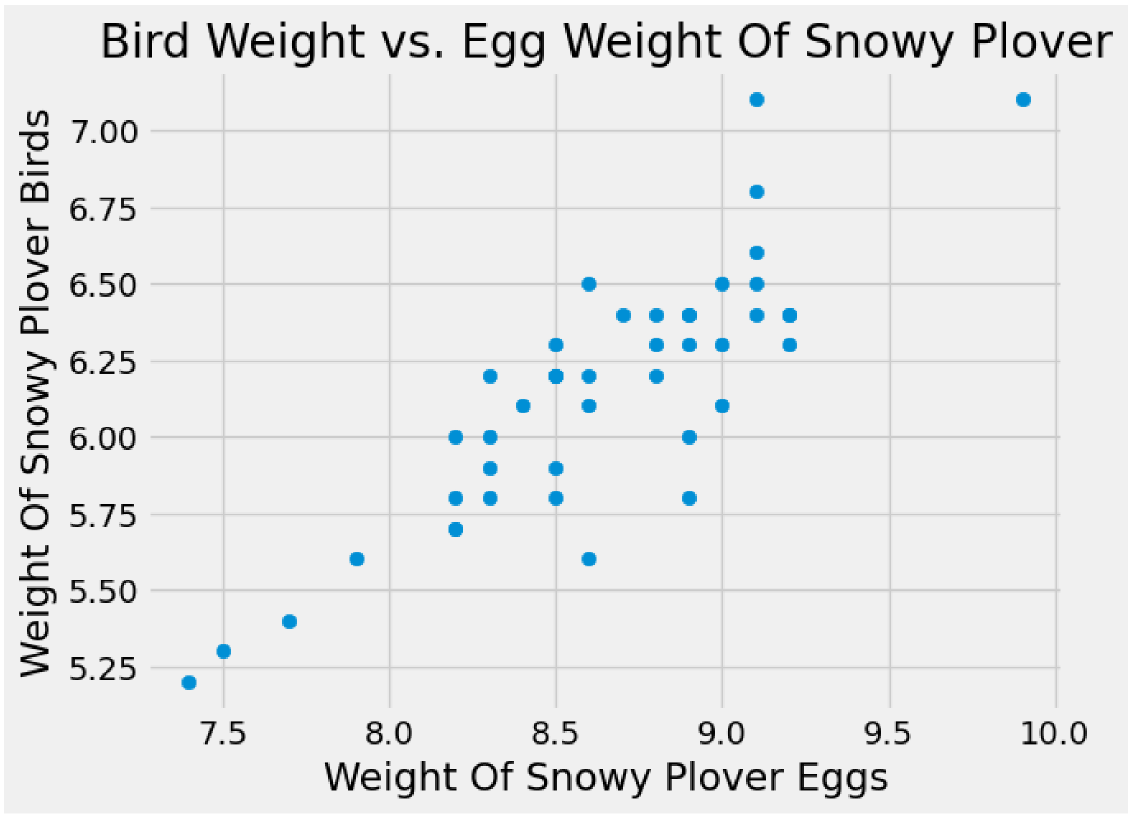
Question 1.1.

a). Create a scatter plot of the egg weights (on the x-axis) vs the bird weights (on the y-axis). Label your axes and give your plot a title.

b). Based only on your plot, make a guess as to what the correlation is between these two variables and assign it to the variable `corr_guess` (don't do any actual calculations yet, just guess based on your visual inspection of the plot).

```
In [4]: plt.scatter(birds['Egg Weight'], birds['Bird Weight'])
        plt.xlabel('Weight Of Snowy Plover Eggs')
        plt.ylabel('Weight Of Snowy Plover Birds')
        plt.title('Bird Weight vs. Egg Weight Of Snowy Plover')
        plt.show()
        # Your code for part (a) above this line
```



```
In [5]: corr_guess = 0.75
```

Question 1.4

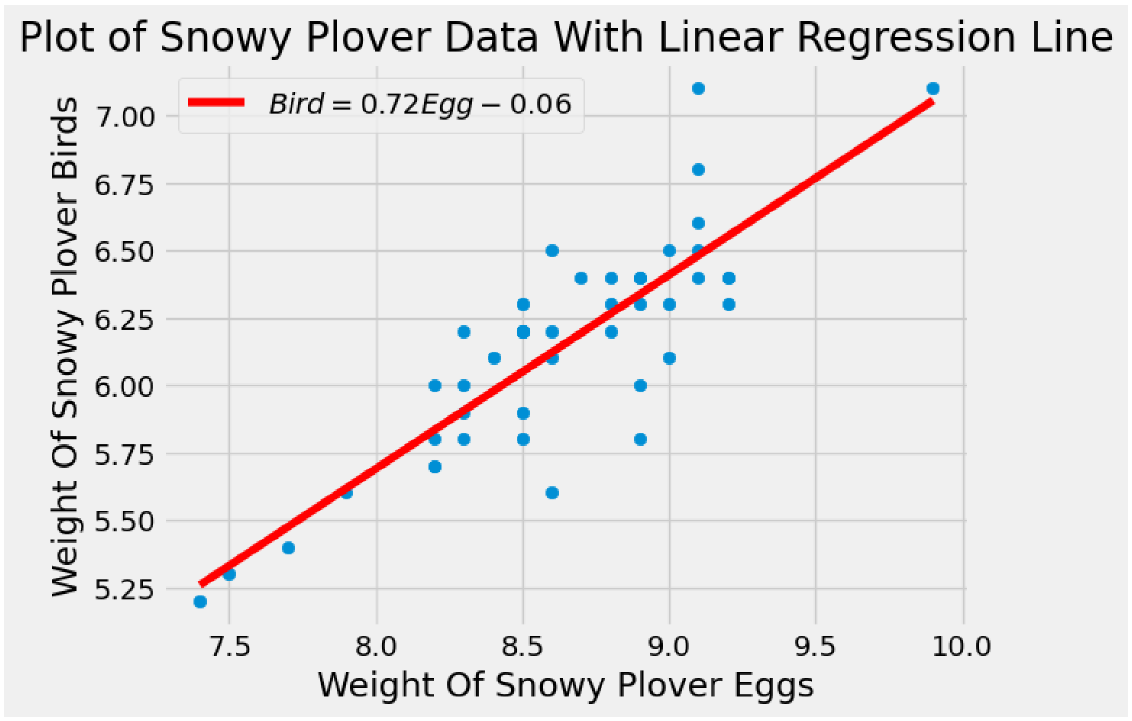
Part a).

- Run `fit_line` on the `birds` table.
- Then create a scatterplot of the birds data with an overlaid plot of the least squares linear regression line (using the output of your `fit_line` function).
- For credit on this problem you must use the output of your `fit_line` function to create the line yourself, to demonstrate you understand how this line is created.
- Label your axes and create a label for the line on the plot that gives the equation of the line. Tip: including the following code in your plt.plot function will add a label with the equation of the best fit line: `label =r'$Bird = {0:.2f}Egg{1:.2f}$'.format(slope, intercept)`

Part b). Based on the slope from your least squares regression line model, we see there is a positive linear association in this sample of data between Snowy Plover egg weight and bird weight. Can we conclude from this that there is also a positive linear association between egg weight and bird weight in the population of Snowy Plover birds? Why or why not? Explain your reasoning.

Because there is a positive linear association between the egg weight and actual weight of the bird, we can say there is a positive linear association for the **sample** but not necessarily for the entire population of snowy polvers. In short, we can say it is true for the sample but not for the entire population. We would require other metrics and a larger sample size to further infer about the population. So no, at this point we cannot say there is a positive linear association between egg weight and bird weight for the **entire population** of snowy plovers.

```
In [10]: slope_int = fit_line(birds, 'Egg Weight', 'Bird Weight')
         x_min = birds['Egg Weight'].min()
         x_max = birds['Egg Weight'].max()
         x = np.linspace(x_min, x_max, 100)
         plt.scatter(birds['Egg Weight'], birds['Bird Weight'])
         plt.plot(x, slope_int[0] * x + slope_int[1], linestyle = 'solid', color = 'red', label=r'$Bird
         plt.xlabel('Weight Of Snowy Plover Eggs')
         plt.ylabel('Weight Of Snowy Plover Birds')
         plt.title('Plot of Snowy Plover Data With Linear Regression Line')
         plt.legend()
         plt.show()
         # Your code for part a above this line
```


Plot of Snowy Plover Data With Linear Regression Line
$Bird = 0.72Egg - 0.06$
Weight Of Snowy Plover Birds
Weight Of Snowy Plover Eggs
7.00
6.75
6.50
6.25
6.00
5.75
5.50
5.25
7.5
8.0
8.5
9.0
9.5
10.0

Question 2.2.

a). Create a *numpy array* called `resampled_slopes` that contains the slope of the best fit line for 10,000 bootstrap resamples of `birds`. (Hint, use your function `fit_line` from question 1).

b). Then create a 95% Confidence Interval for the true value of the slopes and assign the lower and upper values to the variables `CI_lower` and `CI_upper` respectively.

c). Create a plot with a histogram of the density distribution of these slopes AND the confidence interval overlaid on the bottom of the distribution (similar to histograms you made in HW 10).

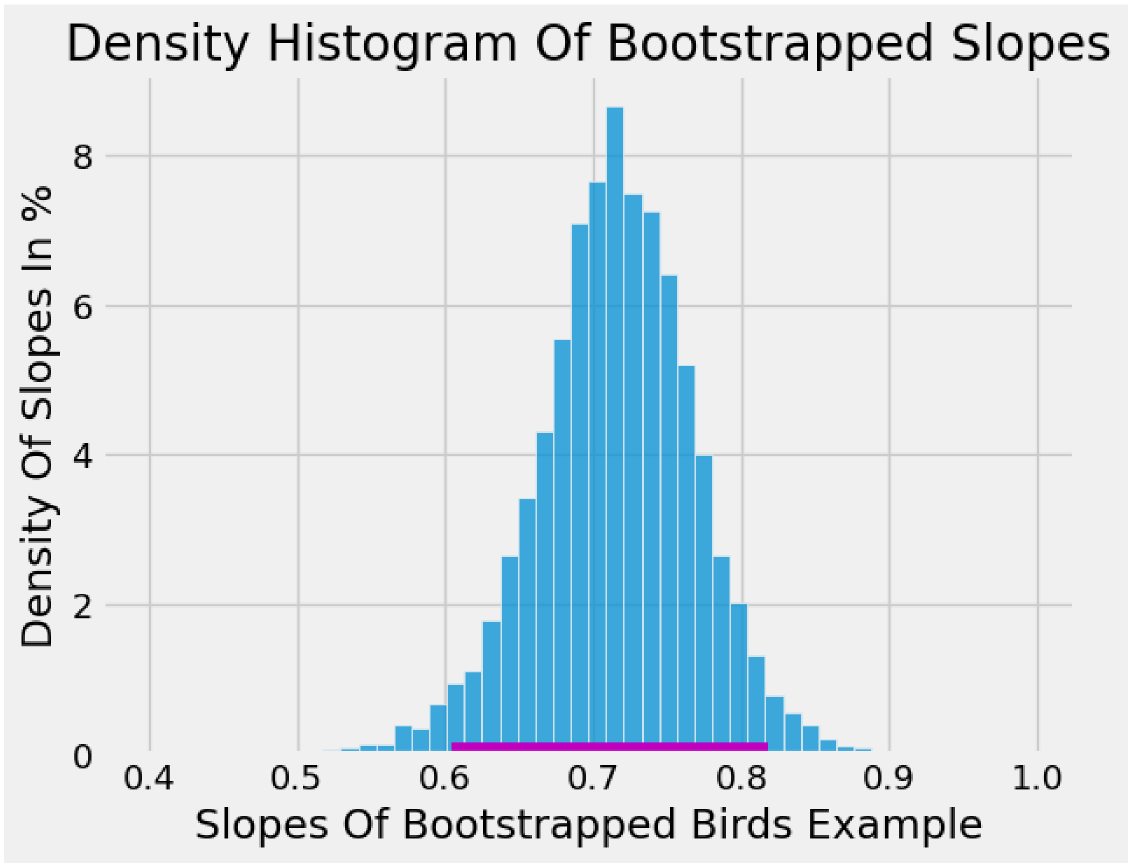
```
In [13]: resampled_slopes = np.zeros(10000)
         for i in range(len(resampled_slopes)):
             duplicate = birds.sample(frac = 1, replace = True)
             slope = fit_line(duplicate, 'Egg Weight', 'Bird Weight')[0]
             resampled_slopes[i] = slope
         # your code for part (a) above

         CI_lower = np.percentile(resampled_slopes, 2.5)
         CI_upper = np.percentile(resampled_slopes, 97.5)

         print("95% confidence interval for slope: [{:g}, {:g}]".format(CI_lower, CI_upper))

         sns.histplot(resampled_slopes, bins = 50, stat = 'density')
         plt.plot(np.array([CI_lower, CI_upper]), np.array([0, 0]), c='m', lw=10)
         plt.xlabel('Slopes Of Bootstrapped Birds Example')
         plt.ylabel('Density Of Slopes In %')
         plt.title('Density Histogram Of Bootstrapped Slopes')
         plt.show()
         # your code for part (c) above
```

```
95% confidence interval for slope: [0.604806, 0.817864]
```



```
In [14]: grader.check("q2_2")
```

```
Out[14]: q2_2 results: All test cases passed!
```

Question 2.3. Based on your confidence interval, would you accept or reject the null hypothesis that the true slope is 0? Explain your reasoning. What p-value cutoff are you using?

With a confidence interval of 95% and using the formula $(100 - q)$ where q is the confidence interval, this would lead to a p value of 0.05. From observing the bootstrapped slopes in this context, we can see that hypothesis parameter 0 is not in our confidence interval of $[\approx 0.6, \approx 0.8]$. Because of this, the data is more consistent with the **alternative** hypothesis and we would **reject the null** hypothesis based on this evidence.

Question 2.7 Define the function `bootstrap_lines`. It takes in four arguments: 1. `df`: a dataframe like `birds` 2. `x_col`: the name of our x-column within the input `tbl` 3. `y_col`: the name of our y-column within the input `tbl` 4. `num_bootstraps`: an integer, a number of bootstraps to run.

It returns a *dataframe* with one row for each bootstrap resample and the following two columns: 1. `Slope`: the bootstrapped slopes 2. `Intercept`: the corresponding bootstrapped intercepts

(Hint, use your function from the previous part of this question)

Then call this function 10,000 times using the bird data.

```
In [22]: def bootstrap_lines(df, x_col, y_col, num_bootstraps):
             cols = ["Slope", "Intercept"]
             ret = pd.DataFrame(np.nan, index = range(num_bootstraps), columns=cols)
             for i in range(num_bootstraps):
                 resample = compute_resampled_line(df, x_col, y_col)
                 ret.loc[i] = [resample[0], resample[1]]
             return ret


         regression_lines = bootstrap_lines(birds, "Egg Weight", "Bird Weight", 10000)
         regression_lines.head()
```

```
Out[22]:       Slope  Intercept
         0  0.730301  -0.193024
         1  0.795006  -0.689385
         2  0.697291   0.100338
         3  0.728826  -0.179460
         4  0.720929  -0.085510
```

Question 2.8.

a). Create a *numpy array* called `predictions_for_eight` that contains the predicted bird weights based on an egg of weight 8 grams for each regression line in `regression_lines` (from Question 2.7).

b). Then create a 95% Confidence Interval for the true value of the prediction for a weight of 8 grams and assign the lower and upper values to the variables `CI_lower_pred` and `CI_upper_pred` respectively.

c). Create a plot with a histogram of the density distribution of these predictions AND the confidence interval overlaid on the bottom of the distribution. Label your axes on the plot.

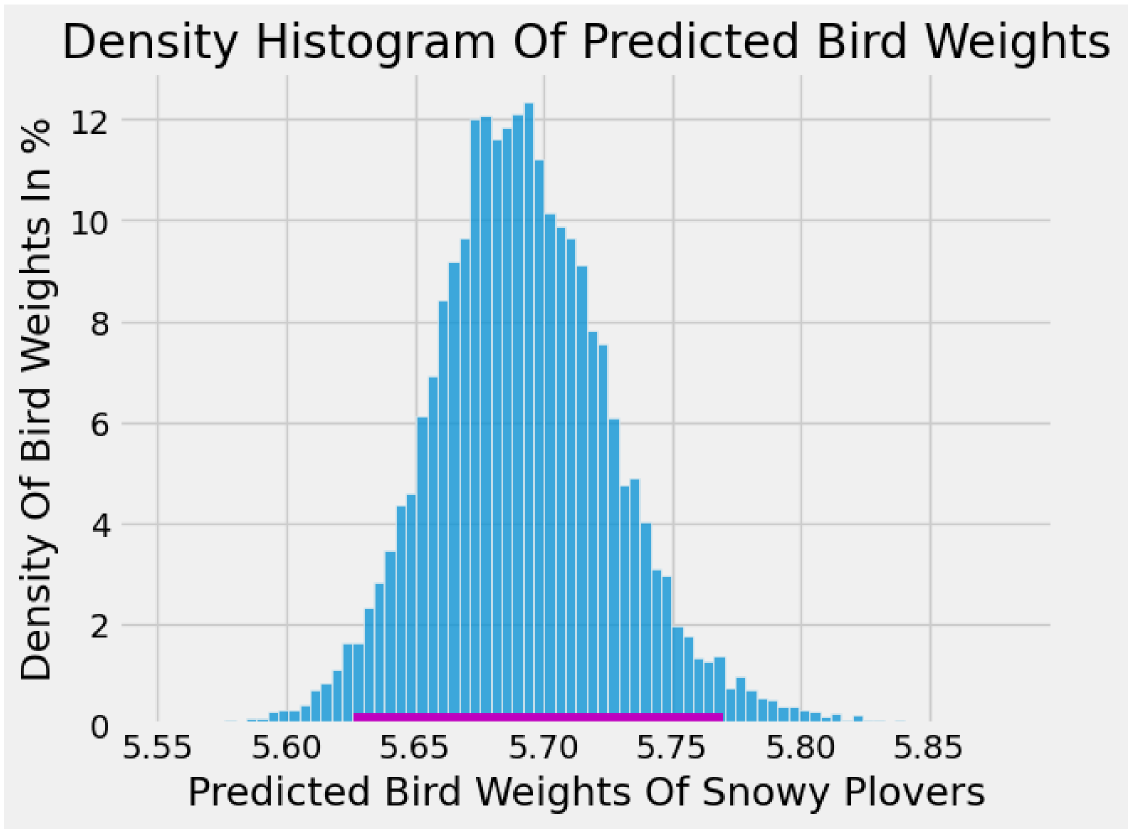
```
In [23]: predictions_for_eight = np.zeros(10000)
         for index, row in regression_lines.iterrows():
             slope = row['Slope']
             inter = row['Intercept']
             prediction = slope * 8 + inter
             predictions_for_eight[index] = prediction
         # your code for part (a) above

         CI_lower_pred = np.percentile(predictions_for_eight, 2.5)
         CI_upper_pred = np.percentile(predictions_for_eight, 97.5)

         print("95% confidence interval for slope: [{:g}, {:g}]".format(CI_lower_pred, CI_upper_pred))


         sns.histplot(predictions_for_eight, stat = 'density')
         plt.plot(np.array([CI_lower_pred, CI_upper_pred]), np.array([0, 0]), c='m', lw=10)
         plt.xlabel('Predicted Bird Weights Of Snowy Plovers')
         plt.ylabel('Density Of Bird Weights In %')
         plt.title('Density Histogram Of Predicted Bird Weights')
         plt.show()
         # your code for part (c) above
```

```
95% confidence interval for slope: [5.62596, 5.76972]
```



```
In [24]: grader.check("q2_8")
```

```
Out[24]: q2_8 results: All test cases passed!
```

Question 3.1. We'll start by building a simple linear regression model to try and predict `mpg` using `horsepower`.

a). Use your function `fit_line` from question to fit this linear model (i.e. find the slope and intercept for the least squares regression line).

b). Then use seaborn `lmplot` to plot a scatterplot of horsepower (on the x-axis) vs mpg (on the y-axis) with the least squares linear regression line and prediction intervals included on the plot.

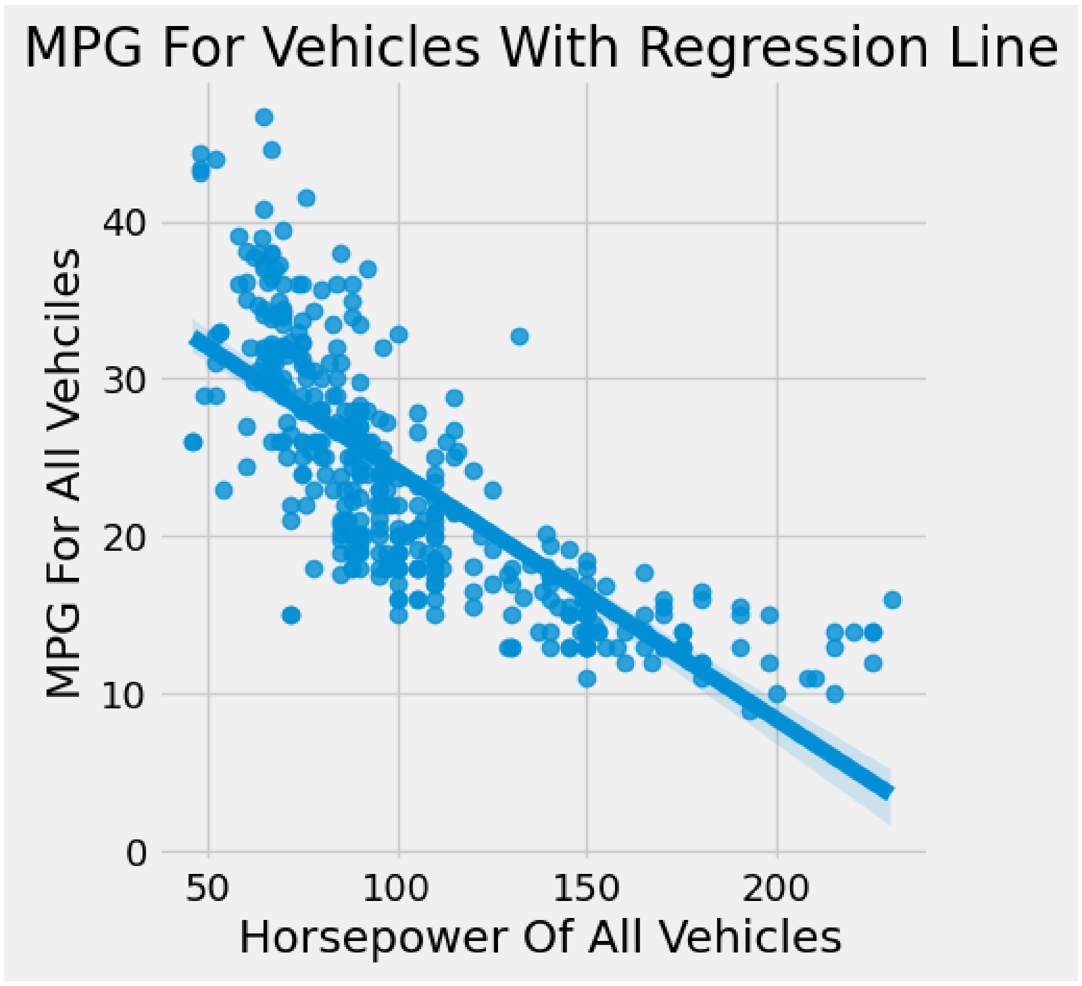
```
In [30]: slope, intercept = fit_line(vehicle_data, "horsepower", "mpg")

         print("Our model slope is ", slope, "and the intercept is", intercept)


         sns.lmplot(x = 'horsepower', y = 'mpg', data = vehicle_data);
         plt.ylabel("MPG For All Vehciles")
         plt.xlabel("Horsepower Of All Vehicles")
         plt.title("MPG For Vehicles With Regression Line")
         # Your code for part b above this line
```

```
Our model slope is  -0.15784473335365365 and the intercept is 39.93586102117047
```

```
Out[30]: Text(0.5, 1.0, 'MPG For Vehicles With Regression Line')
```


MPG For Vehicles With Regression Line
MPG For All Vehciles
Horsepower Of All Vehicles
0
10
20
30
40
50
100
150
200

Question 3.3. One way we check model goodness of fit is to analyze the residuals.

For each of the (horsepower, mpg) data points given below, calculate the residual. Give your answer as a **float value**.

Hint: You can use `my_model.predict()` https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegr

Note that the input and the output of this method are numpy arrays, and the test below requires a single float as your answer.

Data point A: (52, 44)

Data point B: (150, 11)

```
In [38]: res_A = 44 - float(my_model.predict(np.array([[52]])))

         res_B = 11 - float(my_model.predict(np.array([[150]])))

         print(res_A)
         print(res_B)
```

```
12.272065113219512
-5.259151018122417
```

```
In [39]: grader.check("q3_3")
```

```
Out[39]: q3_3 results: All test cases passed!
```

Question 3.4. Do these residual plots indicate that our linear model is a good model for the data? Why or why not?

Based on the plots of the residuals, there are two common patterns that are occurring. For instance, in the residuals vs. mpg plot, the residuals are not randomly spread because most of the residuals occur at a lower horse power. The same can then be said for the residuals vs. predicted mpg plot, but the residuals are largely spread around higher fitted values. Because the residuals are not randomly spread in these plots, it would be safe to say that our linear model does not suffice as a prediction for mpg. We may need to include other features to accurately predict mpg.

Question 4.2

Let's start by trying Option 1 in the list above.

a). Add a new column to vehicle_data called `sqrt(hp)` that contains the square root of the horsepower data.

b). Then plot a scatterplot of `mpg` vs `sqrt(hp)` to visually inspect if this transformation makes the data appear more linear than our first model. Label your axes.

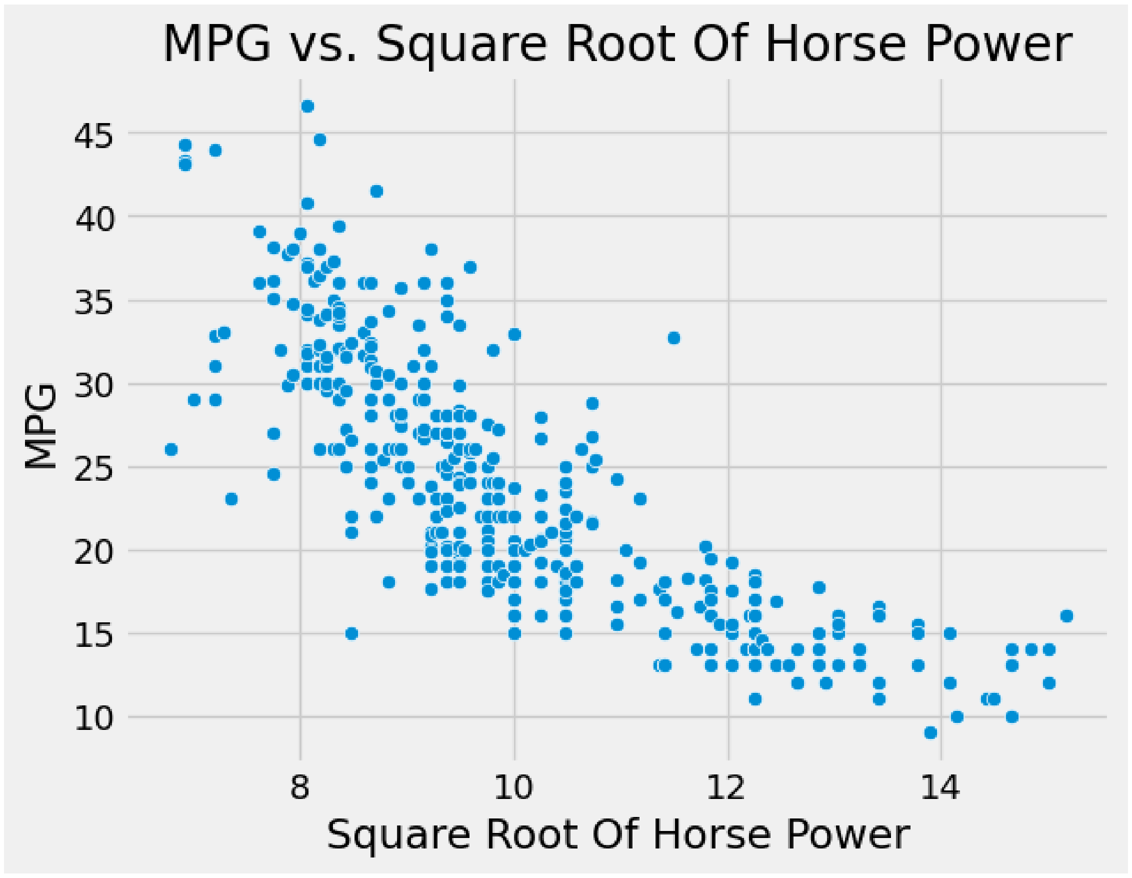
```
In [43]: sqrt_hp = np.sqrt(vehicle_data['horsepower'].astype(float).to_numpy())
         vehicle_data.insert(4, "sqrt(hp)", sqrt_hp)
         # your code for part(a) above


         vehicle_data.head()
```

```
Out[43]:       mpg  cylinders  displacement  horsepower  sqrt(hp)  weight  \
         19   26.0          4          97.0        46.0  6.782330    1835
         102  26.0          4          97.0        46.0  6.782330    1950
         326  43.4          4          90.0        48.0  6.928203    2335
         325  44.3          4          90.0        48.0  6.928203    2085
         244  43.1          4          90.0        48.0  6.928203    1985

              acceleration  model_year  origin                           name
         19           20.5          70  europe     volkswagen 1131 deluxe sedan
         102          21.0          73  europe          volkswagen super beetle
         326          23.7          80  europe               vw dasher (diesel)
         325          21.7          80  europe              vw rabbit c (diesel)
         244          21.5          78  europe  volkswagen rabbit custom diesel
```

```
In [44]: sns.scatterplot(vehicle_data, x = vehicle_data['sqrt(hp)'], y = vehicle_data['mpg']);
         plt.xlabel("Square Root Of Horse Power")
         plt.ylabel("MPG")
         plt.title("MPG vs. Square Root Of Horse Power")
         plt.show()
         # your code for part(b) above
```



```
In [45]: grader.check("q4_2")
```

```
Out[45]: q4_2 results: All test cases passed!
```

In the cell below, explain why we use the term “linear” to describe the model above, even though it incorporates a square root function as a feature.

Although this model involves a polynomial where the variable in $y = mx + b$ is not to the first power, this model is still considered to be linear because linear in this context refers to the slope and intercept precisely being of the first power. We could substitute $\sqrt{\text{horsepower}}$ with z and it would look familiar to us in the context of what is considered liner and what is not. In short, the expression is still linear in the slope and intercept.

Question 4.4.

a). Use `sklearn` to create and fit this new model.

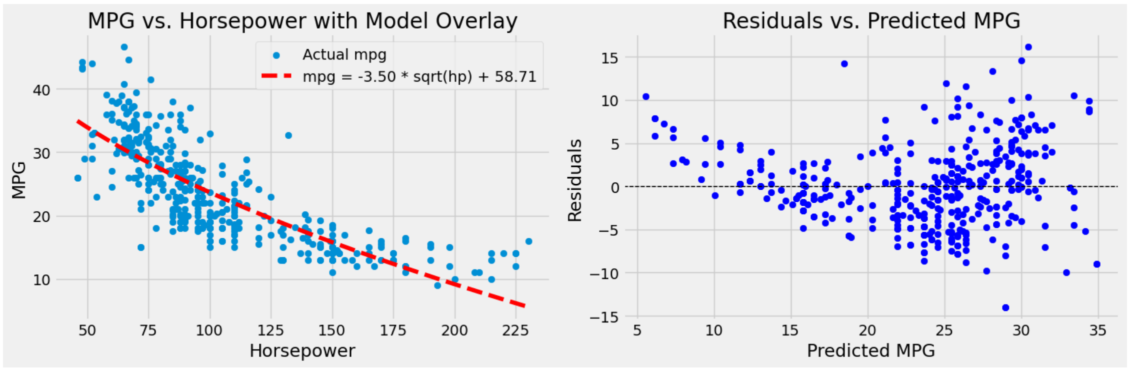
b. Once you've created the new model, set `predicted_mpg_hp_sqrt` to the predicted mpg for the data.

c). Make 2 side-by-side plots:
- A plot of this new model overlaid with a scatterplot of the original data. Include the equation for the new model as a label on your plot (see the plots in question 3 for reference on how to code this). - A plot of the residuals vs the **predicted values of mpg**

d). Calculate the RMSE for this model

```
In [46]: sqrt_hp_model = lm.LinearRegression()
         x_sqrt_hp = vehicle_data[["sqrt(hp)"]]
         y_sqrt_hp = vehicle_data["mpg"]
         sqrt_hp_model.fit(x_sqrt_hp, y_sqrt_hp)
         sqrt_hp_b = sqrt_hp_model.intercept_
         sqrt_hp_m = sqrt_hp_model.coef_[0]
         predicted_mpg_sqrt_hp = sqrt_hp_model.predict(vehicle_data[["sqrt(hp)"]])
         # Your code for parts a) and b) above this line
```

```
In [57]: residuals_mpg_sqrt_hp = vehicle_data['mpg'] - predicted_mpg_sqrt_hp
         fig, ax = plt.subplots(1, 2, figsize=(15, 5))
         ax[0].scatter(vehicle_data['horsepower'], vehicle_data['mpg'], label='Actual mpg')
         ax[0].plot(vehicle_data['horsepower'], predicted_mpg_sqrt_hp, 'r--', label=f'mpg = {sqrt_hp_m:
         ax[0].set_xlabel('Horsepower')
         ax[0].set_ylabel('MPG')
         ax[0].set_title('MPG vs. Horsepower with Model Overlay')
         ax[0].legend()
         ax[1].scatter(predicted_mpg_sqrt_hp, residuals_mpg_sqrt_hp, label='Residuals', color='blue')
         ax[1].axhline(0, color='black', linestyle='--', linewidth=1)
         ax[1].set_xlabel('Predicted MPG')
         ax[1].set_ylabel('Residuals')
         ax[1].set_title('Residuals vs. Predicted MPG')
         plt.tight_layout()
         plt.show()
         # Your code for part c  above this line
```



```
In [48]: real_mpg = vehicle_data['mpg'].astype(float).to_numpy()
         differences = real_mpg - predicted_mpg_sqrt_hp
         squared_diff = np.square(differences)
         mean_squared = np.mean(squared_diff)
         RMSE_hp_sqrt = np.sqrt(mean_squared)

         print("The RMSE of this model is ", RMSE_hp_sqrt)
```

```
The RMSE of this model is  4.652907260805868
```

```
In [49]: grader.check("q4_4")
```

```
Out[49]: q4_4 results: All test cases passed!
```

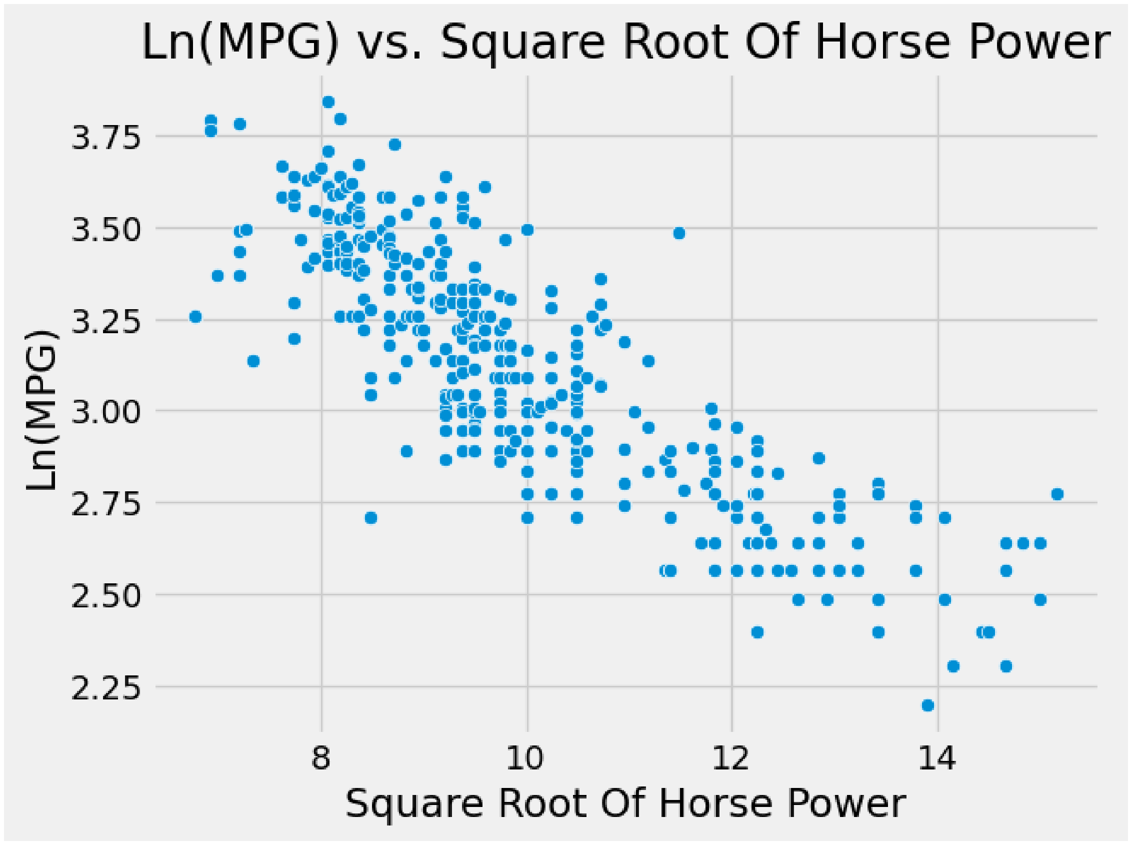
Question 4.5. Analyze the new model compared to the original one. Which RMSE is smaller? Does the residual plot of this new model indicate this model is a good choice? Why or why not?

The RMSE of the first model was roughly 4.89, where as the RMSE of this new model is 4.65. So, this newer model has a slightly smaller RMSE. The residual plot indicates that this model is slightly better than the previous model but it is still probably not as optimal as we would like it to be. This is because the residuals are still not randomly scattered. So there still exists a probable better model out there for this.

a). Add a new column to vehicle_data called `log(mpg)` that contains the that contains the log of the mpg data.

b). Then plot a scatterplot of `log(mpg)` vs `sqrt(hp)` to visually inspect if this transformation makes the data appear more linear than our first and second models. Label your axes.

```
In [50]: log_mpg = np.log(vehicle_data['mpg'].astype(float).to_numpy())
         vehicle_data.insert(1, "log(mpg)", log_mpg)
         sns.scatterplot(vehicle_data, x = vehicle_data['sqrt(hp)'], y = vehicle_data['log(mpg)']);
         plt.xlabel("Square Root Of Horse Power")
         plt.ylabel("Ln(MPG)")
         plt.title("Ln(MPG) vs. Square Root Of Horse Power")
         plt.show()
```



Question 4.7.

a). Use `sklearn` to create and fit this new model.

b). Since the model will output predicted values in units of log(mp), you will need to transform it back to compare to our original data.

Notice:

$$\log(\text{mpg}) = \theta_0 + \theta_1 \sqrt{\text{horsepower}}$$

$$\implies mpg = e^{\theta_0 + \theta_1 \sqrt{\text{horsepower}}}$$

Let `predicted_mpg_model3` equal the output from the transformed equation: $mpg = e^{\theta_0 + \theta_1 \sqrt{\text{horsepower}}}$

c). Make 2 side-by-side plots:
- A plot of this new model overlaid with a scatterplot of the original data. Include the equation for the new model as a label on your plot (see the plots in question 3 for reference on how to code this). - A plot of the residuals vs the **predicted values of mpg** (`predicted_mpg_model3`)

‘ d). Calculate the RMSE for this model

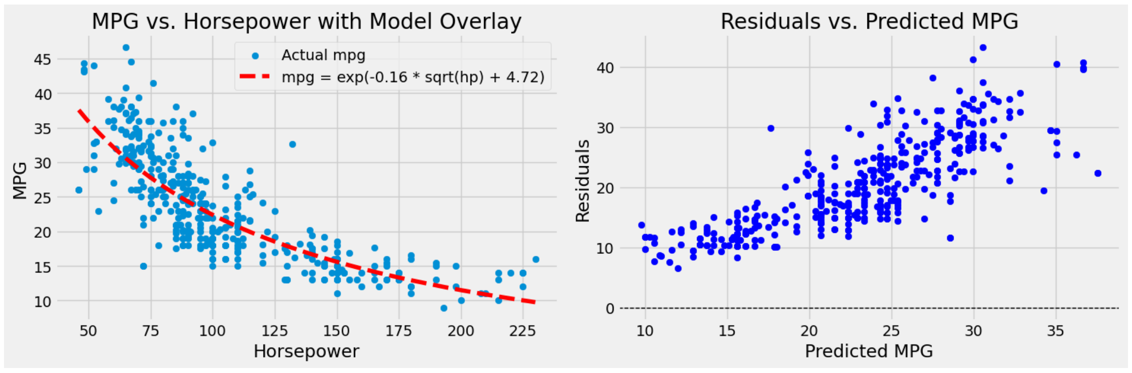
```
In [62]: log_mpg_model = lm.LinearRegression()
         x_log_mpg = vehicle_data[["sqrt(hp)"]]
         y_log_mpg = vehicle_data["log(mpg)"]
         log_mpg_model.fit(x_log_mpg, y_log_mpg)
         log_mpg_b = log_mpg_model.intercept_
         log_mpg_m = log_mpg_model.coef_[0]
         predicted_log_mpg = log_mpg_model.predict(x_log_mpg)
         predicted_mpg_model3 = np.exp(predicted_log_mpg)
         # Your code for parts a) and b) above this line
```

```
4.716232955186357
```

```
In [64]: residuals_log_mpg = vehicle_data['mpg'] - predicted_log_mpg
         fig, ax = plt.subplots(1, 2, figsize=(15, 5))
         ax[0].scatter(vehicle_data['horsepower'], vehicle_data['mpg'], label='Actual mpg')
         ax[0].plot(vehicle_data['horsepower'], predicted_mpg_model3, 'r--', label=f'mpg = exp({log_mpg
         ax[0].set_xlabel('Horsepower')
         ax[0].set_ylabel('MPG')
         ax[0].set_title('MPG vs. Horsepower with Model Overlay')
```

```
ax[0].legend()
ax[1].scatter(predicted_mpg_model3, residuals_log_mpg, label='Residuals', color='blue')
ax[1].axhline(0, color='black', linestyle='--', linewidth=1)
ax[1].set_xlabel('Predicted MPG')
ax[1].set_ylabel('Residuals')
ax[1].set_title('Residuals vs. Predicted MPG')
plt.tight_layout()
plt.show()

# Your code for part c  above this line
```



```
In [65]: real_mpg = vehicle_data['mpg'].astype(float).to_numpy()
         differences = real_mpg - predicted_mpg_model3
         squared_diff = np.square(differences)
         mean_squared = np.mean(squared_diff)
         RMSE_model3 = np.sqrt(mean_squared)

         print("The RMSE of this model is ", RMSE_model3)
```

```
The RMSE of this model is  4.462359658070859
```

```
In [66]: grader.check("q4_7")
```

```
Out[66]: q4_7 results: All test cases passed!
```

Question 4.8. Analyze this new model compared to the first 2 ones. Which RMSE is smaller? Does the residual plot of this new model indicate this model is a better choice than the first two? Why or why not?

This model has the smallest RMSE of roughly 4.46. This residual plot indicates that it is a better choice than the previous two models because of the spread of the residuals compared to the last two. It still has a pattern but this pattern is more what we would expect of a better performing model.