assignment_01_solution

February 22, 2021

Please fill in your name and that of your teammate.

You:

Teammate:

1 Introduction

Welcome to the first lab. Take a moment to familiarize yourself with this interactive notebook. Each notebook is composed of cells. Each cell can be markdown *(easy-to-format text)* or **code** (Python 3). Markdown cells also accept LaTeX formatting. Feel free to double-click on these textual cells (switching to edit mode) to see how they were made.

When a cell is highlighted, it has two modes: command and input. Press `esc` to go to *command mode*, press `enter` to go into *edit/input mode*. There are a few shortcuts that only work in edit mode that can make your life easier. This will probably satiate your curiosity: [link].

A few shortcuts that will make your life easier: you can always press `shift+enter` to evaluate the current cell; in *command mode*, press `a` to create a new cell above the current, `b` to create one below, `m` to convert to markdown, `y` to convert to code. Cells can be also merged and split (did you check the shortcuts?).

And if you look for a command for which you don't know the shortcut yet, press `p` for the command palette and try typing what you are looking for in the search field.

1.0.1 (Computational) kernel

Jupyter as a server generates the interactive web interface that you are seeing (and using) now. To run the code, the server maintains a running instance of Python, what is called a *computational kernel*. Think of it as an open terminal window with the interactive Python open: each time you *evaluate* a cell, the code is run on that "kernel", and the output is displayed below the input cell. Try to run the cell below:

[1]:
```
a = 3+2
print(a)
```

```
5
```

Then you can use the variable `a` again in your next executions:

[2]:
```
print(a)
```

```
5
```

Just remember that the order in which you run the cells matters: for example try running next the cell below this text, and then the cell above this text once more.

[3]:
```
a = 3-2
```

Check the `Kernel` menu on top of the page for options on controlling the underlying Python execution. For example, `Restart & Run All` terminates the current kernel, launches a new fresh one, then executes all (code) cells in the notebook in order. The `Interrupt` command is also useful if a bug gets the execution stuck.

1.0.2 How to pass the lab?

Below you find the exercise questions. Each question awarding points is numbered and states the number of points like this: **[0pts]**. To answer a question, fill the cell below with your answer (markdown for text, code for implementation). Incorrect or incomplete answers are in principle worth 0 points: to assign partial reward is only up to teacher discretion. Overcomplete answers do not award extra points (though they are appreciated and will be kept under consideration). Save your work frequently! (`ctrl+s`)

You need at least 16 points (out of 24 available) to pass (66%).

2 1. Fundamentals

Let's start simple.

1.1 [1pt] Write a sentence that correctly employs the words *problem*, *solution*, *model* and *parameters set*. A problem's solution is described by a model and a fixed parameters set.

1.2 [1pt] Write the equation of a linear model. You should use LaTeX formatting, just wrap the equation in between dollar signs (e.g. `$\LaTeX$`). $y = mx + q$

1.3 [1pt] When is a system of equations *overdetermined*? When there are more know data points than unknown variables.

1.4 [1pt] Describe with your own words (i.e. English) what is a *Training Dataset*. Mind, there is not an explicit definition in the slides, you should understand the math (your lecture notes may help).

A dataset is a set of data points. A data point (in supervise learning) is a pair composed of an input (an element of the input space) and a target (element of the decision space). A training dataset is a dataset used for training a model.

1.5 [1pt] List the other two (main) learning paradigms beside *unsupervised learning*. Supervised and reinforcement.

1.6 [1pt] Which word describes when the model I use is too complex to capture the underlying simplicity of the data? Careful not to pick the wrong term.

Overfitting.

3 2. Error, loss and risk

Understanding the concept of loss and risk is fundamental to comprehending the general idea of an "error". The whole ML is founded on the basis of recognizing error and minimizing it. These questions go into a separate section to highlight how important it is that you understand what is going on here.

2.1 [3pt] About the *Loss Function*: why $L(\hat{y}, y) = 0$ if $\hat{y} = y, \ \forall y \in Y$? Because the loss quantifies the error between two elements. If those two elements are the same, there is no error, hence no loss.

If you want to implement the *Empirical Risk* in Python, you need to understand its mathematical form. Let's say that the *Loss* is a simple difference between prediction and target:

2.2 [3pt] What does $\hat{R}(h) = \sum_{i=1}^{n} L(h(x_i), y_i), \ \forall (x_i, y_i) \in D$ mean? In English here, though you will get to code it in one of the next questions.

The empirical risk is the sum of all differences between the value predicted by the model and the correct target, across all elements (i.e. data points) in the training set.

4 3. Simplest learning

Enough concepts, let's have some fun. I hope you are familiar with Python -- if not yet, you should become so by the end of the course. If your confidence is low you should start a discussion on Moodle, so that you can all help each other (and help us help you).

Do you know about `lambda` functions in Python? You can write a method that returns a function. The function can be used as if it was a method defined with `def`. Only be careful about 1. `lambda`s always (implicitly) `return` the result of their computation, and 2. you cannot write multiline lambdas (go ask Guido van Rossum why). Still, using them is easier than it sounds:

[4]:
```
def add_n(n): return lambda x: n+x
add_3 = add_n(3)
add_3(5)
```

[4]: 8

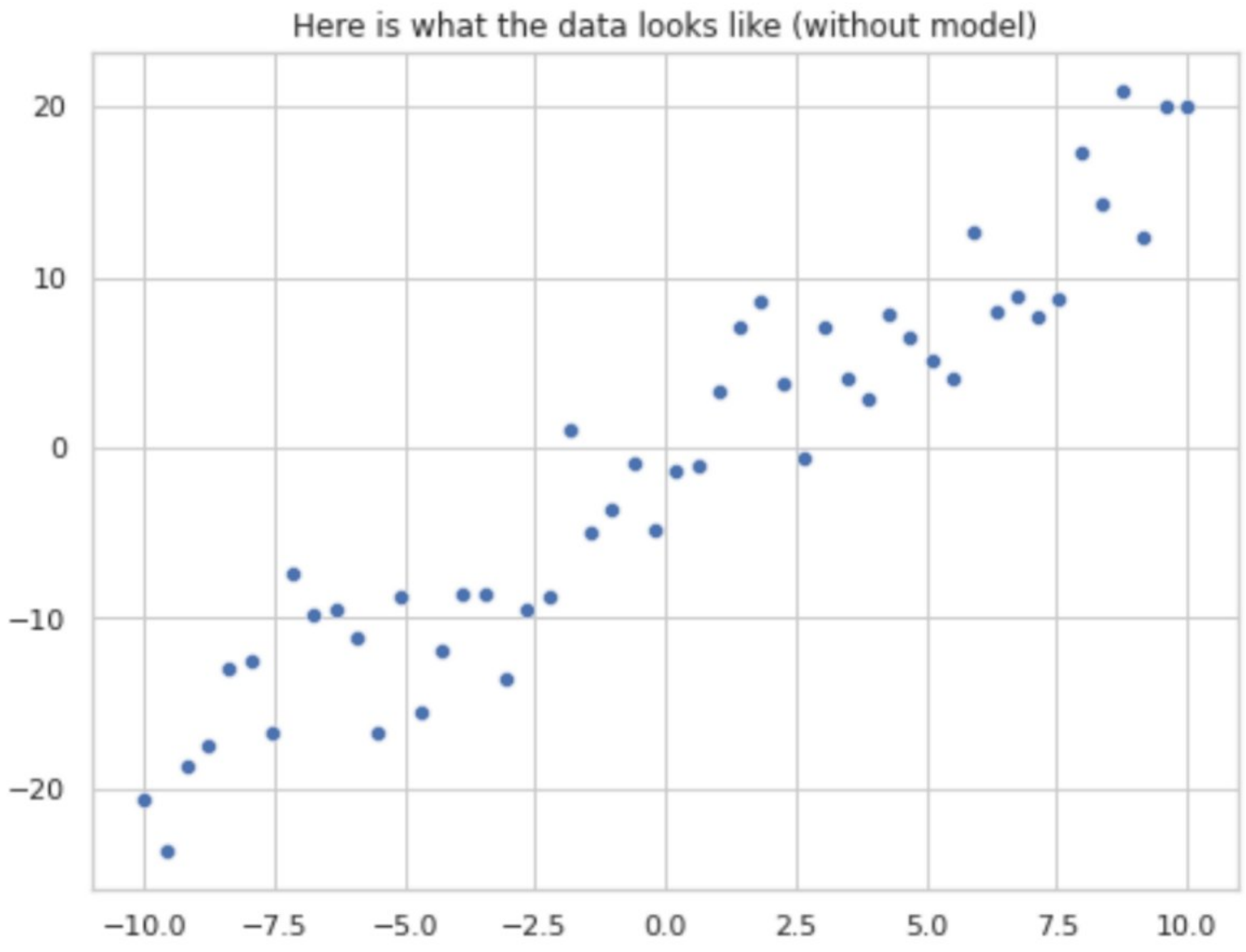
Ok how about we create and plot some artificial data? Study the code below, if there is any feature you are not yet familiar with you should make sure to learn it (ask on Moodle).

I mean it. Later on you will be required to use all of these functionalities yourself. Verify early in the course what you need to refresh and what is entirely new, because later on studying software engineering while working on the (much!) harder assignments may become a problem.

[5]:
```
# These lines are required for our plotting function below
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
# While this is the library for numerical manipulations
import numpy as np

# This is just some styling for the plotting library
sns.set(rc={'figure.figsize':(8,6)}, style="whitegrid")

# Let's create the data from a function you should be acquainted with by now
# (do you know lambdas yet? They are basically short, unnamed methods)
trg_fn = lambda x: 2*x - 1
# Of course we want the data to be a bit noisy
some_noise = lambda: np.random.normal(0,3)
# Let's generate it using numpy's linear space and a python list comprehension,
# just to make sure you know these too
data = [[x, trg_fn(x) + some_noise()] for x in np.linspace(-10, 10, 50)]

# You will find commonly data treated by axis/column rather than coordinate␣
↪pairs.
# This aggregates data series belonging to the same dimension (feature)
transpose = lambda lst: list(map(list, zip(*lst)))
data_x, data_y = transpose(data)

# And here's a canned plotting function that you are free to use (for now...)
def plot_data_and_model(model=None, text=None):
    ret = sns.scatterplot(x=data_x, y=data_y) # hard-coded data plotting␣
↪because we can
    if model is not None:
        sns.lineplot(x=data_x, y=[model(x) for x in data_x], color='darkred')
    if text is not None:
        plt.title(text)
    return ret
```

Here is what the data you just generated looks like:

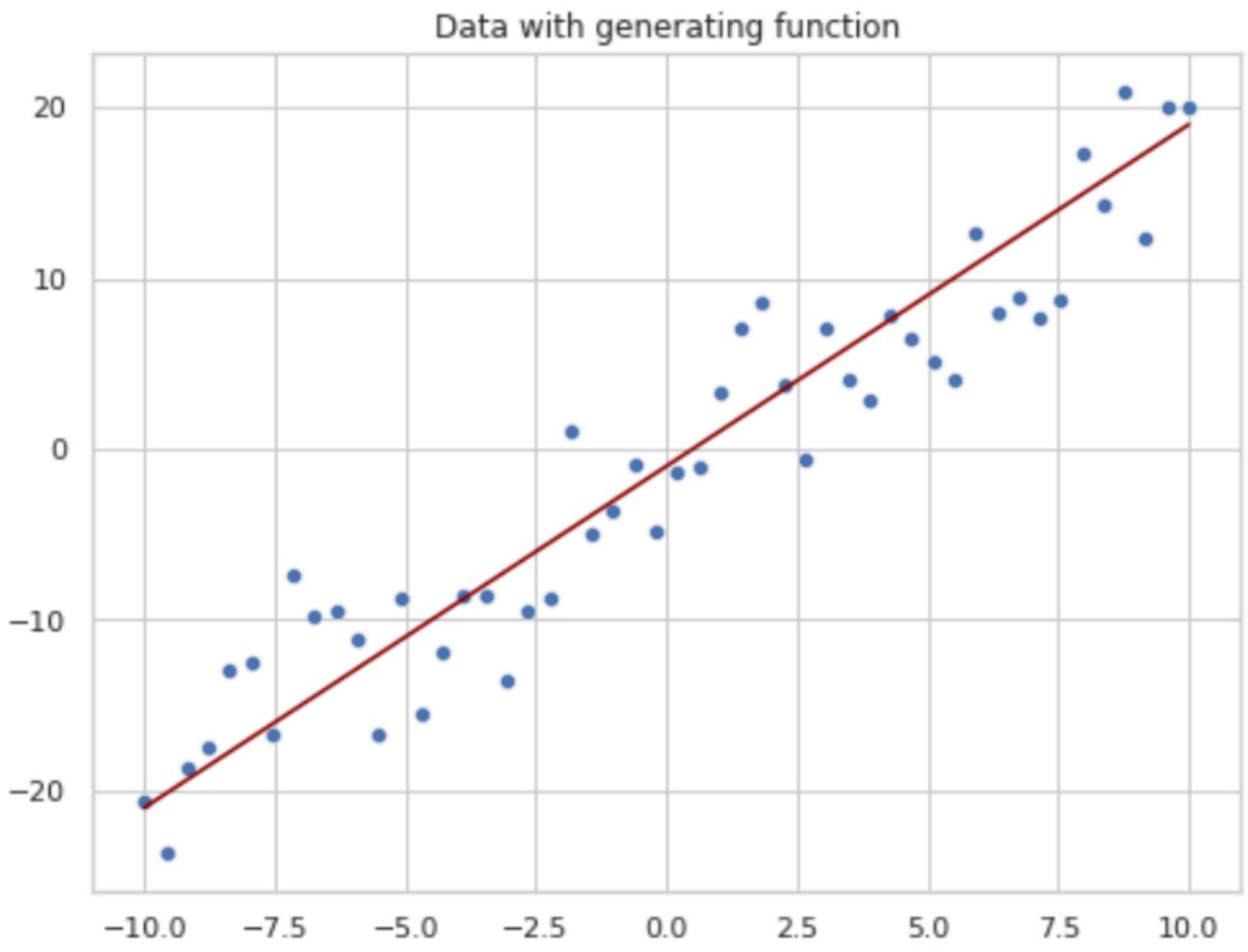
[6]:
```
plot_data_and_model(text="Here is what the data looks like (without model)");
```



And here is what it looks like if you cheat and plot the underlying function (ideally your final, learned model should look similar)

[7]:
```
plot_data_and_model(trg_fn, text="Data with generating function")
```

[7]:
```
<AxesSubplot:title={'center':'Data with generating function'}>
```



3.1 [2pt] Write your linear model as a method that takes `m` and `q` as input, and return a linear function of the form `mx + q`.

[8]:
```
lin_model = lambda m, q: lambda x: m*x+q
```

3.2 [2pt] Write your loss as a method that takes an `x` and a target and returns the absolute value of their difference (think: what happens if we forget the absolute value?)

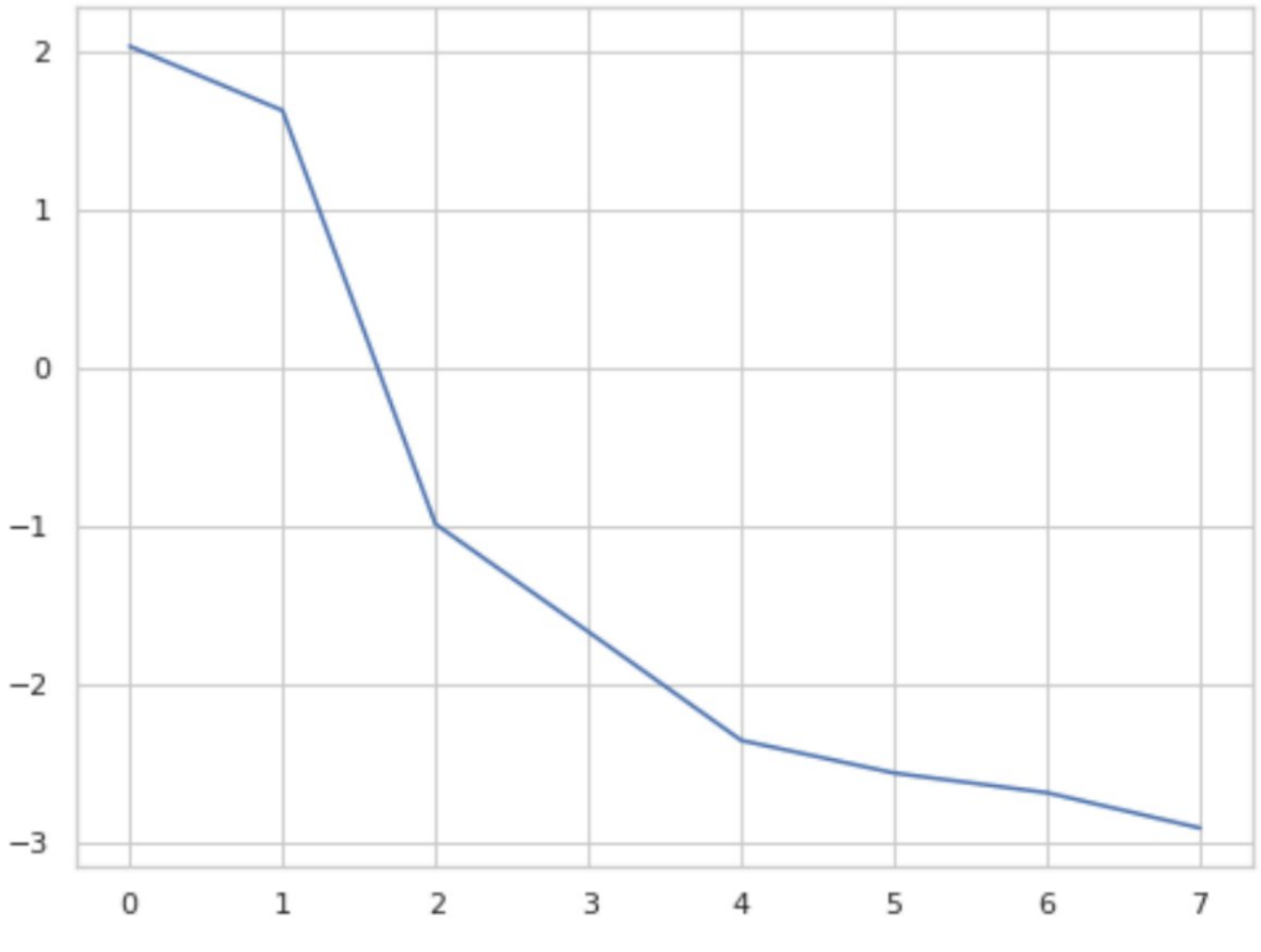
[9]:
```
loss = lambda x, trg: abs(trg - model(x))
```

3.3 [2pt] Write your risk as a method (or lambda) that takes a model as input, and returns the total loss over our (hard-coded) data

[10]:
```
risk = lambda model: sum([loss(x, trg) for x, trg in data])
```

Here is an example of a loop that generates random numbers and maintains a *minimum*.
(think: will you need to minimize or maximize the risk of your model?)

[11]:
```
min_guess = np.Infinity # higher than highest possible

best_guesses = []
for _ in range(100):
    guess = np.random.uniform(-3,3)
    if guess < min_guess:
        min_guess = guess
        best_guesses.append(min_guess)

sns.lineplot(x=range(len(best_guesses)), y=best_guesses);
```



TIP: it is always useful to visualize how the loss decreases over time, especially for debugging purposes. You can do the same next for your errors/losses.

3.4 [6pt] Randomly guess a model's parameters 1000 times. Then plot it using the call below.

```
plot_data_and_model(lin_model(m, q), text=f"m={round(m,2)}     q={round(q,2)}")
```

Make sure you understand how string interpolation works when using the format `f"hello w{2+1-3}rld"`.

You will need to modify the loop above in order to maintain both a best guess for your model and its corresponding best risk/error.

[12]:
```
best_guess = None
best_error = np.Infinity

errors = []
for _ in range(1000):
    guess = np.random.uniform(-3,3,2)
    model = lin_model(*guess)
    error = risk(model)
    if error < best_error:
        best_guess = guess
        best_error = error
    errors.append(error)

m, q = best_guess
plot_data_and_model(lin_model(m, q), text=f"m={round(m,2)}    q={round(q,2)}")
```

[12]: <AxesSubplot:title={'center':'m=1.84   q=-1.57'}>