mean of the values $y_1, y_2, \ldots, y_n$ stored in **`y_train`**. Formulas for $\bar{x}$ and $\bar{y}$ are provided below.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + x_2 + \cdots + x_n)$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{n}(y_1 + y_2 + \cdots + y_n)$$

Calculate the mean of the values in **`x_train`** and **`y_train`**, storing the resulting values in variables named **`mean_x`** and **`mean_y`**. You may use the **`sum()`** function, as well as the variable **`n_train`** that you defined earlier in the notebook.

Print the results with text output as shown below. Round the displayed value of **`mean_x`** to 2 decimal places, and round the displayed value of **`mean_y`** to 4 decimal places. Do not round the actual values stored within the variables themselves, just the values displayed.

```
Mean of X = xxxx
Mean of Y = xxxx
```

We will now work toward calculating the variance of the values of X in the training set as well as the variance of the values of Y in the training set. The variance is a measure of how dispersed, or spread out, observations of a particular variable are. We will calculate the desired variances by first calculating an intermediate result. Specifically, we will calculate the **sum of squared deviations** for X and Y, which are denoted by S_{XX} and S_{YY}, respectively. Formulas for these quantities are provided below. We will use these values again in later parts of this project.

$$S_{XX} = \sum_{i=1}^{n} (x_i - \bar{x})^2 = (x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2$$

$$S_{YY} = \sum_{i=1}^{n} (y_i - \bar{y})^2 = (y_1 - \bar{y})^2 + (y_2 - \bar{y})^2 + \cdots + (y_n - \bar{y})^2$$

Create a markdown cell with unformatted text explaining that we will now be calculating S_{xx} and S_{yy}.

Calculate S_{XX} and S_{YY}, storing the results in variables named **Sxx** and **Syy**. You will need to use a loop to perform this task. For the sake of efficient memory usage, you should not create any additional lists when performing these calculations.

Print the results with text output as shown below. Round the displayed value of **Sxx** to 2 decimal places and round the displayed value of **`mean_y`** to 4 decimal places. Do not round the actual values stored within the variables themselves, just the values displayed.

```
Sxx = xxxx
Syy = xxxx
```

We are now ready to calculate the variance of the training values of X and Y. These values are typically denoted by s_X^2 and s_Y^2, and formulas for their calculation are provided below.

$$s_X^2 = \frac{S_{XX}}{n-1} \quad \text{and} \quad s_Y^2 = \frac{S_{YY}}{n-1}$$

Create a markdown cell with unformatted text explaining that we will now be calculating the variance of the training values of X and Y.

Calculate s_X^2 and s_Y^2, storing the results in variables named **`var_x`** and **`var_y`**. Print the results with text output as shown below. Round the displayed value of **Sxx** to 2 decimal places, and round the displayed value of **`mean_y`** to 4 decimal places. Do not round the actual values stored within the variables themselves, just the values displayed.

```
Variance of X = xxxx
Variance of Y = xxxx
```

Part 4: Linear Regression Model

In this part, we will create our linear regression model by calculating the model coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$. Before we can do so, we have one last quantity to calculate, which we will denote by S_{XY}. The formula for this value is provided below.

$$S_{XY} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = (x_1 - \bar{x})(y_1 - \bar{y}) + (x_2 - \bar{x})(y_2 - \bar{y}) + \cdots + (x_n - \bar{x})(y_n - \bar{y})$$

Create a markdown cell with a level 2 header that reads "**Part 4: Linear Regression Model**".

In the same cell, add unformatted text explaining that in this part we will calculate S_{XY}, which we will then use to find the coefficients for our linear regression model.

Calculate S_{XY}, storing the result in a variable named **Sxy**. You will need to use a loop to perform this task. For the sake of efficient memory usage, you should not create any additional lists when performing this calculation.

Print the result with text output as shown below. Round the displayed value of **Sxy** to 2 decimal places. Do not round the actual values stored within the variables themselves, just the values displayed.

```
Sxy = xxxx
```

We are now ready to calculate the model coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ using values that we have previously calculated. Formulas for these coefficients are provided below.

$$\hat{\beta}_1 = \frac{S_{XY}}{S_{XX}} \quad \text{and} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \cdot \bar{x}$$

Create a markdown cell with unformatted text explaining that we will now be calculating the coefficients of our model.

Calculate the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ storing the results in variables named **`beta_0`** and **`beta_1`**. Print the results with text output as shown below. Round the displayed value of **`beta_0`** to 4 decimal places and round the displayed value of **`beta_1`** to 8 decimal places. Do not round the actual values stored within the variables themselves, just the values displayed.

```
beta_0 = xxxx
beta_1 = xxxx
```

You should find that $\hat{\beta}_0 = 4.1538$ and $\hat{\beta}_1 = -0.00035266$. As a result, the equation of our regression model is as follows:

$$\hat{Y} = 4.1538 - 0.00035266 \cdot X$$

Rewriting this using more descriptive names for the variables, we get the following:

$$\ln\left(\widehat{\text{MPG}}\right) = 4.1538 - 0.00035266 \cdot \text{Weight}$$

This model allows us to approximate the natural logarithm of the MPG for a vehicle, and thus MPG itself, by plugging the vehicles weight in pounds into the equation.

We will close this part of the project by plotting the regression line we have found against the scatter plots for both the training and test sets. You will be provided with the code for these plots.

Create a markdown cell explaining that we will now visualize the regression line by plotting it on top of the scatter plots for the training set as well as for the test set.

Copy the code below (without the indentation) into a code cell and then execute that cell.

```
y_vals = [beta_0 + beta_1 * 1500, beta_0 + beta_1 * 5500]
plt.figure(figsize=[12,4])
plt.subplot(1,2,1)
plt.scatter(x_train, y_train, c='skyblue', edgecolor='k')
plt.plot([1500,5500], y_vals, c='crimson', lw=3)
plt.xlabel('Natural Log of Weight')
plt.ylabel('MPG')
plt.title('Training Set')

plt.subplot(1,2,2)
plt.scatter(x_test, y_test, c='skyblue', edgecolor='k')
plt.plot([1500,5500], y_vals, c='crimson', lw=3)
plt.xlabel('Natural Log of Weight')
plt.ylabel('MPG')
plt.title('Test Set')
plt.show()
```

Part 5: Training Score

Regression models are often scored using the **r-squared** metric, denoted by r^2. This score is a number between 0 and 1, and represents the proportion of the variance in the response variable that has been explained by the model. For example, if $r^2 = 0.9$ for a model, then the model explains 90% of the variability in the values of the response variable, whereas if $r^2 = 0.05$, then the model explains only 5% of the variability in the response.

In this part and the next, we will calculate the model's r-squared score on both the training set and on the test set. We will be primarily interested in the test score, but the training score will be useful for the sake of comparison.

To calculate the r-squared score on a particular set, we must first use the model to find estimated Y values for that set, and then use the observed Y values to calculate the amount of error involved in each prediction. We will now walk through the steps involved in this process.

Let $x_1, x_2, \ldots, x_n$ denote the predictor values in either the training or test set. For each predictor value, we can calculate an estimated response value, denoted by $\hat{y}_i$, using the formula below:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 \cdot x_i$$

Let $y_1, y_2, \ldots, y_n$ denote the true response values for the same set, and let $\hat{\varepsilon}_1, \hat{\varepsilon}_2, \ldots, \hat{\varepsilon}_n$ denote the errors involved in each of our predictions. These error values are also called **residuals**. We can calculate each of the residuals as follows:

$$\hat{\varepsilon}_i = y_i - \hat{y}_i$$

We will now calculate the estimated response values, $\hat{y}_i$.

Create a markdown cell with a level 2 header that reads "**Part 5: Training Score**".

In the same cell, add unformatted text explaining that in this section, we will be calculating the training r-squared score, and that we will start by calculating estimated response values for the training set.

Use the variables **`beta_0`** and **`beta_1`** as well as the list **`x_train`** to calculate the estimated response values for the training set. Store the results in a list named **`pred_y_train`**. You can accomplish this task using either a loop or a list comprehension. This cell should not produce any output.

Create a markdown cell explaining that we will now calculate the residuals for the training set.

Use the values stored in **y_train** and **pred_y_train** to calculate the residuals for the training set. Store the results in a list named **`error_y_train`**. You can accomplish this task using either a loop or a list comprehension. This cell should not produce any output.

Before continuing on with our calculation of the r-squared score, we will display the true Y values, the predicted Y values, and the residual for each of the first 10 observations in the training set.

Create a markdown cell explaining that we will be displaying the values mentioned above.

Use a code cell to print the first 10 values of each of the lists **`y_train`**, **`pred_y_train`**, and **`error_y_train`**. Format the output as follows:

- The values from each list should be arranged in columns, with each row of output corresponding to a single training observation.
- The output should include column headers and a dividing line, as shown below.
- The number of characters reserved for each column, in order, should be 6, 10, and 10.
- The columns should be right aligned.
- The values displayed should all be rounded to 4 decimal places.

The first few rows of your output should look exactly as shown below:

```
True y    Pred y     Error
-------------------------
3.3032    3.0289    0.2743
3.4904    3.4538    0.0366
```

Before calculating the r-squared score, we must calculate one more intermediate value, named the **sum of squared errors**, and denoted by SSE**.** The formula for a model's SSE on a given set is provided below.

$$SSE = \sum_{i=1}^{n} \hat{\varepsilon}_i^2 = \hat{\varepsilon}_1^2 + \hat{\varepsilon}_2^2 + \cdots + \hat{\varepsilon}_n^2$$

Create a markdown cell explaining that we will now calculate the sum of squared errors score for the training set.

Use the values stored in the list **`error_y_train`** to calculate the training sum of squared errors score, storing the result in a variable name **`sse_train`**. Display the result with text output as shown below, rounding the displayed value to 4 decimal places.

```
Training SSE = xxxx
```

We are now ready to calculate the r-squared score for the training set. The formula for this metric is provided below:

$$r^2 = 1 - \frac{SSE}{S_{YY}}$$

Create a markdown cell explaining that we will now calculate the r-squared score for the training set.

Calculate the training r-squared score, storing the result in a variable named **`r2_train`**. Display the result with text output as shown below, rounding the displayed value to 4 decimal places.

```
Training r-Squared = xxxx
```

Part 6: Test Score

We will now calculate the testing r-squared score for our model by repeating the steps from Part 5, but using the test set rather than the training set.

Create a markdown cell with a level 2 header that reads "**Part 6: Test Score**".

In the same cell, add unformatted text explaining that in this section, we will be calculating the test r-squared score, and that we will start by calculating estimated response values for the test set.

Use the variables **`beta_0`** and **`beta_1`** as well as the list **`x_test`** to calculate the estimated response values for the test set. Store the results in a list named **`pred_y_test`**. You can accomplish this task using either a loop or a list comprehension. This cell should not produce any output.

Create a markdown cell explaining that we will now calculate the residuals for the test set.

Use the values stored in **`y_test`** and **`pred_y_test`** to calculate the residuals for the test set. Store the results in a list named **`error_y_test`**. You can accomplish this task using either a loop or a list comprehension. This cell should not produce any output.

Before continuing on with our calculation of the r-squared score, we will display the true Y values, the predicted Y values, and the residual for each of the first 10 observations in the test set.

Create a markdown cell explaining that we will be displaying the values mentioned above.

Use a code cell to print the first 10 values of each of the lists **`y_test`**, **`pred_y_test`**, and **`error_y_test`**. The output should be formatted in the same way as what was requested for the training set in Part 5.

Create a markdown cell explaining that we will now calculate the sum of squared errors score for the test set.

Use the values stored in the list **`error_y_test`** to calculate the testing sum of squared errors score, storing the result in a variable name **`sse_test`**. Display the result with text output as shown below, rounding the displayed value to 4 decimal places.

```
Test SSE = xxxx
```

Before calculating the test r-squared score, we must first calculate the S_{YY} value for the test set. We calculated this for the training set in Part 3, but have not done so for the test set.

Create a markdown cell explaining that we will now calculate the value of S_{YY} on the test set, and will then use that and the test sum of squared errors to calculate the test r-squared score.

Use the formula from Part 3 to calculate the value S_{YY} for the test set, storing the results in a variable named **`Syy_test`**.

Then calculate the test r-squared score, storing the result in a variable named **`r2_test`**. Display the result with text output as shown below, rounding the displayed value to 4 decimal places.

```
Test r-Squared = xxxx
```

Create a markdown cell explaining that we will now create a plot to visualize the errors for the observations in the test set.

Copy the code below (without the indentation) into a code cell and then execute that cell.

```
plt.figure(figsize=[8,4])
plt.scatter(x_test, y_test, c='skyblue', edgecolor='k')
plt.plot([1500,5250], [beta_0 + beta_1 * 1500, beta_0 + beta_1 * 5250],
         c='crimson', lw=3)
for i in range(n_test):
    plt.plot([x_test[i], x_test[i]], [pred_y_test[i], y_test[i]],
             c='coral', zorder=0)
plt.show()
```

Each vertical shown in this plot represents the estimation error, or residual, for a single observation in the test set.

Part 7: Transforming Test Predictions

Recall that our model does not directly estimate values for the average MPG of a vehicle, but instead estimates values for the natural log of the average MPG. However, once we have an estimate for the natural logarithm of MPG, we can easily exponentiate that to get an estimate for the MPG. In particular, if $\hat{y}_i$ is an estimate for the natural log of the average MPG for an observation, we can calculate the estimated average MPG for that observation as follows:

$$MPG_i = e^{\hat{y}_i}$$

We will conclude this project by transforming the values we estimated for the natural log of MPG for observations in the test set into estimates for the average MPG.

Create a markdown cell with a level 2 header that reads "**Part 7: Transforming Test Predictions**".

In the same cell, add unformatted text explaining that we will be calculating estimates for the average MPG for observations in our test set.

Create a variable named **`e`**, setting its value to 2.718281828. Use this variable along with the values stored in **`pred_y_test`** to calculate estimates for the average MPG for each observation in the test set. Store the results in a variable named **`pred_mpg_test`**.

Next, we will calculate the error involved in the MPG estimates for each of the test observations. This is accomplished by subtracting the estimated MPG from the true MPG.

Create a markdown cell explaining that we will now calculate the error in each estimate for the average MPG.

Use the values stored in **mpg_test** and **pred_mpg_test** to calculate the error for the test set. Store the results in a list named **`error_mpg_test`**. You can accomplish this task using either a loop or a list comprehension. This cell should not produce any output.

We will now display the results from the previous two code cells in a tabular format.

Create a markdown cell explaining that we will now display the true MPG, the estimated MPG, and the estimation error for each of the first 10 observations in the test set.

Use a code cell to print the first 10 values of each of the lists **`mpg_test`**, **`pred_mpg_test`**, and **`error_mpg_test`**. Format the output as follows:

- The values from each list should be arranged in columns, with each row of output corresponding to a single training observation.
- The output should include column headers and a dividing line, as shown below.
- The number of characters reserved for each column, in order, should be 8, 12, and 9.
- The columns should be right aligned.
- The values displayed should all be rounded to 1 decimal place.

The first few rows of your output should look exactly as shown below:

```
True MPG    Pred MPG    Error
-----------------------------
    14.0        21.4     -7.4
    18.0        16.8      1.2
```

Submission Instructions

When you are done, click **Kernel > Restart and Run All**. If any cell produces an error, then manually run every cell after that one, in order. Save your notebook, and then export the notebook as an HTML file. Upload the HTML file to Canvas and upload the IPYNB file to CoCalc, placing the file into the folder **Projects/Project 01**.