# hw3

September 30, 2018

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says YOUR CODE HERE or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "Benjamin Liu"
COLLABORATORS = ""
```

---

## 1 Homework 3: Loss Minimization

### 1.1 Modeling, Estimation and Gradient Descent

### 1.2 Due Date: Tuesday 10/9, 11:59 PM

### 1.3 Course Policies

Here are some important course policies. These are also located at http://www.ds100.org/fa18/.

**Collaboration Policy**

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your solution.

### 1.4 This Assignment

In this homework, we explore modeling data, estimating optimal parameters and a numerical estimation method, gradient descent. These concepts are some of the fundamentals of data science and machine learning and will serve as the building blocks for future projects, classes, and work.

After this homework, you should feel comfortable with the following:

- Practice reasoning about a model
- Build some intuition for loss functions and how they behave
- Work through deriving the gradient of a loss with respect to model parameters
- Work through a basic version of gradient descent.

This homework is comprised of completing code, deriving analytic solutions, writing LaTex and visualizing loss.

### 1.5 Submission - IMPORTANT, PLEASE READ

For this assignment and future assignments (homework and projects) you will also submit your free response and plotting questions to gradescope. To do this, you can download as PDF (File->Download As->PDF via Latex (.pdf)). You are responsible for submitting and tagging your answers in gradescope. For each free response and plotting question, please include:

1. Relevant code used to generate the plot or inform your insights
2. The written free response or plot

We are doing this to make it easier on our graders and for you, in the case you need to submit a regrade request. Gradescope (as of now) is still better for manual grading.

### 1.6 Score breakdown

| Question | Points |
|---|---|
| Question 1a | 1 |
| Question 1b | 1 |
| Question 1c | 1 |
| Question 1d | 1 |
| Question 1e | 1 |
| Question 2a | 2 |
| Question 2b | 1 |
| Question 2c | 1 |
| Question 2d | 1 |
| Question 2e | 1 |
| Question 2f | 1 |
| Question 3a | 1 |
| Question 3b | 3 |
| Question 3c | 2 |
| Question 4a | 3 |
| Question 4b | 1 |
| Question 4c | 1 |
| Question 4d | 1 |
| Question 4e | 1 |
| Question 5a | 2 |
| Question 5b | 4 |
| Question 5c | 0 |
| Question 5d | 0 |
| Question 6a | 3 |
| Question 6b | 3 |
| Question 6c | 3 |
| Question 6d | 3 |
| Question 6e | 3 |
| Question 6f | 3 |
| Question 6g | 3 |
| Question 7a | 1 |
| Question 7b | 1 |

| Question | Points |
|---|---|
| Question 7c | 1 |
| Question 7d | 1 |
| Question 7e | 0 |
| Total | 56 |

# 2 Getting Started

In [2]:
```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
import re
import seaborn as sns

# Set some parameters
plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.size'] = 16
np.set_printoptions(4)
```

In [3]:
```
# We will use plot_3d helper function to help us visualize gradient
from hw3_utils import plot_3d
```

## 2.1 Load Data

Load the data.csv file into a pandas dataframe.
Note that we are reading the data directly from the URL address.

In [4]:
```
# Run this cell to load our sample data
data = pd.read_csv("https://github.com/DS-100/fa18/raw/gh-pages/assets/datasets/hw3_data.csv", inde
data.head()
```

Out[4]:

| | x | y |
|---|---|---|
| 0 | -5.000000 | -7.672309 |
| 1 | -4.966555 | -7.779735 |
| 2 | -4.933110 | -7.995938 |
| 3 | -4.899666 | -8.197059 |
| 4 | -4.866221 | -8.183883 |

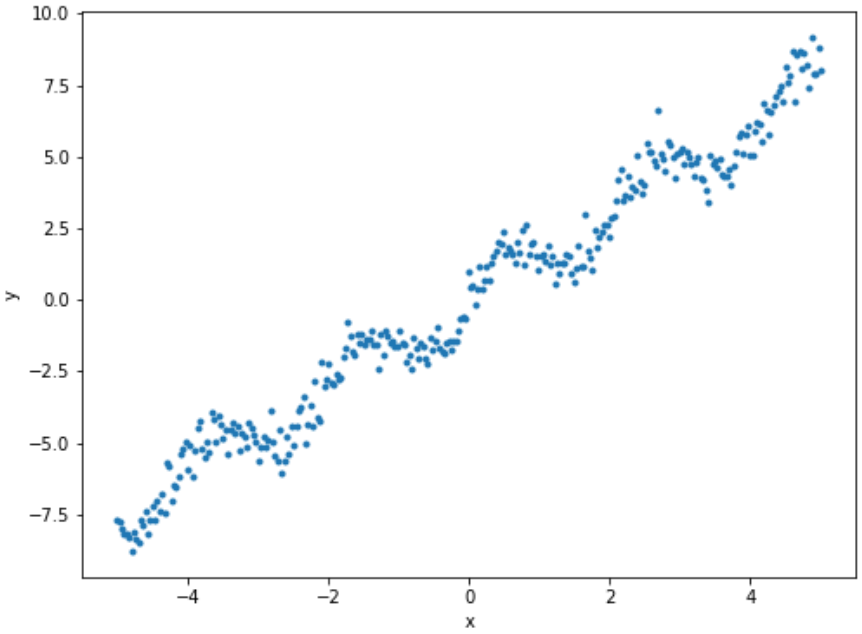
---

## 2.2 1: A Simple Model

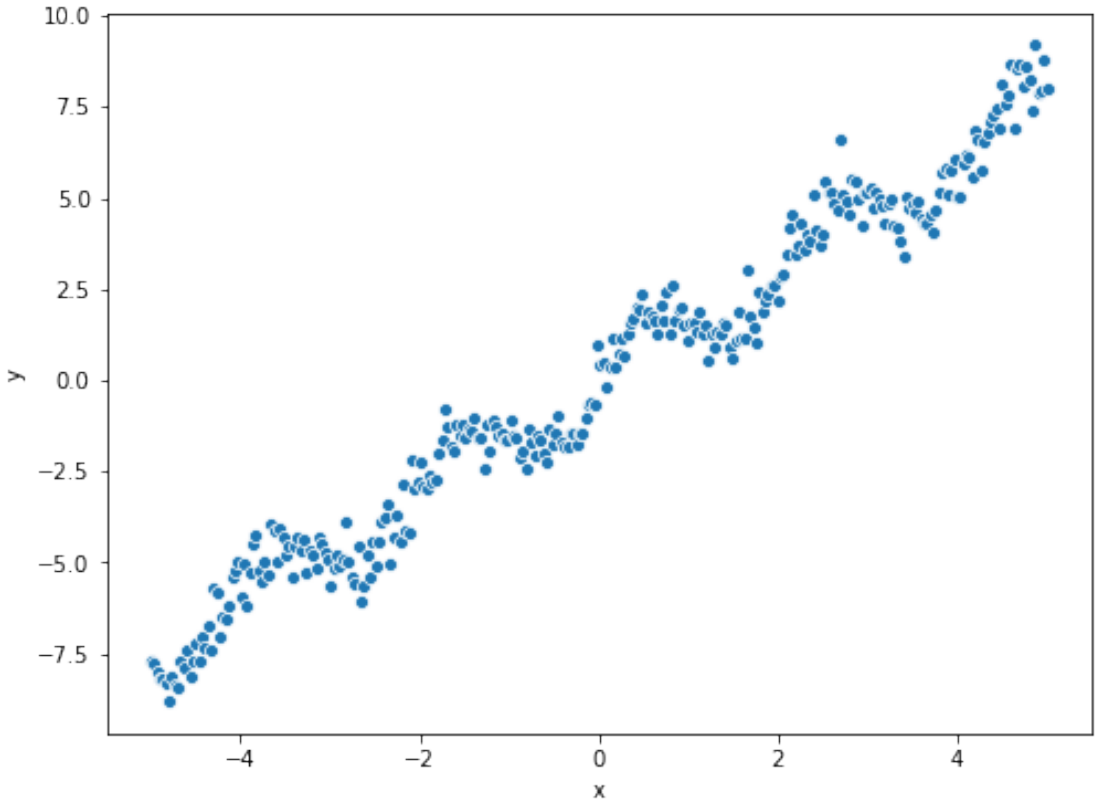
Let's start by examining our data and creating a simple model that can represent this data.

### 2.2.1 Question 1

**Question 1a** First, let's visualize the data in a scatter plot. After implementing the scatter function below, you should see something like this:




In [5]:
```
def scatter(x, y):
    """
    Generate a scatter plot using x and y

    Keyword arguments:
    x -- the vector of values x
    y -- the vector of values y
    """
    plt.figure(figsize=(8, 6))
    ### BEGIN Solution
    sns.scatterplot(x, y)
    ### END Solution
    # YOUR CODE HERE
#     raise NotImplementedError()


x = data['x']
y = data['y']
scatter(x,y)
```




**Question 1b** Describe any significant observations about the distribution of the data. How can you describe the relationship between $x$ and $y$?

### 2.2.2 Answer

1. x and y are positively correlated
2. after removing the linear trend, there is significant periodic pattern (seasonality) between x and y

**Question 1c** The data looks roughly linear, with some extra noise. For now, let's assume that the data follows some underlying linear model. We define the underlying linear model that predicts the value $y$ using the value $x$ as: $f_{\theta^*}(x) = \theta^* \cdot x$

Since we cannot find the value of the population parameter $\theta^*$ exactly, we will assume that our dataset approximates our population and use our dataset to estimate $\theta^*$. We denote our estimation with $\theta$, our fitted estimation with $\hat{\theta}$, and our model as:

$$f_\theta(x) = \theta \cdot x$$

Based on this equation, define the linear model function $\mathrm{linear\_model}$ below to estimate **y** (the $y$-values) given **x** (the $x$-values) and $\theta$. This model is similar to the model you defined in Lab 5: Modeling and Estimation.

```
In [6]: def linear_model(x, theta):
            """
            Returns the estimate of y given x and theta

            Keyword arguments:
            x -- the vector of values x
            theta -- the scalar theta
            """
            ### BEGIN Solution
            y = x * theta
            ### END Solution
            # YOUR CODE HERE
        #     raise NotImplementedError()
            return y
```

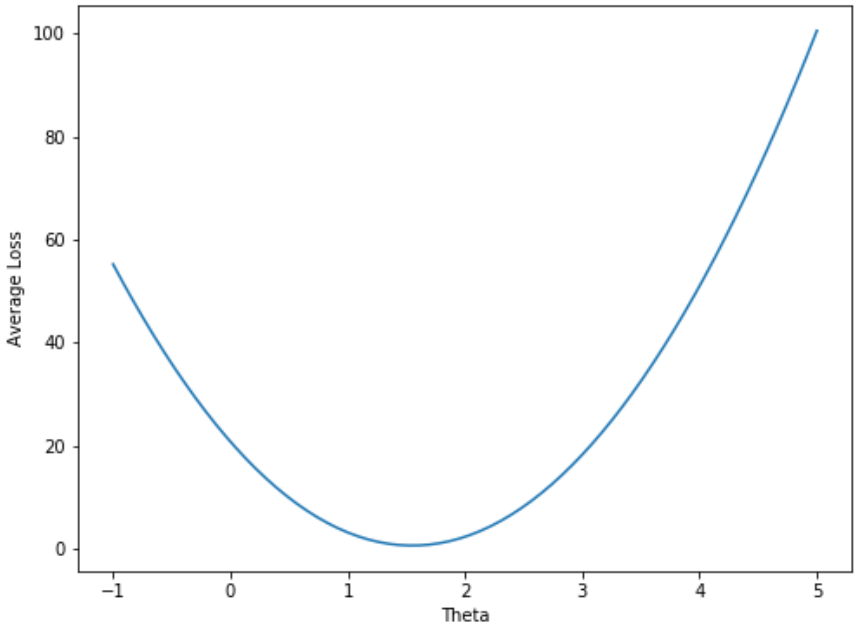
```
In [7]: assert linear_model(0, 1) == 0
        assert linear_model(10, 10) == 100
        assert np.sum(linear_model(np.array([3, 5]), 3)) == 24
        assert linear_model(np.array([7, 8]), 4).mean() == 30
```

**Question 1d** In class, we learned that the $L^2$ (or squared) loss function is smooth and continuous. Let's use $L^2$ loss to evaluate our estimate $\theta$, which we will use later to identify an optimal $\theta$, represented as $\hat{\theta}$. Define the $L^2$ loss function l2_loss below.

```
In [8]: def l2_loss(y, y_hat):
            """
            Returns the average l2 loss given y and y_hat

            Keyword arguments:
            y -- the vector of true values y
            y_hat -- the vector of predicted values y_hat
            """
            ### BEGIN Solution
            return np.mean((y-y_hat)**2)
            ### END Solution
            # YOUR CODE HERE
        #     raise NotImplementedError()
```

```
In [9]: assert l2_loss(2, 1) == 1
        assert l2_loss(2, 0) == 4
        assert l2_loss(5, 1) == 16
        assert l2_loss(np.array([5, 6]), np.array([1, 1])) == 20.5
        assert l2_loss(np.array([1, 1, 1]), np.array([4, 1, 4])) == 6.0
```

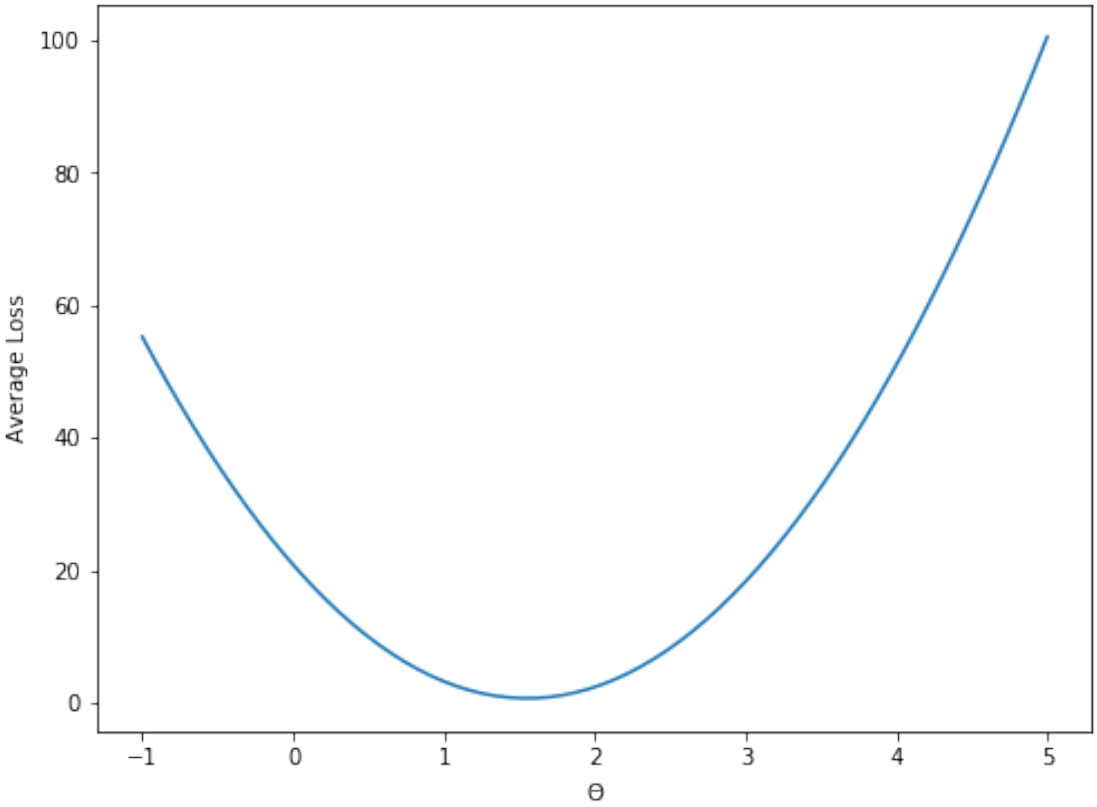
**Question 1e** First, visualize the $L^2$ loss as a function of $\theta$, where several different values of $\theta$ are given. Be sure to label your axes properly. You plot should look something like this:




What looks like the optimal value, $\hat{\theta}$, based on the visualization? Set theta_star_guess to the value of $\theta$ that appears to minimize our loss.

```
In [10]: def visualize(x, y, thetas):
             """
             Plots the average l2 loss for given x, y as a function of theta.
             Use the functions you wrote for linear_model and l2_loss.

             Keyword arguments:
             x -- the vector of values x
             y -- the vector of values y
             thetas -- an array containing different estimates of the scalar theta
             """
             avg_loss = np.array([l2_loss(y, linear_model(x, theta)) for theta in thetas])

             plt.figure(figsize=(8,6))

             # Create your plot here
             sns.lineplot(x=thetas, y=avg_loss)
         #     plt.legend()
             plt.xlabel(r"$\Theta$")
             plt.ylabel("Average Loss");

             res = {}
             res["avg_loss"] = avg_loss
             return res
```

```
    # YOUR CODE HERE
#     raise NotImplementedError()

thetas = np.linspace(-1, 5, 70)
r = visualize(x, y, thetas)
theta_star_guess = 1.5217
theta_star_guess = thetas[np.argmin(r["avg_loss"])]

# YOUR CODE HERE
# raise NotImplementedError()
```




```
In [11]: assert l2_loss(3, 2) == 1
         assert l2_loss(0, 10) == 100
         assert 1 <= theta_star_guess <= 2
```

---

### 2.3 2: Fitting our Simple Model

Now that we have defined a simple linear model and loss function, let's begin working on fitting our model to the data.

### 2.3.1 Question 2

Let's confirm our visual findings for optimal $\hat{\theta}$.

**Question 2a** First, find the analytical solution for the optimal $\hat{\theta}$ for average $L^2$ loss. Write up your solution in the cell below using LaTex.

Hint: notice that we now have $\mathbf{x}$ and $\mathbf{y}$ instead of $x$ and $y$. This means that when writing the loss function $L(\mathbf{x}, \mathbf{y}, \theta)$, you'll need to take the average of the squared losses for each $y_i$, $f_\theta(x_i)$ pair. For tips on getting started, see chapter https://www.textbook.ds100.org/ch/10/modeling_loss_functions.html of the textbook. Note that if you click "Open in DataHub", you can access the LaTeX source code of the book chapter, which you might find handy for typing up your work. Show your work, i.e. don't just write the answer.

### 2.3.2 Answer

In the matrix form, the simple linear regression is $\mathbf{x}\theta = \mathbf{y}$, where $\mathbf{x}, \mathbf{y}$ are column vector and $\theta$ is a scalar.

Define error vector $y^{er} = \mathbf{x}\theta - \mathbf{y}$, it is eivident that minimizing squared-loss is equivalent to minimizing the error vector, which is equivalent that dot product of $< y^{er}, \mathbf{x'} >= 0$ (where $x'$ is the convention for x transpose)

$$\mathbf{x'}(\mathbf{x}\theta - \mathbf{y}) = 0$$
$$\mathbf{x'x}\theta = \mathbf{x'y}$$
$$\theta = (\mathbf{x'x})^{-1}\mathbf{x'y}$$

In other words,

$$\theta = \frac{< \mathbf{y}, \mathbf{x} >}{< \mathbf{x}, \mathbf{x} >}$$

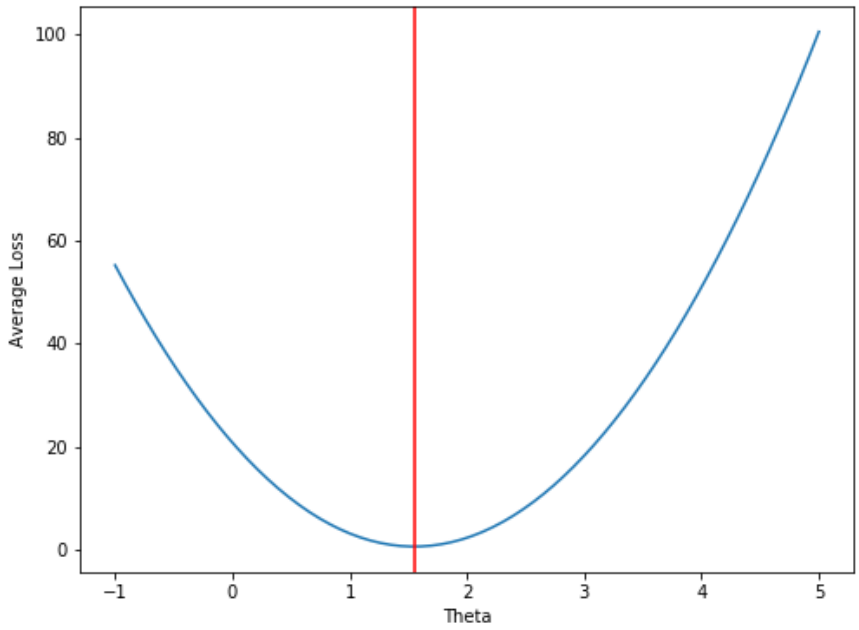
**Question 2b** Now that we have the analytic solution for $\hat{\theta}$, implement the function `find_theta` that calculates the numerical value of $\hat{\theta}$ based on our data $\mathbf{x}$, $\mathbf{y}$.

```
In [12]: def find_theta(x, y):
             """
             Find optimal theta given x and y

             Keyword arguments:
             x -- the vector of values x
             y -- the vector of values y
             """
             theta_opt = np.dot(x, y) / np.dot(x, x)
             # YOUR CODE HERE
         #    raise NotImplementedError()
             return theta_opt

In [13]: t_hat = find_theta(x, y)
         print(f'theta_opt = {t_hat}')
```

```
    assert 1.4 <= t_hat <= 1.6
```

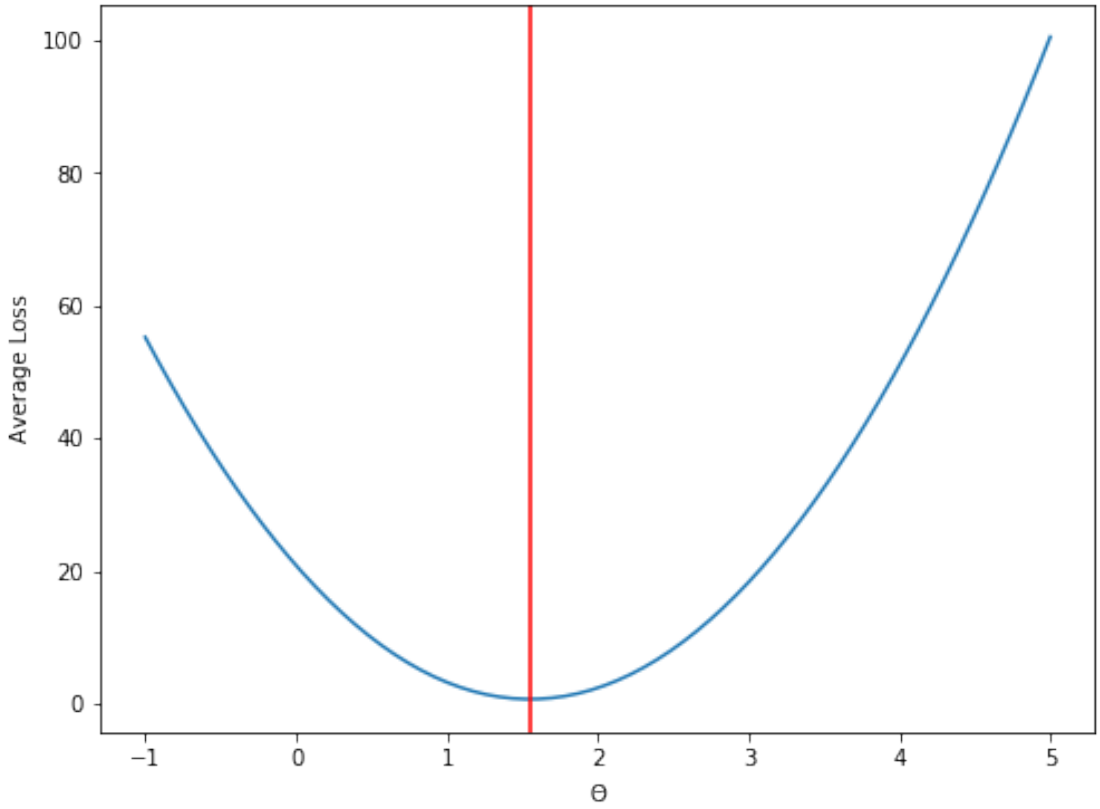
```
theta_opt = 1.550264808596222
```

**Question 2c** Now, let's plot our loss function again using the `visualize` function. But this time, add a vertical line at the optimal value of theta (plot the line $x = \hat{\theta}$). Your plot should look some-



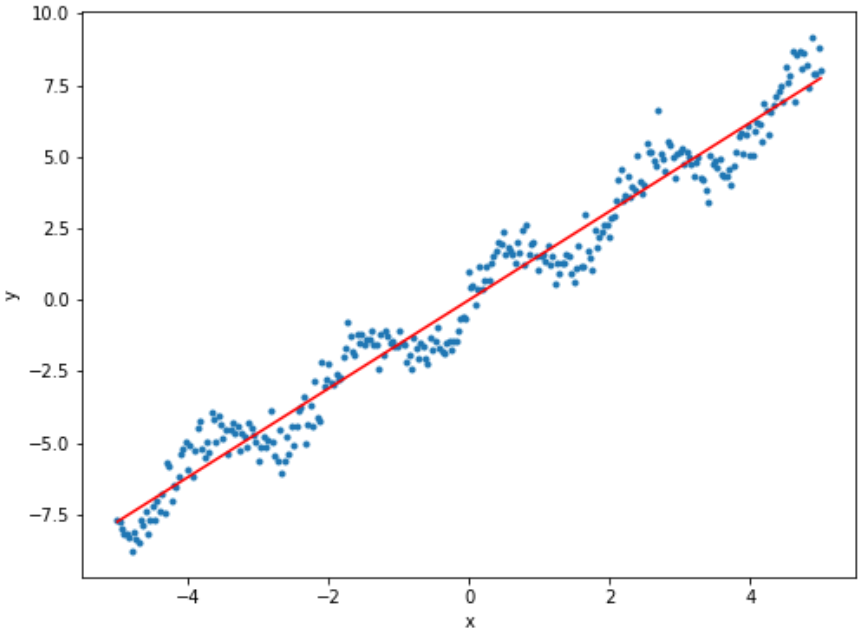

thing like this:

In [14]:
```
### BEGIN Solution
theta_opt = find_theta(x, y)
thetas = np.linspace(-1, 5, 70)
visualize(x, y, thetas)
plt.axvline(x=theta_opt, color='r');
### END Solution
# YOUR CODE HERE
# raise NotImplementedError()
```

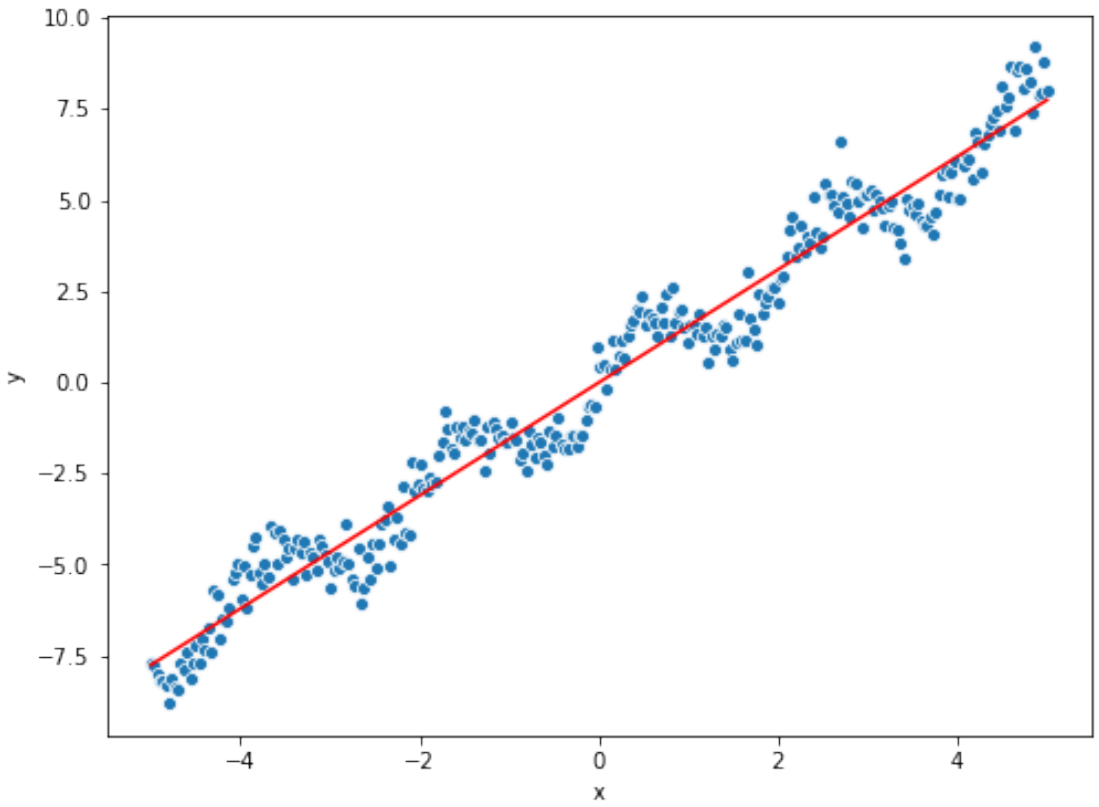



### 2.3.3 Question 2d

We now have an optimal value for $\theta$ that minimizes our loss. In the cell below, plot the scatter plot of the data from Question 1a (you can reuse the `scatter` function here). But this time, add the line $f_{\hat{\theta}}(x) = \hat{\theta} \cdot \mathbf{x}$ using the $\hat{\theta}$ you computed above. Your plot should look something like this:




In [15]:
```
### BEGIN Solution
scatter(x,y)
theta_opt = find_theta(x, y)
y_hat = x * theta_opt
sns.lineplot(x, y_hat, color='r');
plt.ylabel("y");
### END Solution
# YOUR CODE HERE
# raise NotImplementedError()
```




**Question 2e** Great! It looks like our estimator $f_{\hat{\theta}}(x)$ is able to capture a lot of the data with a single parameter $\theta$. Now let's try to remove the linear portion of our model from the data to see if we missed anything.

The remaining data is known as the residual, $\mathbf{r} = \mathbf{y} - \hat{\theta} \cdot \mathbf{x}$. Below, write a function to find the residual and plot the residuals corresponding to $x$ in a scatter plot. Plot a horizontal line at $y = 0$ to assist visualization.

```
In [16]: def visualize_residual(x, y):
             """
             Plot a scatter plot of the residuals, the remaining
             values after removing the linear model from our data.

             Keyword arguments:
             x -- the vector of values x
             y -- the vector of values y
             """
             ...
             # YOUR CODE HERE
         #     raise NotImplementedError()
             ### BEGIN Solution
             sns.scatterplot(x, y-x*find_theta(x, y));
```