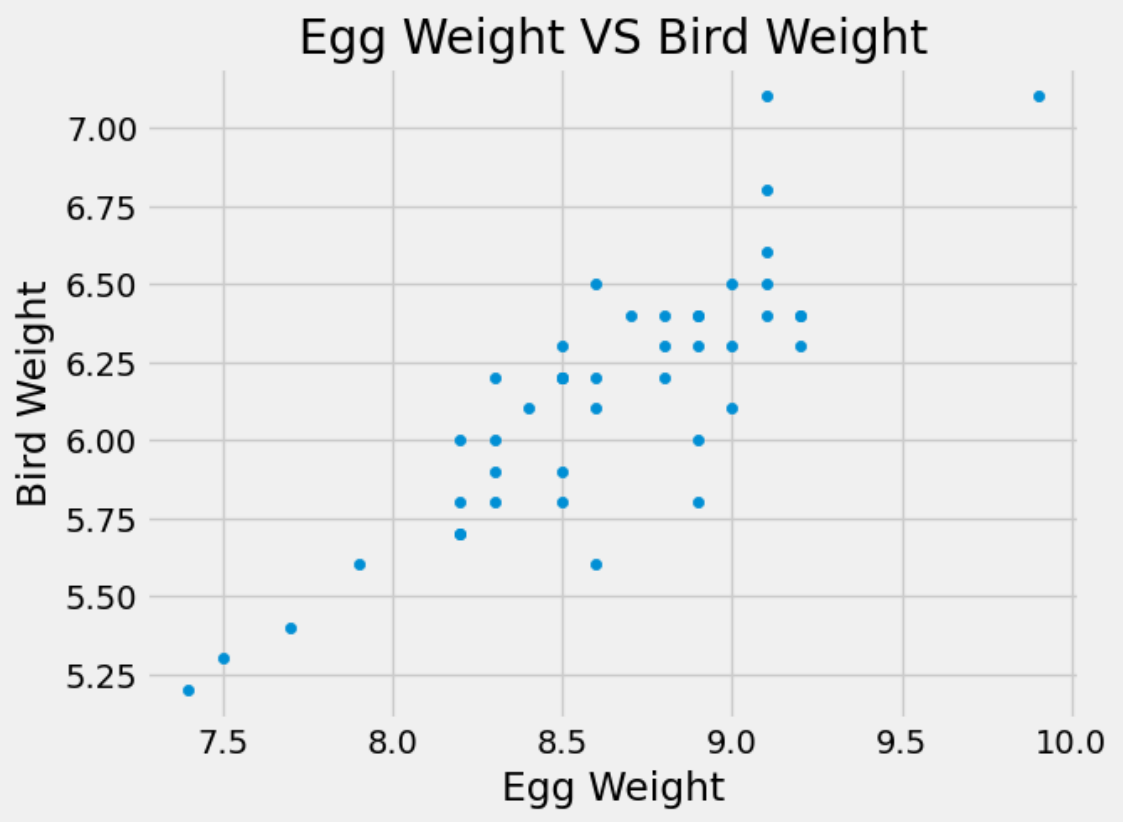
Question 1.1.

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]: egg_bird_weights = birds.drop(columns=["Egg Length", "Egg Breadth"])
        egg_bird_weights.plot.scatter(x="Egg Weight", y= "Bird Weight")
        plt.title("Egg Weight VS Bird Weight")
        # Your code for part (a) above this line
```

```
Out[4]: Text(0.5, 1.0, 'Egg Weight VS Bird Weight')
```




```
In [5]: corr_guess = .7
```

Question 1.4

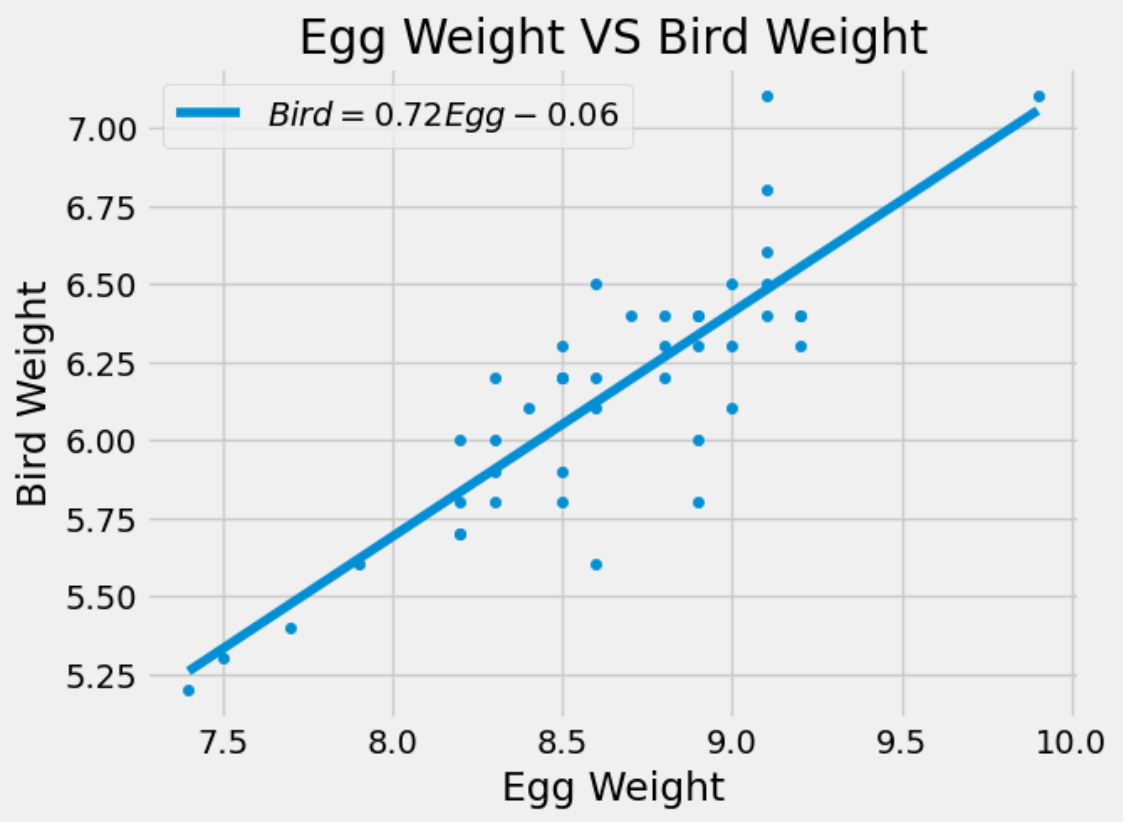
Part a).

- 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.

Part b answer cell:
We cannot conclude that there is also a positive linear association between egg weight and bird weight in the population of Snowy Plover birds because the model created only shows association for our sample which is smaller than the population size. Thus, the actual population's scatter plot might look different from our sample's scatter plot. The positive linear association is just a prediction, but with each sample the prediction might be different because of variability. So just from one sample we cannot assume the trend is true. We would need to bootstrap and do confidence intervals.

```
In [10]: line = fit_line(birds, "Egg Weight", "Bird Weight")
         min_x = min(birds["Egg Weight"])
         max_x = max(birds["Egg Weight"])
         min_y = line[0]*min_x + line[1]
         max_y = line[0]*max_x + line[1]
         egg_bird_weights = birds.drop(columns=["Egg Length", "Egg Breadth"])
         egg_bird_weights.plot.scatter(x="Egg Weight", y= "Bird Weight")
         plt.plot([min_x,max_x], [min_y, max_y], label=r'$Bird = {0:.2f}Egg{1:.2f}$'\
                  .format(line[0], line[1]))
         plt.legend()
         plt.title("Egg Weight VS Bird Weight")




         # Your code for part a above this line


Out[10]: Text(0.5, 1.0, 'Egg Weight VS Bird Weight')
```


Egg Weight VS Bird Weight
$Bird = 0.72Egg - 0.06$
Bird Weight
Egg Weight
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

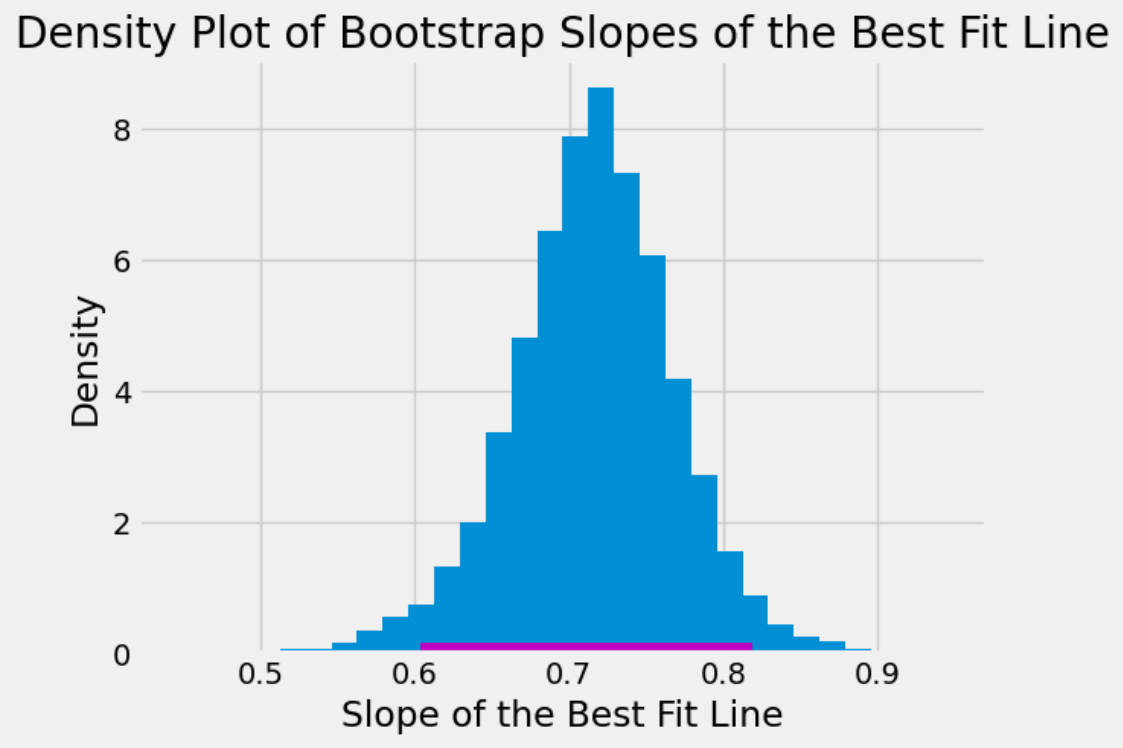
Question 2.2.

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.array([])
         for i in range(10000):
             resample = birds.sample(frac=1, replace = True)
             line = fit_line(resample, "Egg Weight", "Bird Weight")
             resampled_slopes = np.append(resampled_slopes, line[0])
         # 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))


         plt.hist(resampled_slopes, bins = 30, density = True)
         plt.plot(np.array([CI_lower, CI_upper]), np.array([0, 0]), c='m', lw=10)
         plt.title("Density Plot of Bootstrap Slopes of the Best Fit Line")
         plt.ylabel("Density")
         plt.xlabel("Slope of the Best Fit Line")
         # your code for part (c) above


95% confidence interval for slope: [0.603201, 0.818919]



Out[13]: Text(0.5, 0, 'Slope of the Best Fit Line')
```



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?

Based on the confidence interval, we would reject the null hypothesis that the true slope is 0 because 0 is not in the confidence interval of [.607328, .819571]. The p-value cutoff I am using is .05 because I am using a 95% confidence interval.

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):
             slopes = np.array([])
             intercepts = np.array([])
             for i in range(num_bootstraps):
                 line = compute_resampled_line(df, x_col, y_col)
                 slopes = np.append(slopes, line[0])
                 intercepts = np.append(intercepts, line[1])
             bootstrapped = pd.DataFrame(list(zip(slopes, intercepts)), columns=['Slope', 'Intercept'])
             return bootstrapped

         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).

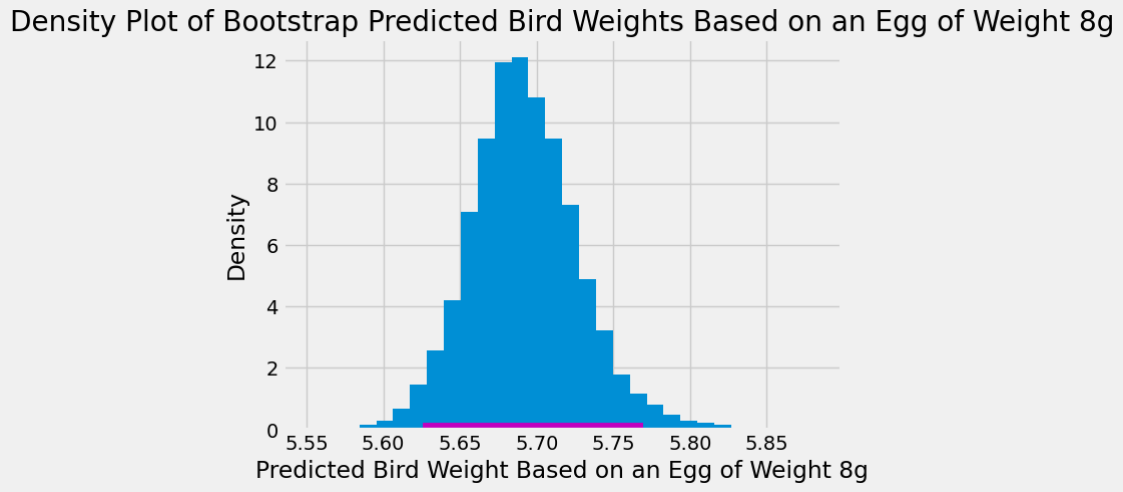
b). Then create a 95% Confidence Interval for the true value of the prediction for a weight of 8 grams and assign the lower and upper values to the variables `CI_lower_pred` and `CI_upper_pred` respectively.

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.array([])

for index, row in regression_lines.iterrows():
    predictions_for_eight = np.append(predictions_for_eight, (row["Slope"]*8)+row["Intercept"])

print(predictions_for_eight)
# 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))


plt.hist(predictions_for_eight, bins = 30, density = True)
plt.plot(np.array([CI_lower_pred, CI_upper_pred]), np.array([0, 0]), c='m', lw=10)
plt.title("Density Plot of Bootstrap Predicted Bird Weights Based on an Egg of Weight 8g")
plt.ylabel("Density")
plt.xlabel("Predicted Bird Weight Based on an Egg of Weight 8g")
# your code for part (c) above
```

```
[5.64938142 5.67066667 5.67866482 … 5.71062939 5.72980436 5.71779115]
95% confidence interval for slope: [5.62596, 5.76972]
```

Out[23]:
```
Text(0.5, 0, 'Predicted Bird Weight Based on an Egg of Weight 8g')
```




```
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`.

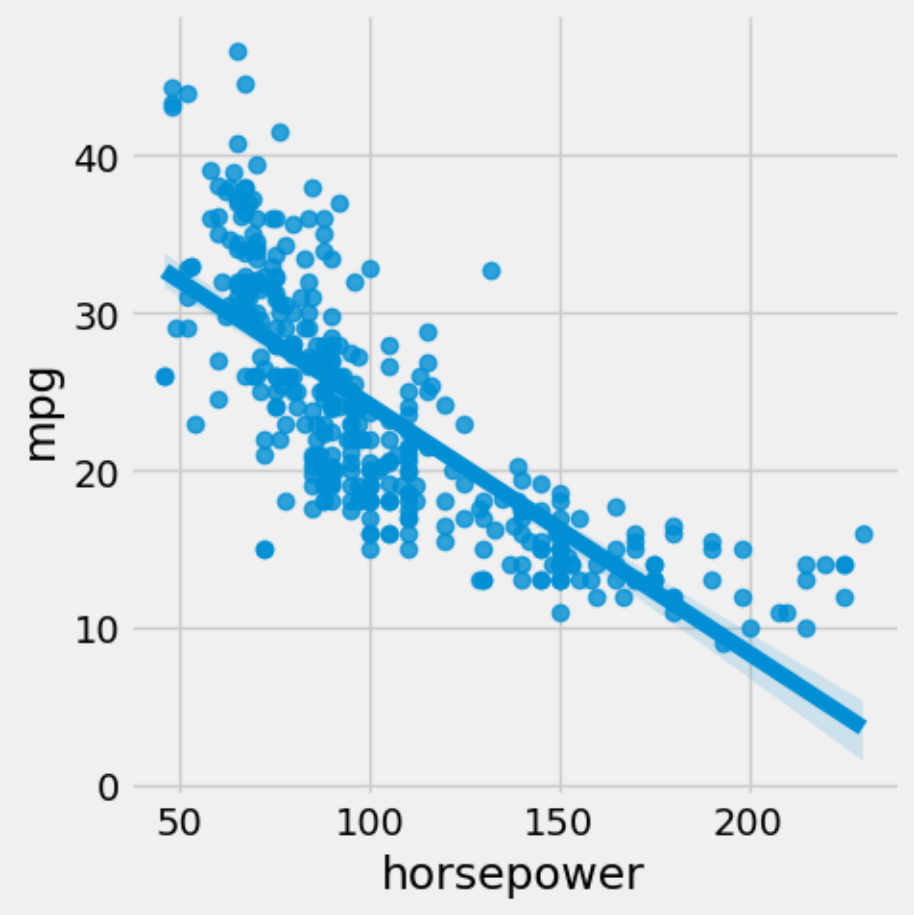
a). Use your function `fit_line` from question to fit this linear model (i.e. find the slope and intercept for the least squares regression line).

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]: line = fit_line(vehicle_data, "horsepower", "mpg")
         slope = line[0]
         intercept = line[1]

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


         sns.lmplot(data=vehicle_data, x="horsepower", y="mpg")
         # Your code for part b above this line
```

```
Our model slope is  -0.15784473335365343 and the intercept is 39.935861021170446
```

```
Out[30]: <seaborn.axisgrid.FacetGrid at 0x7fedd7612f80>
```


40
30
20
10
0
mpg
50
100
150
200
horsepower

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 - my_model.predict([[52]])[0]

         res_B = 11 - my_model.predict([[150]])[0]

         print(res_A)
         print(res_B)
```

```
12.272065113219519
-5.259151018122424
```

```
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?

These residual plots indicate that our linear model is not a good model for the data because they do not look random. The Residuals vs mpg is clumped together on the left and has a pattern. The Residuals vs predicted mpg is clumped together on the right and has a pattern.

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.

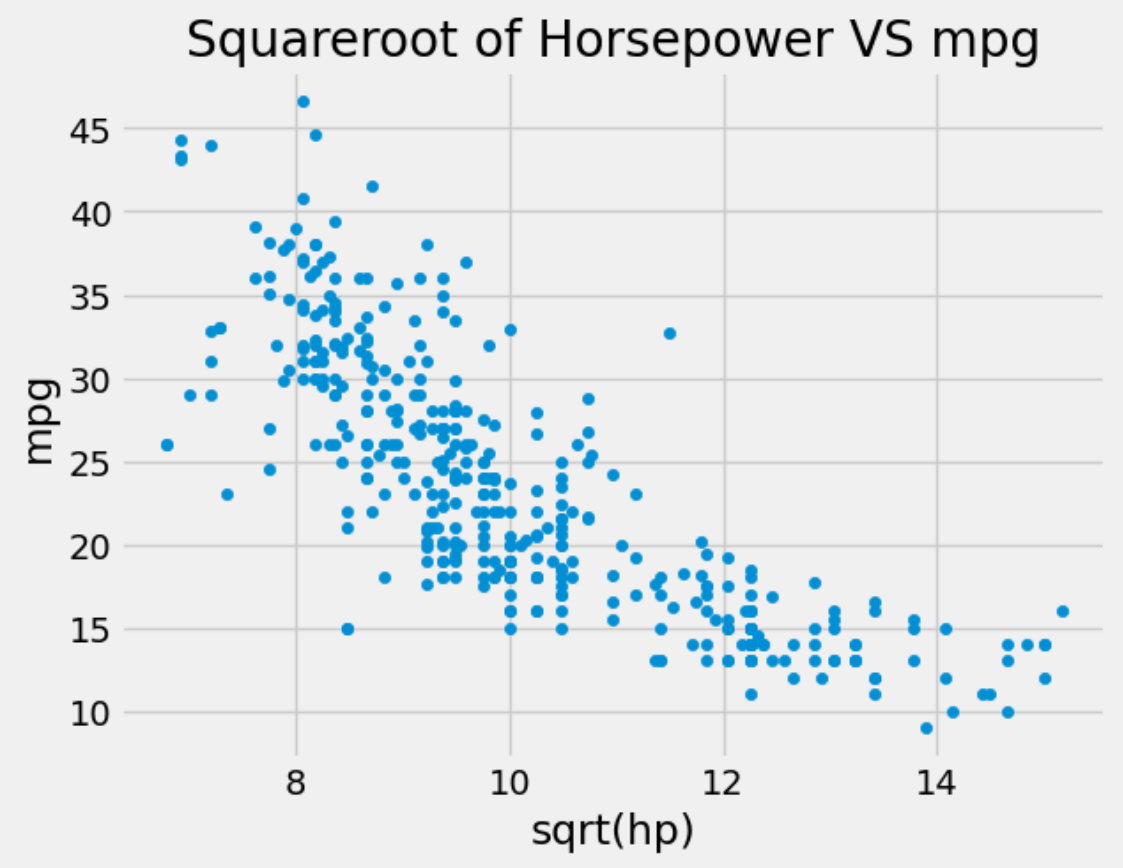
```
In [43]: vehicle_data["sqrt(hp)"] = np.sqrt(vehicle_data["horsepower"])
         # your code for part(a) above


         vehicle_data.head()
```

Out[43]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name	sqrt(hp)
19	26.0	4	97.0	46.0	1835	20.5	70	europe	volkswagen 1131 deluxe sedan	6.782330
102	26.0	4	97.0	46.0	1950	21.0	73	europe	volkswagen super beetle	6.782330
326	43.4	4	90.0	48.0	2335	23.7	80	europe	vw dasher (diesel)	6.928203
325	44.3	4	90.0	48.0	2085	21.7	80	europe	vw rabbit c (diesel)	6.928203
244	43.1	4	90.0	48.0	1985	21.5	78	europe	volkswagen rabbit custom diesel	6.928203

```
In [44]: hp_mpg = vehicle_data.drop(columns=["cylinders", "displacement", "horsepower", "weight", \
                                            "acceleration", "model_year", "origin", "name"])
         hp_mpg.plot.scatter(x="sqrt(hp)", y= "mpg")
         plt.title("Squareroot of Horsepower VS mpg")
         # your code for part(b) above
```

```
Out[44]: Text(0.5, 1.0, 'Squareroot of Horsepower VS mpg')
```



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.

Even though the model incorporates a square root function, the model is described as linear because the square root function is linearizing the model. θ_0 is the intercept, θ_1 is the slope, and $\sqrt{\text{horsepower}}$ is the x.

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.

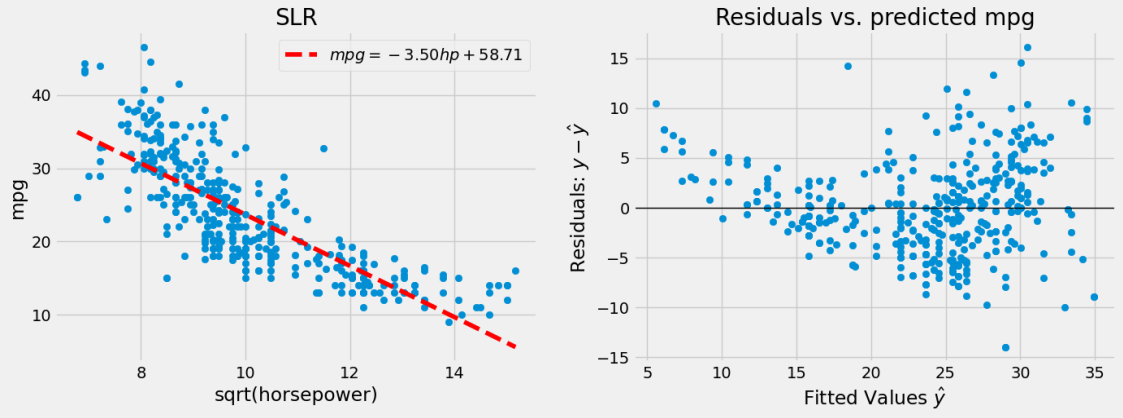
c). Make 2 side-by-side plots:
- A plot of this new model overlaid with a scatterplot of the original data. Include the equation for the new model as a label on your plot (see the plots in question 3 for reference on how to code this). - A plot of the residuals vs the **predicted values of mpg**

d). Calculate the RMSE for this model

```
In [46]: vehicle_model = lm.LinearRegression()
         X = hp_mpg[["sqrt(hp)"]]
         y = hp_mpg["mpg"]
         vehicle_model.fit(X, y)
         predicted_mpg_hp_sqrt = vehicle_model.predict(hp_mpg[["sqrt(hp)"]])
         # Your code for parts a) and b) above this line
```

```
In [47]: fig, ax = plt.subplots(1,2, figsize=(15, 5))

         ax[0].scatter(hp_mpg['sqrt(hp)'], hp_mpg['mpg'])
         ax[0].plot(hp_mpg['sqrt(hp)'], predicted_mpg_hp_sqrt, 'r--', label =r'$mpg = {0:.2f}hp+{1:.2f}$
                    .format(vehicle_model.coef_[0],vehicle_model.intercept_))
         ax[0].set_xlabel('sqrt(horsepower)')
         ax[0].set_ylabel('mpg')
         ax[0].set_title('SLR')
         ax[0].legend()

         ax[1].scatter(predicted_mpg_hp_sqrt, hp_mpg['mpg'] - predicted_mpg_hp_sqrt)
         ax[1].axhline(0, c='black', linewidth=1)
         ax[1].set_xlabel(r'Fitted Values $\hat{y}$')
         ax[1].set_ylabel(r'Residuals: $y - \hat{y}$');
         ax[1].set_title("Residuals vs. predicted mpg")
         # Your code for part c  above this line
```

```
Out[47]: Text(0.5, 1.0, 'Residuals vs. predicted mpg')
```




```
In [48]: RMSE_hp_sqrt = np.sqrt(np.mean((vehicle_data["mpg"]-predicted_mpg_hp_sqrt)**2))
         print("The RMSE of this model is ", RMSE_hp_sqrt)
```

```
The RMSE of this model is  4.6529072608058675
```

```
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?

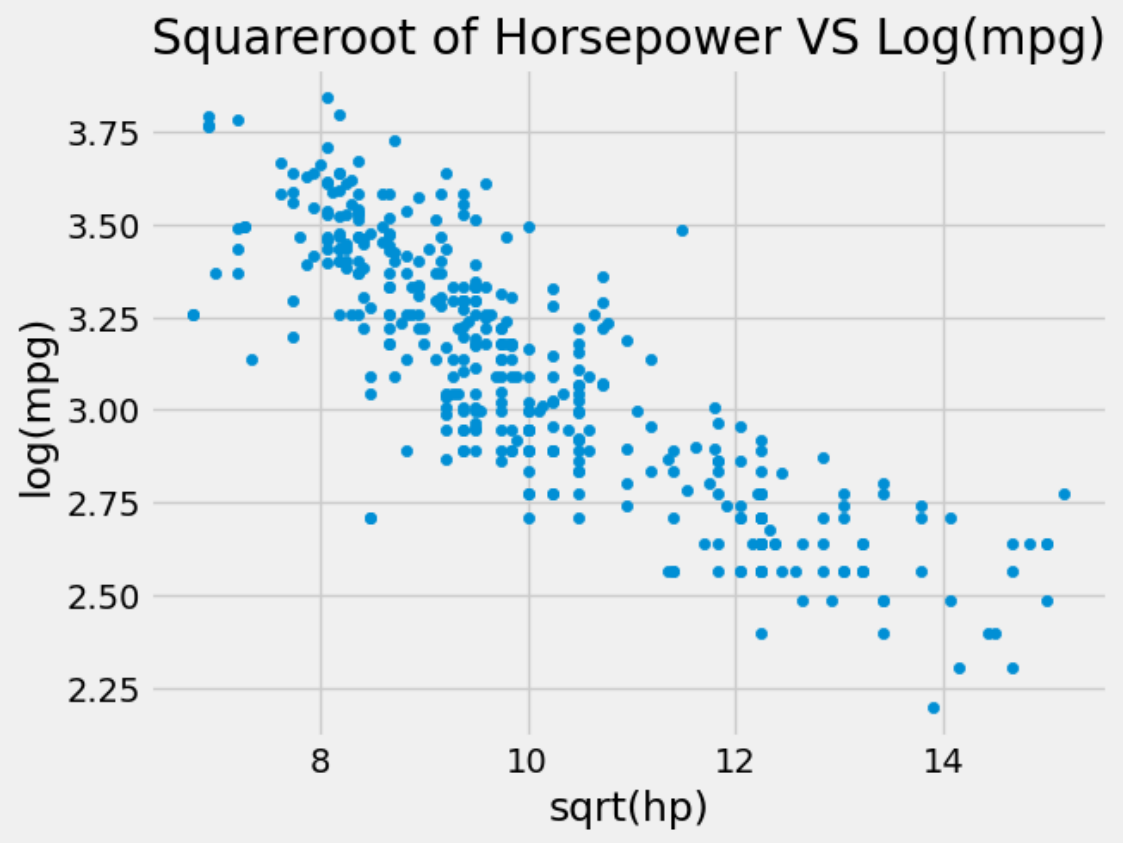
The new model has a smaller RMSE compared to the original one. However, the new residual plot indicates that the new model is still not a good fit because there is still a pattern to the points.

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

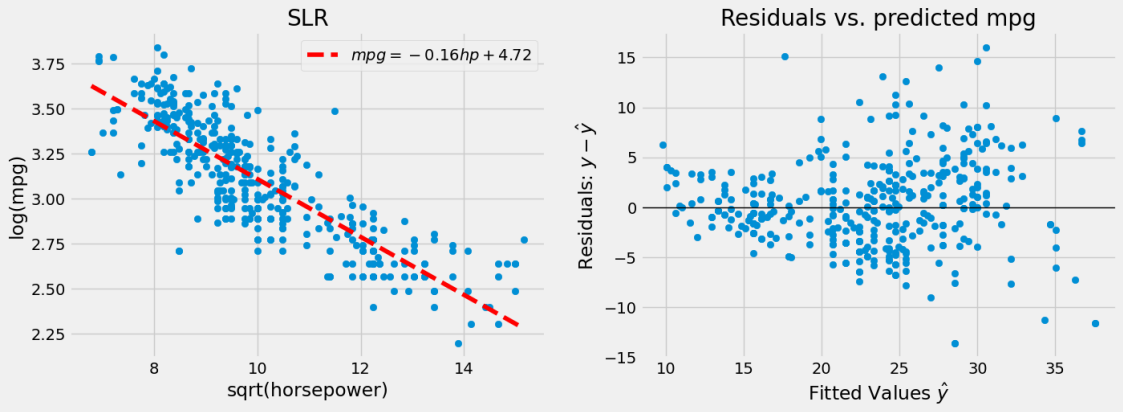
```
In [50]: vehicle_data["log(mpg)"] = np.log(vehicle_data["mpg"])
         vehicle_data.plot.scatter(x="sqrt(hp)", y= "log(mpg)")
         plt.title("Squareroot of Horsepower VS Log(mpg)")
```

```
Out[50]: Text(0.5, 1.0, 'Squareroot of Horsepower VS Log(mpg)')
```



Question 4.7:

```
In [51]: transformed_model = lm.LinearRegression()
         X = vehicle_data[["sqrt(hp)"]]
         y = vehicle_data["log(mpg)"]
         transformed_model.fit(X, y)
         predicted_mpg_model3 = transformed_model.predict(vehicle_data[["sqrt(hp)"]])
         # Your code for parts a) and b) above this line


In [52]: fig, ax = plt.subplots(1,2, figsize=(15, 5))

         # vehicle_data["log(mpg)"] = np.log(vehicle_data["mpg"])
         # vehicle_data.plot.scatter(x="sqrt(hp)", y= "log(mpg)")
         # plt.title("Squareroot of Horsepower VS Log(mpg)")

         ax[0].scatter(vehicle_data["sqrt(hp)"], vehicle_data["log(mpg)"])
         ax[0].plot(vehicle_data['sqrt(hp)'], predicted_mpg_model3, 'r--', label =r'$mpg = {0:.2f}hp+{1
                    .format(transformed_model.coef_[0],transformed_model.intercept_))
         ax[0].set_xlabel('sqrt(horsepower)')
         ax[0].set_ylabel('log(mpg)')
         ax[0].set_title('SLR')
         ax[0].legend()

         ax[1].scatter(np.exp(predicted_mpg_model3), vehicle_data['mpg'] - np.exp(predicted_mpg_model3))
         ax[1].axhline(0, c='black', linewidth=1)
         ax[1].set_xlabel(r'Fitted Values $\hat{y}$')
         ax[1].set_ylabel(r'Residuals: $y - \hat{y}$');
         ax[1].set_title("Residuals vs. predicted mpg")

         # Your code for part c  above this line


Out[52]: Text(0.5, 1.0, 'Residuals vs. predicted mpg')
```



In [53]:
```
RMSE_model3 = np.sqrt(np.mean((vehicle_data["mpg"]-np.exp(predicted_mpg_model3))**2))
print("The RMSE of this model is ", RMSE_model3)
```

```
The RMSE of this model is  4.462359658070861
```

In [54]:
```
grader.check("q4_7")
```

Out[54]:
```
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?

Compared to the first 2 models, model 3 has a smaller RMSE. The residual plot of the new model indicates it is a better choice than the first two because it is looking more spread out randomly. However, this still isn't a very good model because the residual plot is still fanning out the right.