Project 02 Instructions

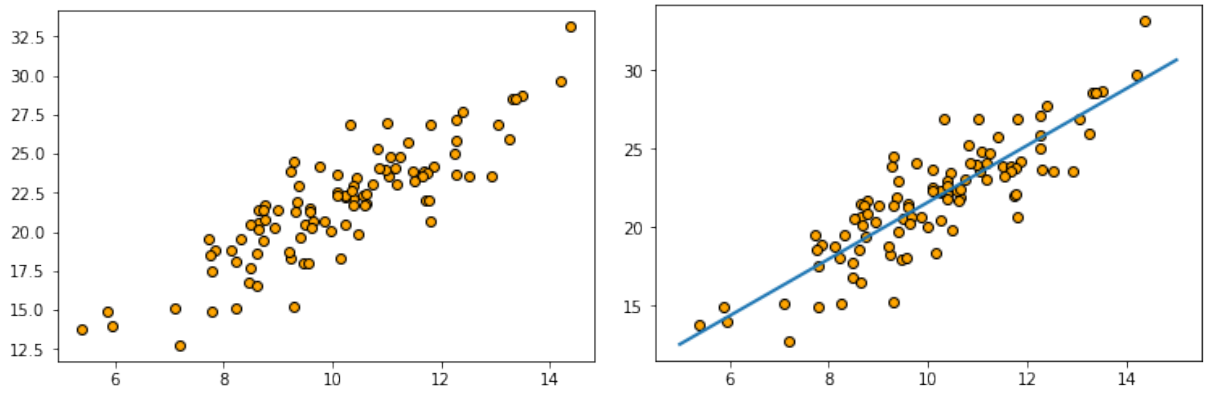
DSCI 11000: Introduction to Data Science

Background on Linear Regression

Assume that we have the following information:

- We have n values of a variable x denoted by $x_1, x_2, \ldots, x_n$.
- We have n values of a variable y denoted by $y_1, y_2, \ldots, y_n$.
- The values of x and y are paired together into points of the following form: (x_i, y_i).

If we plot all of these points, we might get a scatter plot that looks something 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 such a line to explain the relationship between the variables x and y. In fact, we could use this line to try to predict a value of y for a given value of x.

A **linear model** is an equation of the form $\hat{y} = b_0 + b_1 x$. If we plug in a value of x, this will give us a predicted value of y, which we denote as $\hat{y}$. Any values of the coefficients b_0 and b_1 will give us a linear model. The goal of **linear regression** is to find the model that provides the best fit for our data. In other words, we want to find the values for b_0 and b_1 that produce the line that is the best match for our data. This is accomplished as follows:

1. For each x_i in our data set, we generate a predicted y value $\hat{y}_i = b_0 + b_1 x_i$.
2. We then calculate the error in this prediction as follows: $e_i = y_i - \hat{y}_i$.
3. We then square all of the errors, and sum them together to calculate the sum of squared errors: $SSE = \sum_{i=0}^{n} e_i^2 = e_1^2 + e_2^2 + \cdots + e_n^2$

The linear regression model is the one with the smallest value for SSE. In this project, we will create functions to find a linear regression model for a given dataset.

Part A: Define the `predict()` function.

Define a function called `predict()`. This function should take two parameters: A list `b` and a float `x0`. This function should calculate the *y*-value obtained by plugging `x0` into a linear model whose coefficients are stored in the list `b`. In other words, the function should return the value `b[0] + b[1] * x0`.

The test code provided for this part should print the value 19.

Part B: Define the `find_sse()` function.

Define a function called `find_sse()`. This function should take three parameters:

- A list called `x`, a list called `y`, and a list called `b`.
- The lists `x` and `y` should be assumed to be the same size.
- The list `b` should be assumed to contain two elements. These elements will be the values for the coefficients b_0 and b_1 in a proposed linear regression model.

The function should return the value $SSE = e_1^2 + e_2^2 + \cdots + e_n^2$. This can be calculated by doing the following for every element `x[i]` in the list `x`:

- Call the `predict()` function, passing it the list `b` and value `x[i]`. Store the result in a variable called `y_hat`.
- Define an "error term" variable `e` to be equal to the difference between `y[i]` and `y_hat`.
- Add the square of `e` to a running total.

When you have done this for every element of `x`, return the total you have calculated.

This test code provided for this part should print out 14.

Part C: Define the `regression()` function.

Define a function called `regression()`. This function should take two parameters: A list `x` and a list `y`.

The primary goal of this function is to find the values for b_0 and b_1 for the linear regression model determined by the data in the lists `x` and `y`. It will start with a "guess" of $[b_0, b_1] = [0,0]$ and will make slight adjustments to this guess until it arrives at the values of b_0 and b_1 that result in the smallest SSE (or something very close to it). An outline of the function might be as follows:

1. Define a variable `b = [0, 0]`. This will contain our initial guesses for b_0 and b_1.

2. Use the function `find_sse()` to calculate SSE based on the provided lists `x` and `y`, and using our initial guesses for b_0 and b_1. Store the resulting SSE value in a variable called `sse`.

3. Create a loop that does the following:

    a. Stores the current value of `sse` in a variable called something like `old_sse`. We do this because we will be updating the value of `sse` and will need to compare the updated value to the previous value.

    b. Create four new updated guesses for b_0 and b_1. These should be stored in lists called something like `new_b1`, `new_b2`, `new_b3`, and `new_b4`. These new guesses should be defined as follows:
        - `new_b1` is formed by adding 0.001 to the guess for b_0, leaving b_1 alone.
        - `new_b2` is formed by subtracting 0.001 from the guess for b_0, leaving b_1 alone.
        - `new_b3` is formed by adding 0.001 to the guess for b_1, leaving b_0 alone.
        - `new_b4` is formed by subtracting 0.001 from the guess for b_1, leaving b_0 alone.

    c. Calculate four new SSE values using `x` and `y` and each of our **new_b** lists. You might store these in variables called `new_sse1`, `new_sse2`, `new_sse3`, and `new_sse4`.

d. Compare each of these new SSE values to the current SSE value. If Any of the new values are lower than the current SSE value, then do the following:
   - Replace `sse` with the new (lower) SSE value.
   - Replace `b` with the new b list that produced this lower SSE value.

   Note that all four of these comparisons should be allowed to run with each iteration of the loop. As a result, it is possible that the value of `sse` could get updated 4 times during any single iteration of your loop.

e. If an iteration of the loop completes without the value of `sse` changing, then the loop should stop.

Warning: You will be working with a while loop in this function. If you ever call this function and it takes a very long time to run, you should hit the "stop" button in Jupyter and check to see if you possibly have an infinite loop.

The test code provided for this part should produce the following output:

```
[3.994999999999671, 1.4019999999999566]
3.2000199999999994
7794
```

Be aware that there is a very easy mistake to make that will give you 8060 for the last line of output.

Part D: Working with Simulated Data

Cell 1: Generate Data. Run the code provided to generate random lists `x_data` and `y_data`. This cell also generates a scatter plot of the randomly generated data.

Cell 2: Create Regression Model. Pass the randomly generated lists to the function `regression()`, storing the returned values.

Cell 3: Print a Summary of the Model. Use the returned values to print the following statements, with the red values replaced with their appropriated values. **Add spaces to align the numbers. Round all output to 3 decimal places.**

```
Intercept of Model: b0
Slope of Model:     b1
Optimal SSE:        sse
Iterations:         iterations
```

If done correctly, this should generate the following output:

```
Intercept of Model: 3.472
Slope of Model:     1.812
Optimal SSE:        344.678
Iterations:         5947
```

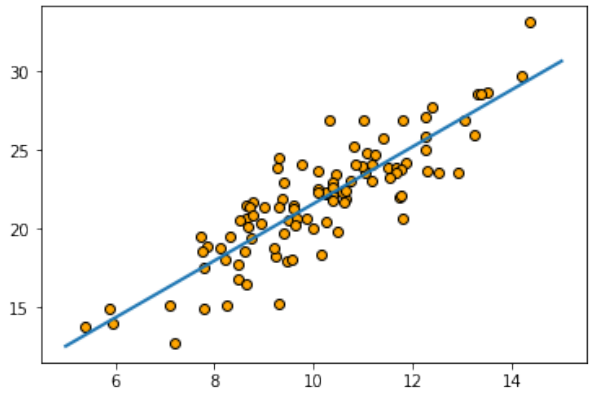
Cell 4: Generate Predictions. A list of x values is provided. Use a loop and the `predict()` function to print the following sentence for each value in the list, with the blanks filled in as appropriate:

```
If x is equal to [__], the predicted value of y is [__].
```

If done correctly, this should generate the following output:

```
If x is equal to 7.0, the predicted value of y is 16.156.
If x is equal to 8.2, the predicted value of y is 18.33.
If x is equal to 10.7, the predicted value of y is 22.86.
If x is equal to 11.3, the predicted value of y is 23.948.
If x is equal to 12.4, the predicted value of y is 25.941.
```

Cell 5: Display the Regression Line. Replace the blanks with the name of the list storing the coefficient values returned by your `regression()` function. Then run the cell to display the regression line. This should produce the following plot.



Part E: Predicting Car Prices

Cell 1: Generate Data. Run the code provided to create the lists `price` and `mileage`, as well as a scatterplot of price against mileage. These lists contains the prices and mileages for 20 used cars, all of the same model and year.

Cell 2: Create Regression Model. Use the `regression()` function to create a regression model that could be used to predict the price of a car from its mileage. Store the returned values.

Cell 3: Print a Summary of the Model. Use these returned values to print the following statements, with the red values replaced with their appropriated values. **Add spaces to align the numbers. Round all output to 3 decimal places.**

```
Intercept of Model: b0
Slope of Model:     b1
Optimal SSE:        sse
Iterations:         iterations
```

Cell 4: Generate Predictions. Predict the price of cars with the following mileages: 20500, 37100, and 62900. For each mileage, print the following statement:

```
The predicted price of a car with [__] miles is $[__].
```

You do not have to use a loop for this if you would prefer not to. **Round the predicted prices to the nearest dollar.** Pay careful attention to the units used when you provide input to the `predict()`, as well as the units for the number returned by this function.

Cell 5: Display the Regression Line. Replace the blanks with the name of the list storing the coefficient values returned by your `regression()` function. Then run the cell to display the regression line.