lab12

May 24, 2022

Name: Allan Gongora

Section: 0131

1 Lab 12: Regression

Welcome to Lab 12!

Today we will get some hands-on practice with linear regression. You can find more information about this topic in Section 15.2.

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests.

[1]:
```
pip install gofer-grader
```

```
Collecting gofer-grader
  Using cached gofer_grader-1.1.0-py3-none-any.whl (9.9 kB)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
packages (from jinja2->gofer-grader) (2.0.1)
Installing collected packages: gofer-grader
Successfully installed gofer-grader-1.1.0
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# Run this cell, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
```

```
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
warnings.simplefilter('ignore', UserWarning)

# These lines load the tests.
from gofer.ok import check
```

Recommended Reading: * The Regression Line

1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include:
   A) Sentence reponses to questions that ask for an explanation
   B) Numeric responses to multiple choice questions
   C) Programming code
2) Moreover, throughout this lab and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Otherwise, you will fail tests that you thought you were passing previously!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. How Faithful is Old Faithful? Revisited

Let's revisit a question from Lab 11. Last lab, we investigated Old Faithful, a geyser in Yellowstone National Park in Central United States. It's famous for erupting on a fairly regular schedule.

To recap, some of Old Faithful's eruptions last longer than others. Today, we will use the same dataset on eruption durations and waiting times to see if we can make predict the wait time from the eruption duration using linear regression.

The dataset has one row for each observed eruption. It includes the following columns: - **Duration**: Eruption duration, in minutes - **Wait**: Time between this eruption and the next, also in minutes

Run the next cell to load the dataset.

[3]:
```
faithful = Table.read_table("faithful.csv")
faithful
```

[3]:
```
duration | wait
3.6      | 79
1.8      | 54
3.333    | 74
2.283    | 62
4.533    | 85
2.883    | 55
4.7      | 88
3.6      | 85
1.95     | 51
4.35     | 85
… (262 rows omitted)
```

Remember from last lab that we concluded eruption time and waiting time are positively correlated. The table below called `faithful_standard` contains the eruption durations and waiting times in standard units.

[4]:
```
duration_mean = np.mean(faithful.column("duration"))
duration_std = np.std(faithful.column("duration"))
wait_mean = np.mean(faithful.column("wait"))
wait_std = np.std(faithful.column("wait"))

faithful_standard = Table().with_columns(
    "duration (standard units)", (faithful.column("duration") - duration_mean) /
 ↪ duration_std,
    "wait (standard units)", (faithful.column("wait") - wait_mean) / wait_std
)
faithful_standard
```

[4]:
```
duration (standard units) | wait (standard units)
0.0984989                 | 0.597123
-1.48146                  | -1.24518
-0.135861                 | 0.228663
-1.0575                   | -0.655644
0.917443                  | 1.03928
-0.530851                 | -1.17149
1.06403                   | 1.26035
0.0984989                 | 1.03928
-1.3498                   | -1.46626
0.756814                  | 1.03928
… (262 rows omitted)
```

The next cell computes the correlation `r`:

[5]:
```
r = np.mean(faithful_standard.column(0) * faithful_standard.column(1))
r
```

[5]:
```
0.9008111683218132
```

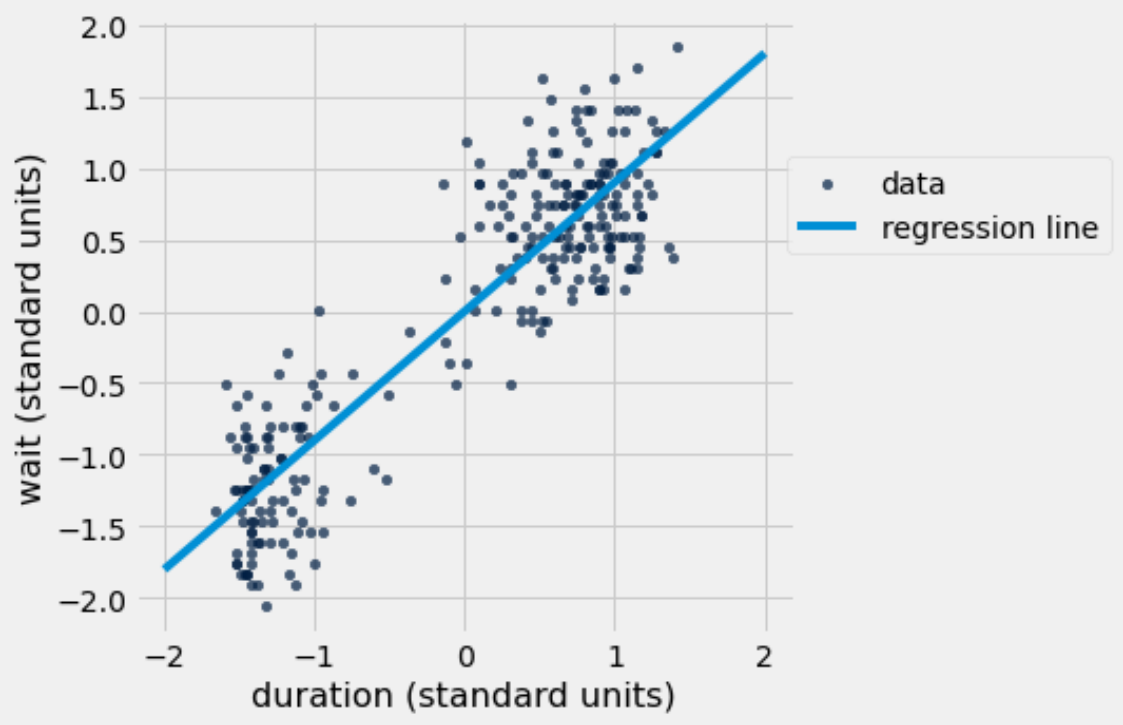
1.2 2. The Regression Line

The correlation coefficient is the slope of the regression line when the data are expressed in standard units.

The next cell plots the regression line in standard units:

$$\text{waiting time (standard units)} = r \times \text{eruption duration (standard units)}.$$

[6]:
```
def plot_data_and_line(dataset, x, y, point_0, point_1):
    """Makes a scatter plot of the dataset, along with a line passing through␣
 ↪two points."""
    dataset.scatter(x, y, label="data")
    xs, ys = zip(point_0, point_1)
    plots.plot(xs, ys, label="regression line")
    plots.legend(bbox_to_anchor=(1.5,.8))

plot_data_and_line(faithful_standard,
                   "duration (standard units)",
                   "wait (standard units)",
                   [-2, -2*r],
                   [2, 2*r])
```



How would you take a point in standard units and convert it back to original units? We'd have to "stretch" its horizontal position by `duration_std` and its vertical position by `wait_std`.

That means the same thing would happen to the slope of the line.

Stretching a line horizontally makes it less steep, so we divide the slope by the stretching factor. Stretching a line vertically makes it more steep, so we multiply the slope by the stretching factor.

Question 2.1 What is the slope of the regression line in original units?

(If the "stretching" explanation is unintuitive, consult Section 15.2 in the textbook.)

[30]:
```
slope = r * (wait_std / duration_std)
slope
```

[30]:
```
10.729641395133527
```

We know that the regression line passes through the point `(duration_mean, wait_mean)`. You might recall from high school algebra that the equation for the line is:

$$\text{waiting time} - \texttt{wait\_mean} = \texttt{slope} \times (\text{eruption duration} - \texttt{duration\_mean})$$

After rearranging that equation slightly, the intercept turns out to be:

[31]:
```
intercept = slope*(-duration_mean) + wait_mean
intercept
```

[31]:
```
33.47439702275335
```

[32]:
```
check('tests/q2_1.py')
```

[32]:
```
<gofer.ok.OKTestsResult at 0x7f40eb9e6110>
```

1.3 3. Investigating the Regression Line

The slope and intercept tell you exactly what the regression line looks like. To predict the waiting time for an eruption, multiply the eruption's duration by `slope` and then add `intercept`.

Question 3.1 Compute the predicted waiting time for an eruption that lasts 2 minutes, and for an eruption that lasts 5 minutes.

[36]:
```
two_minute_predicted_waiting_time = 2*slope + intercept
five_minute_predicted_waiting_time = 5*slope + intercept

# Here is a helper function to print out your predictions
# (you don't need to modify it):
def print_prediction(duration, predicted_waiting_time):
    print("After an eruption lasting", duration,
          "minutes, we predict you'll wait", predicted_waiting_time,
          "minutes until the next eruption.")

print_prediction(2, two_minute_predicted_waiting_time)
print_prediction(5, five_minute_predicted_waiting_time)
```

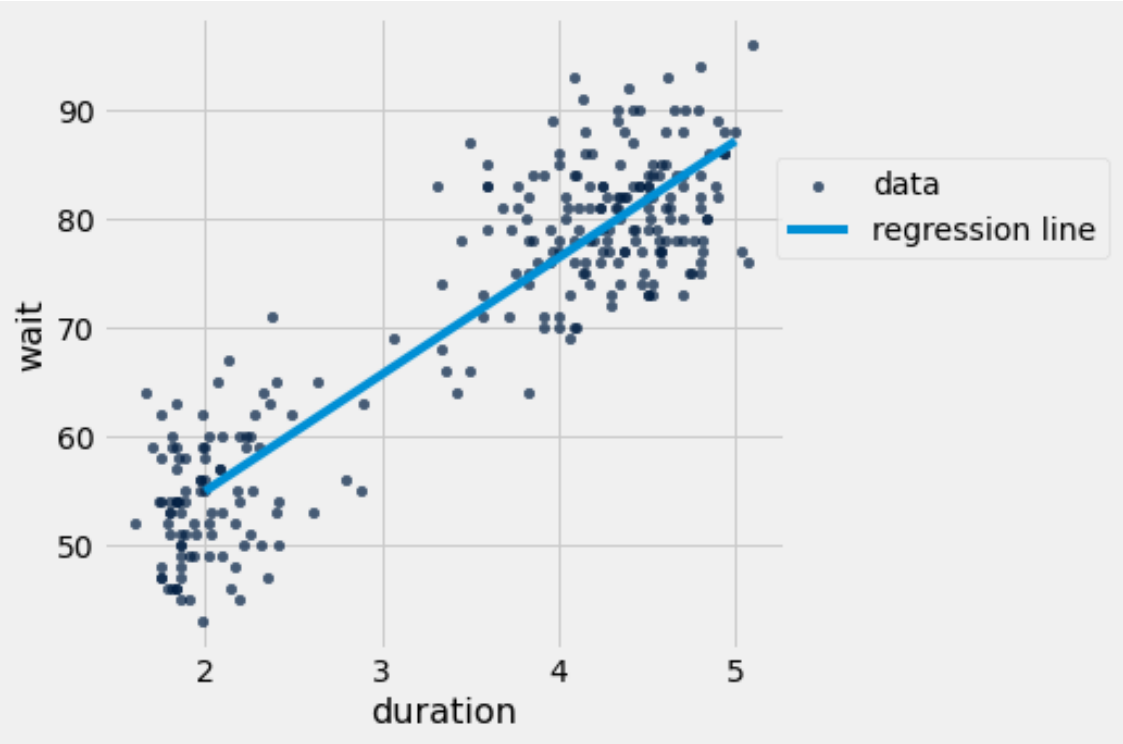
```
After an eruption lasting 2 minutes, we predict you'll wait 54.933679813020404
minutes until the next eruption.
After an eruption lasting 5 minutes, we predict you'll wait 87.12260399842098
minutes until the next eruption.
```

[39]:
```
check('tests/q3_1.py')
```

[39]:
```
<gofer.ok.OKTestsResult at 0x7f40d27e9fd0>
```

The next cell plots the line that goes between those two points, which is (a segment of) the regression line.

[40]:
```
plot_data_and_line(faithful, "duration", "wait",
                   [2, two_minute_predicted_waiting_time],
                   [5, five_minute_predicted_waiting_time])
```



Question 3.2 Make predictions for the waiting time after each eruption in the `faithful` table. (Of course, we know exactly what the waiting times were! We are doing this so we can see how accurate our predictions are.) Put these numbers into a column in a new table called `faithful_predictions`. Its first row should look like this:

duration	wait	predicted wait
3.6	79	72.1011

Hint: Your answer can be just one line. There is no need for a `for` loop; use array arithmetic instead.

[43]:
```
faithful_predictions = faithful.with_column("predicted wait",␣
  ↪faithful["duration"]*slope + intercept)
faithful_predictions
```

[43]:
```
duration | wait | predicted wait
3.6      | 79   | 72.1011
1.8      | 54   | 52.7878
3.333    | 74   | 69.2363
2.283    | 62   | 57.9702
4.533    | 85   | 82.1119
2.883    | 55   | 64.408
4.7      | 88   | 83.9037
3.6      | 85   | 72.1011
1.95     | 51   | 54.3972
4.35     | 85   | 80.1483
… (262 rows omitted)
```

[44]:
```
check('tests/q3_2.py')
```

[44]:
```
<gofer.ok.OKTestsResult at 0x7f40e405cdd0>
```

Question 3.3 How close were we? We computed the *residual* for each eruption in the dataset. The residual is the difference (not the absolute difference) between the actual waiting time and the predicted waiting time. Add the residuals to `faithful_predictions` as a new column called `"residual"`, naming the resulting table `faithful_residuals`.

Hint: Again, your code will be much simpler if you don't use a `for` loop.

[45]:
```
residual = faithful_predictions.column(1) - faithful_predictions.column(2)
faithful_residuals = faithful_predictions.with_column("residual", residual)
faithful_residuals
```

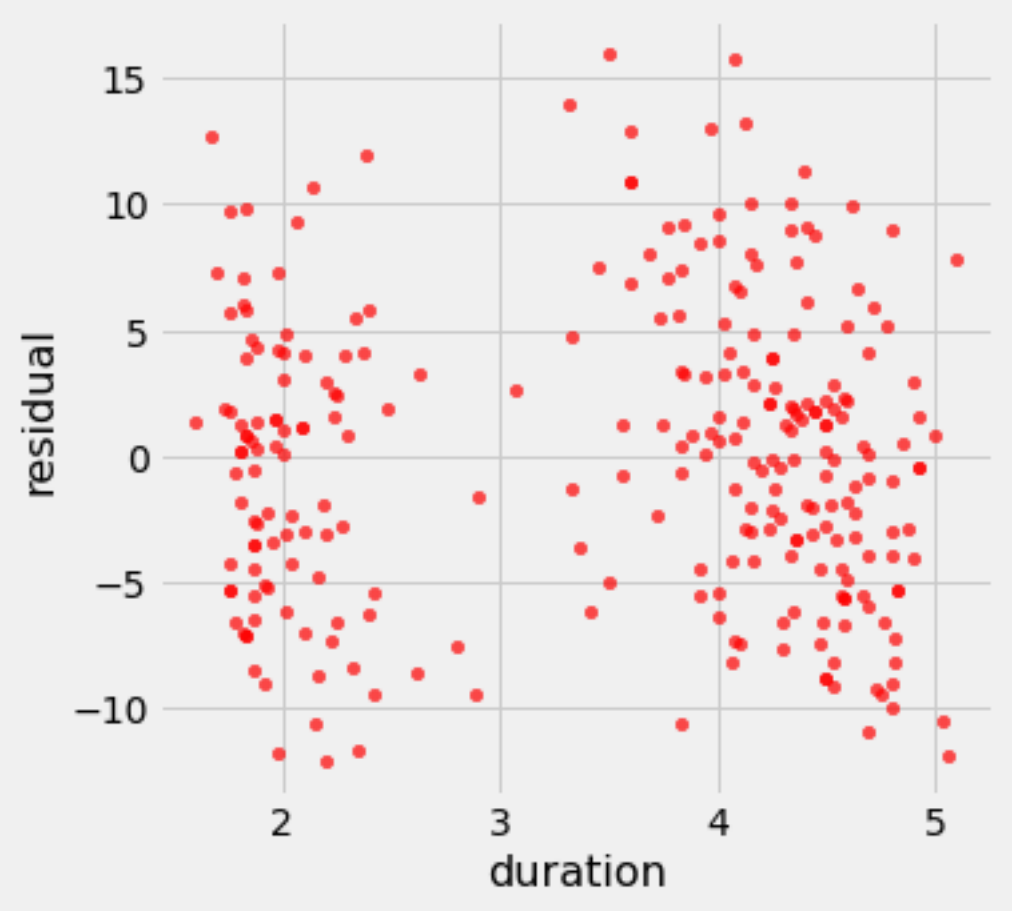
[45]:
```
duration | wait | predicted wait | residual
3.6      | 79   | 72.1011        | 6.89889
1.8      | 54   | 52.7878        | 1.21225
3.333    | 74   | 69.2363        | 4.76371
2.283    | 62   | 57.9702        | 4.02983
4.533    | 85   | 82.1119        | 2.88814
2.883    | 55   | 64.408         | -9.40795
4.7      | 88   | 83.9037        | 4.09629
3.6      | 85   | 72.1011        | 12.8989
1.95     | 51   | 54.3972        | -3.3972
4.35     | 85   | 80.1483        | 4.85166
… (262 rows omitted)
```

[46]:
```
check('tests/q3_3.py')
```

[46]:
```
<gofer.ok.OKTestsResult at 0x7f40d1604c10>
```

Here is a plot of the residuals you computed. Each point corresponds to one eruption. It shows how much our prediction over- or under-estimated the waiting time.

[47]:
```
faithful_residuals.scatter("duration", "residual", color="r")
```



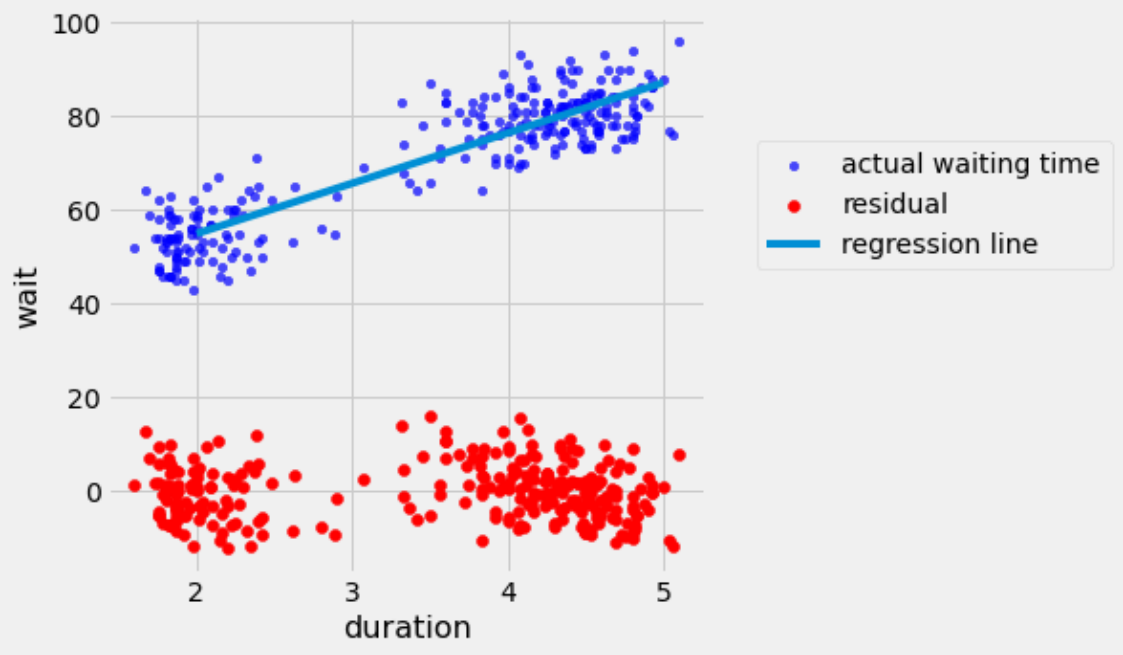
There isn't too much of a pattern in the residuals, which confirms that it's reasonable to use linear regression for prediction. It's true that there are two separate clouds; the eruption durations seemed to fall into two distinct clusters. But that's just a pattern in the eruption durations, not a pattern in the relationship between eruption durations and waiting times. A larger concern is that there may be more positive than negative residuals in a particular region of the horizontal axis. For both clusters, the points are distributed fairly evenly above and below zero, which is a confirmation that the association is mostly linear.

1.4 4. How Accurate Are Different Predictions?

The correlation coefficient is close to 1, implying that the observed values are tightly clustered around the regression line. The residuals are overall small (close to 0) in comparison to the waiting times.

We can see that visually by plotting the waiting times and residuals together:

[48]:
```
faithful_residuals.scatter("duration", "wait", label="actual waiting time",␣
 ↪color="blue")
plots.scatter(faithful_residuals.column("duration"), faithful_residuals.
 ↪column("residual"), label="residual", color="r")
plots.plot([2, 5], [two_minute_predicted_waiting_time,␣
 ↪five_minute_predicted_waiting_time], label="regression line")
plots.legend(bbox_to_anchor=(1.7,.8));
```



Question 4.1 In `faithful`, no eruption lasted exactly 0, 2.5, or 60 minutes. Using this line, what is the predicted waiting time for an eruption that lasts 0 minutes? 2.5 minutes? An hour?

[49]:
```
zero_minute_predicted_waiting_time = intercept
two_point_five_minute_predicted_waiting_time = 2.5 * slope + intercept
hour_predicted_waiting_time = 60 * slope + intercept

print_prediction(0, zero_minute_predicted_waiting_time)
print_prediction(2.5, two_point_five_minute_predicted_waiting_time)
print_prediction(60, hour_predicted_waiting_time)
```

```
After an eruption lasting 0 minutes, we predict you'll wait 33.47439702275335
```

```
minutes until the next eruption.
After an eruption lasting 2.5 minutes, we predict you'll wait 60.29850051058717
minutes until the next eruption.
After an eruption lasting 60 minutes, we predict you'll wait 677.252880730765
minutes until the next eruption.
```

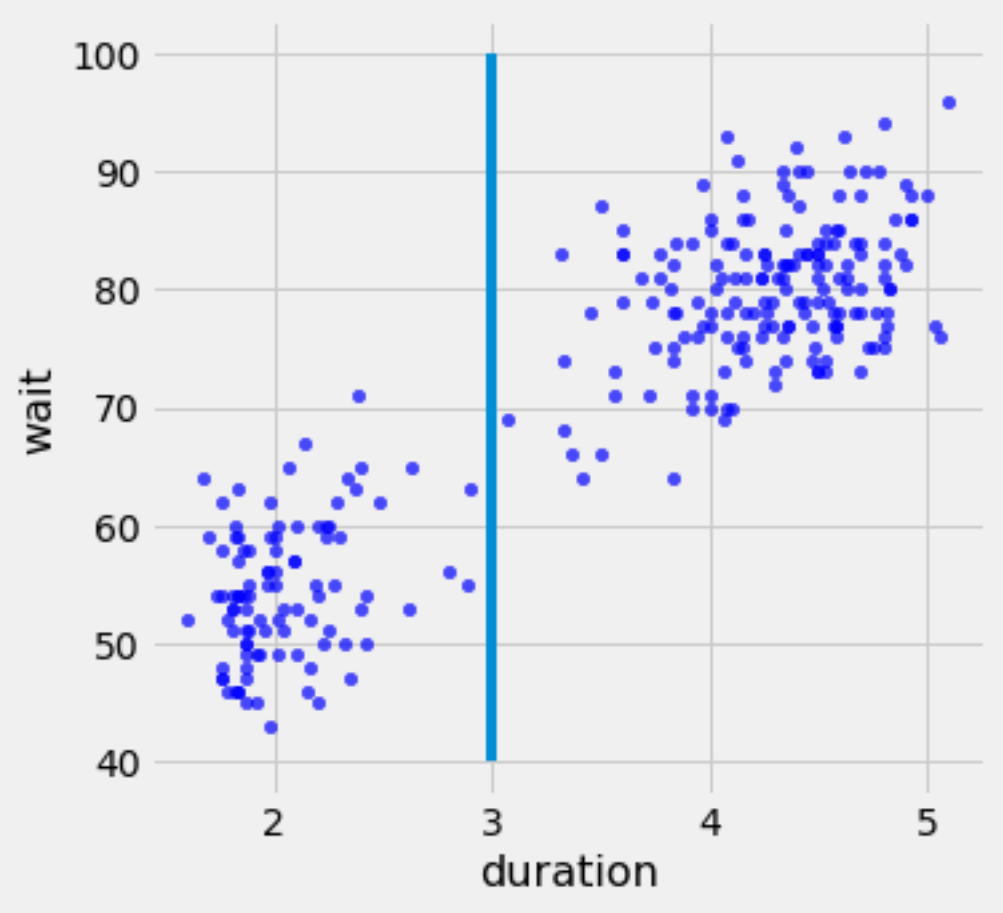
[50]:
```
check('tests/q4_1.py')
```

[50]:
```
<gofer.ok.OKTestsResult at 0x7f40d1620c10>
```

1.5 5. Divide and Conquer

Let's see what happens if we treat the two clusters of observations differently. It appears from the scatter diagram that there are two clusters of points: one for durations around 2 and another for durations between 3.5 and 5. A vertical line at 3 divides the two clusters.

[51]:
```
faithful.scatter("duration", "wait", label="actual waiting time", color="blue")
plots.plot([3, 3], [40, 100]);
```



The `standardize` function from lecture appears below, which returns a table of values in standard

units.

[52]:
```
def standard_units(any_numbers):
    "Convert any array of numbers to standard units."
    return (any_numbers - np.mean(any_numbers)) / np.std(any_numbers)

def standardize(t):
    """Return a table in which all columns of t are converted to standard units.
    """
    t_su = Table()
    for label in t.labels:
        t_su = t_su.with_column(label + ' (su)', standard_units(t.
    column(label)))
    return t_su
```

Question 5.1 Separately compute the regression coefficients r for all the points with a duration below 3 **and then** for all the points with a duration above 3. To do so, create a function that computes `r` from a table and pass it two different tables of points, `below_3` and `above_3`.

[56]:
```
def reg_coeff(t):
    """Return the regression coefficient for columns 0 & 1."""
    t_su = standardize(t)
    return np.mean(t_su[0] * t_su[1])

below_3 = faithful.where("duration", are.below(3))
above_3 = faithful.where("duration", are.above(3))
below_3_r = reg_coeff(below_3)
above_3_r = reg_coeff(above_3)
print("For points below 3, r is", below_3_r, "; for points above 3, r is",␣
    above_3_r)
```

```
For points below 3, r is 0.2901895264925431 ; for points above 3, r is
0.3727822255707511
```

[57]:
```
check('tests/q5_1.py')
```

[57]:
```
<gofer.ok.OKTestsResult at 0x7f40d1024a50>
```

Question 5.2 Write functions `slope_of` and `intercept_of` below.

When you're done, the functions `wait_below_3` and `wait_above_3` should each use a different regression line to predict a wait time for a duration. The first function should use the regression line for all points with duration below 3.2. The second function should use the regression line for all points with duration above 3.2.

[69]:
```
def slope_of(t, r):
    """Return the slope of the regression line for t in original units.

    Assume that column 0 contains x values and column 1 contains y values.
```

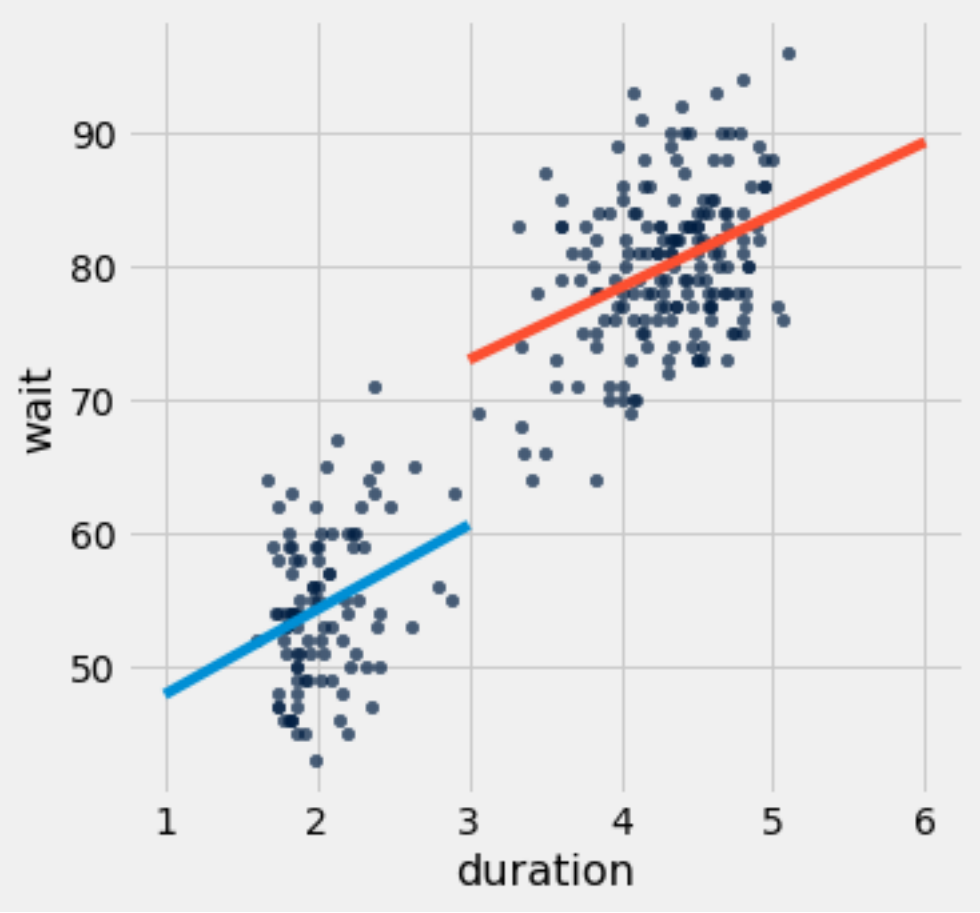
```
    r is the regression coefficient for x and y.
    """
    return r * (t[1].std() / t[0].std())

def intercept_of(t, r):
    """Return the slope of the regression line for t in original units."""
    s = slope_of(t, r)
    return s*(-t[0].mean()) + t[1].mean()


below_3_a = slope_of(below_3, below_3_r)
below_3_b = intercept_of(below_3, below_3_r)
above_3_a = slope_of(above_3, above_3_r)
above_3_b = intercept_of(above_3, above_3_r)

def wait_below_3(duration):
    return below_3_a * duration + below_3_b

def wait_above_3(duration):
    return above_3_a * duration + above_3_b
```

[70]:
```
check('tests/q5_2.py')
```

[70]:
```
<gofer.ok.OKTestsResult at 0x7f40d3439350>
```

The plot below shows two different regression lines, one for each cluster!

[71]:
```
faithful.scatter(0, 1)
plots.plot([1, 3], [wait_below_3(1), wait_below_3(3)])
plots.plot([3, 6], [wait_above_3(3), wait_above_3(6)]);
```



Question 5.3 Write a function `predict_wait` that takes a `duration` and returns the predicted wait time using the appropriate regression line, depending on whether the duration is below 3 or greater than (or equal to) 3.

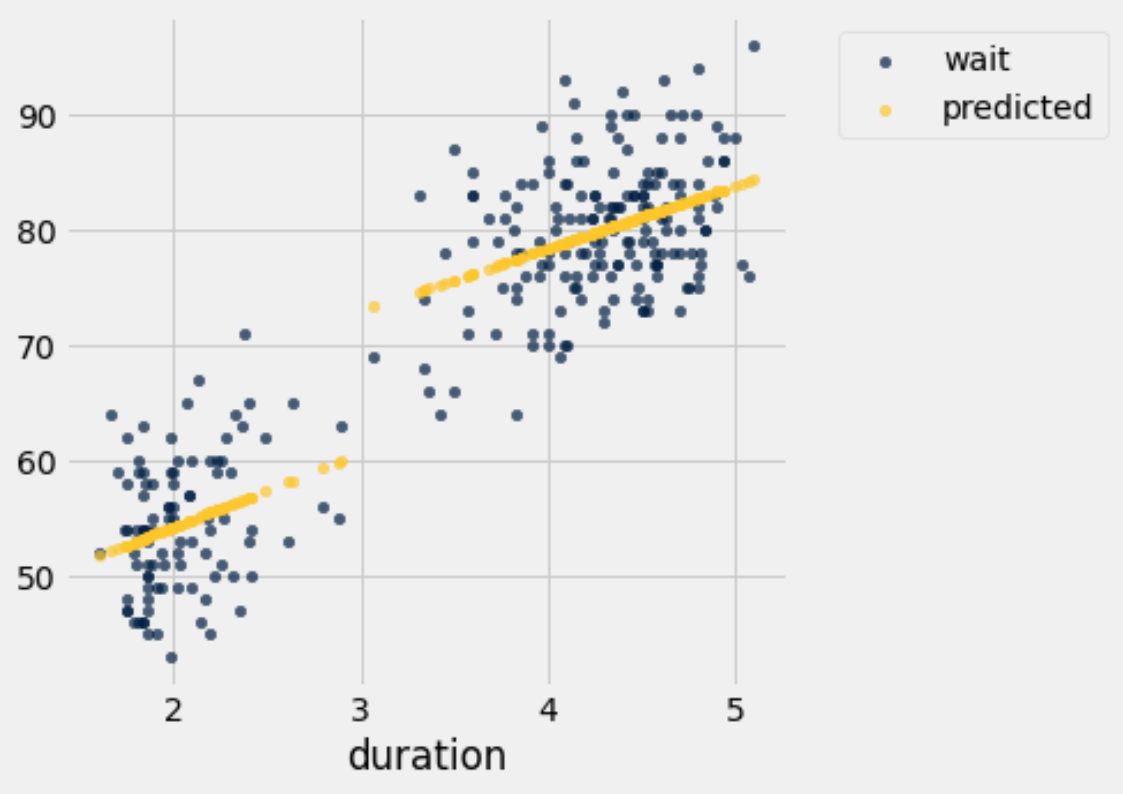
[72]:
```
def predict_wait(duration):
    """Return the wait predicted by the appropriate one of the two regression␣
 ↪lines above."""
    return wait_below_3(duration) if duration < 3 else wait_above_3(duration)
```

[73]:
```
check('tests/q5_3.py')
```

[73]:
```
<gofer.ok.OKTestsResult at 0x7f40d11a7350>
```

The predicted wait times for each point appear below.

[74]:
```
faithful.with_column('predicted', faithful.apply(predict_wait, 'duration')).
 ↪scatter(0)
```



Further Exploration (ungraded): When drawing a line through each cluster separately, we discovered two different but similar lines. Here are some natural questions to explore, if you want to continue working with these data: * How much more accurate do we expect predictions to be using two lines instead of one? Can we measure this improvement using residuals? * Are the lines really different, or did they just come out different due to chance because we have only a small number of observations? How could we tell? * Could it be that the slopes of the lines are the same, but the intercepts are different?

1.6 6. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

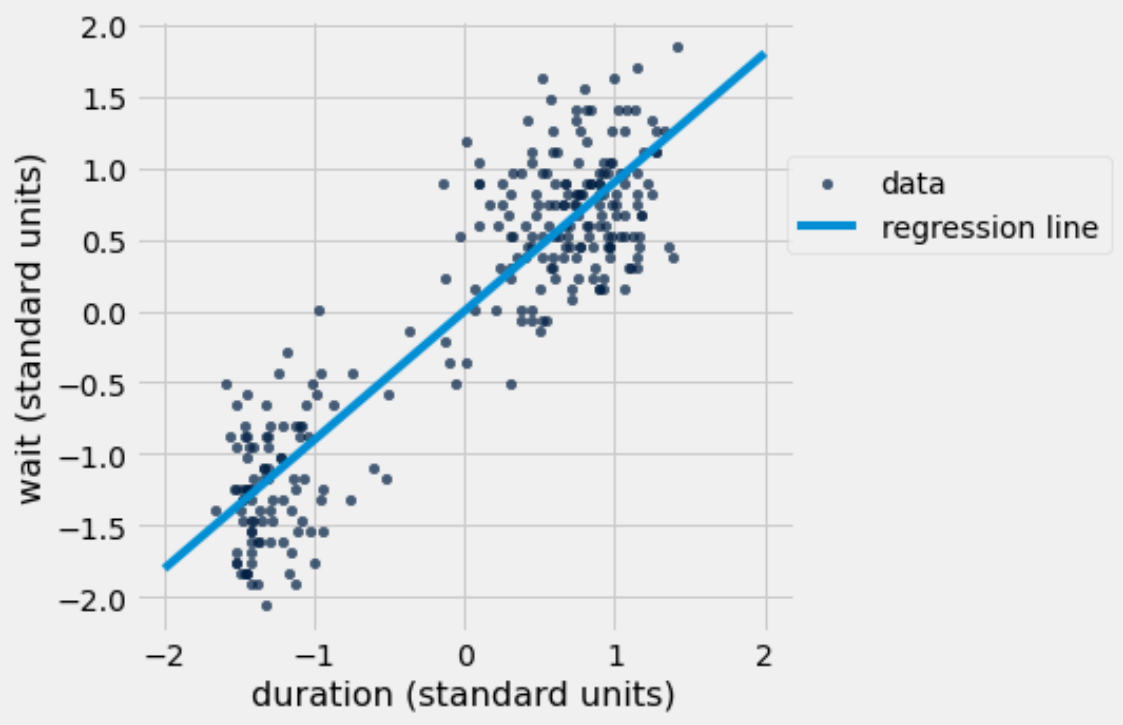
Double check that you have completed all of the free response questions as the auto-grader does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this lab. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

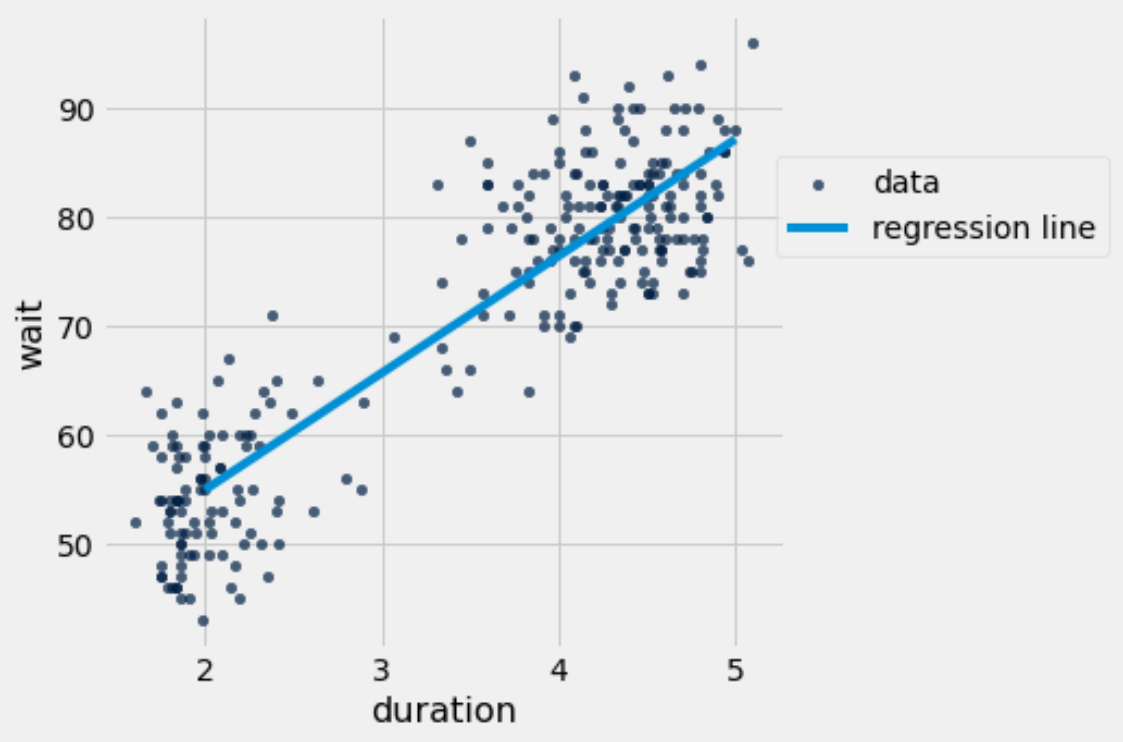
The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

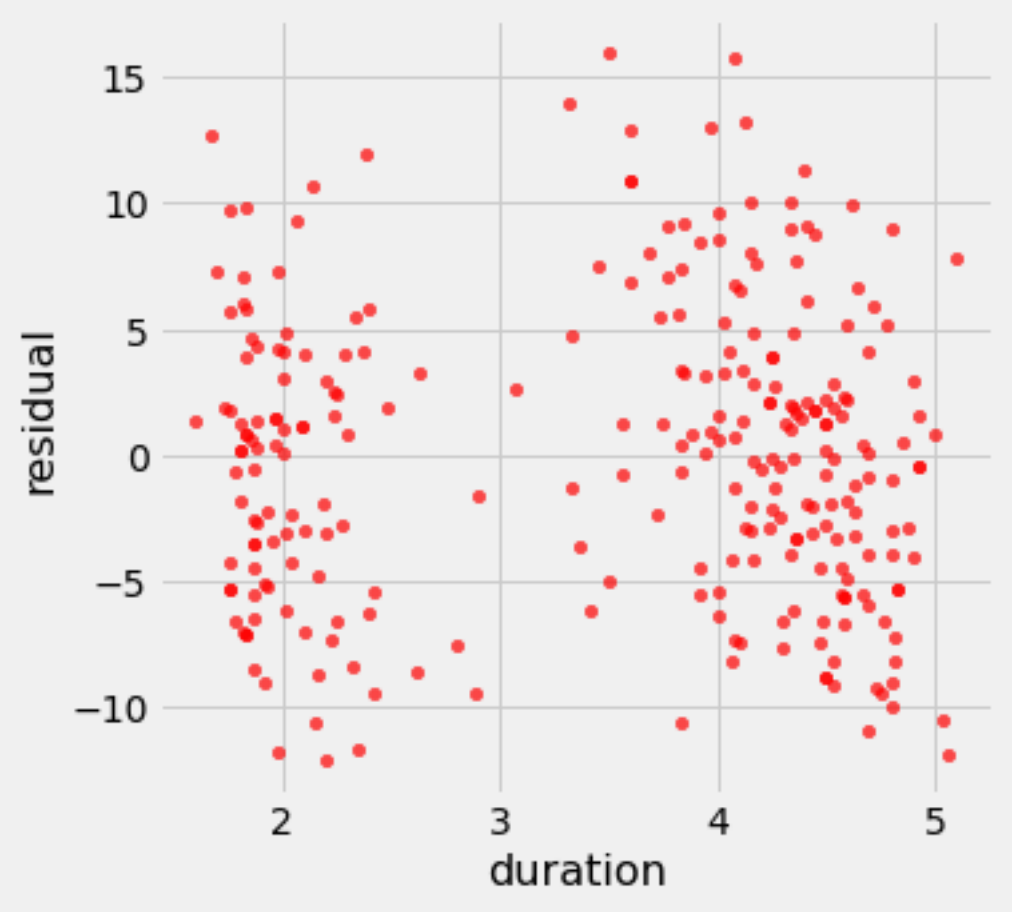
Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

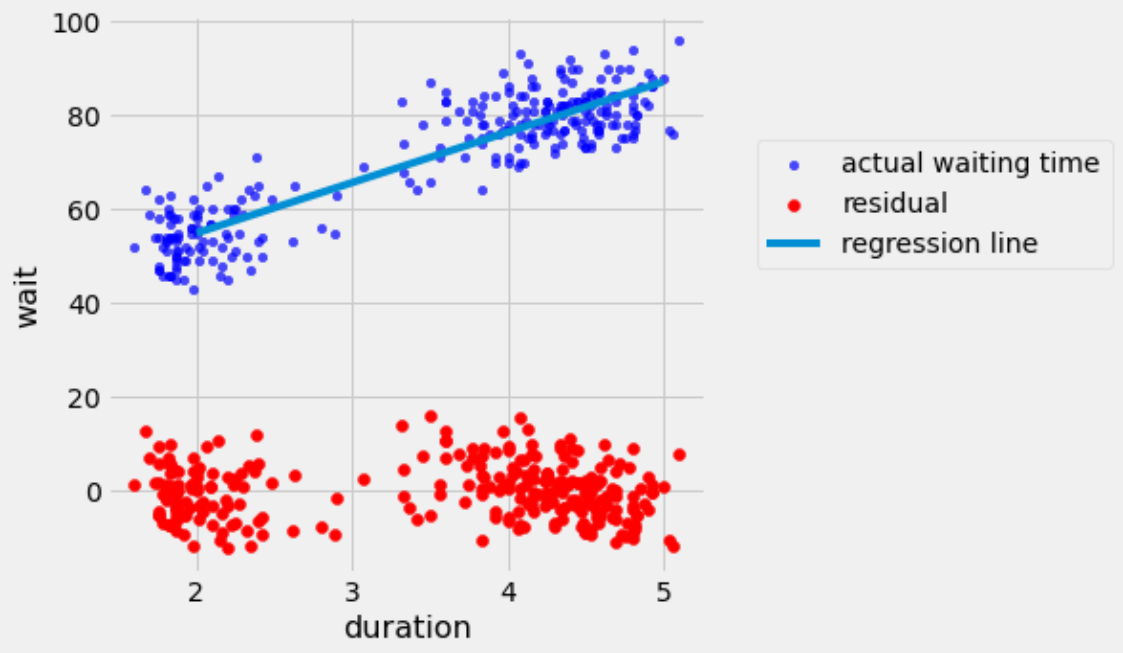
[75]:
```
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('lab12.ipynb', sorted(glob.glob('tests/q*.py'))))
```

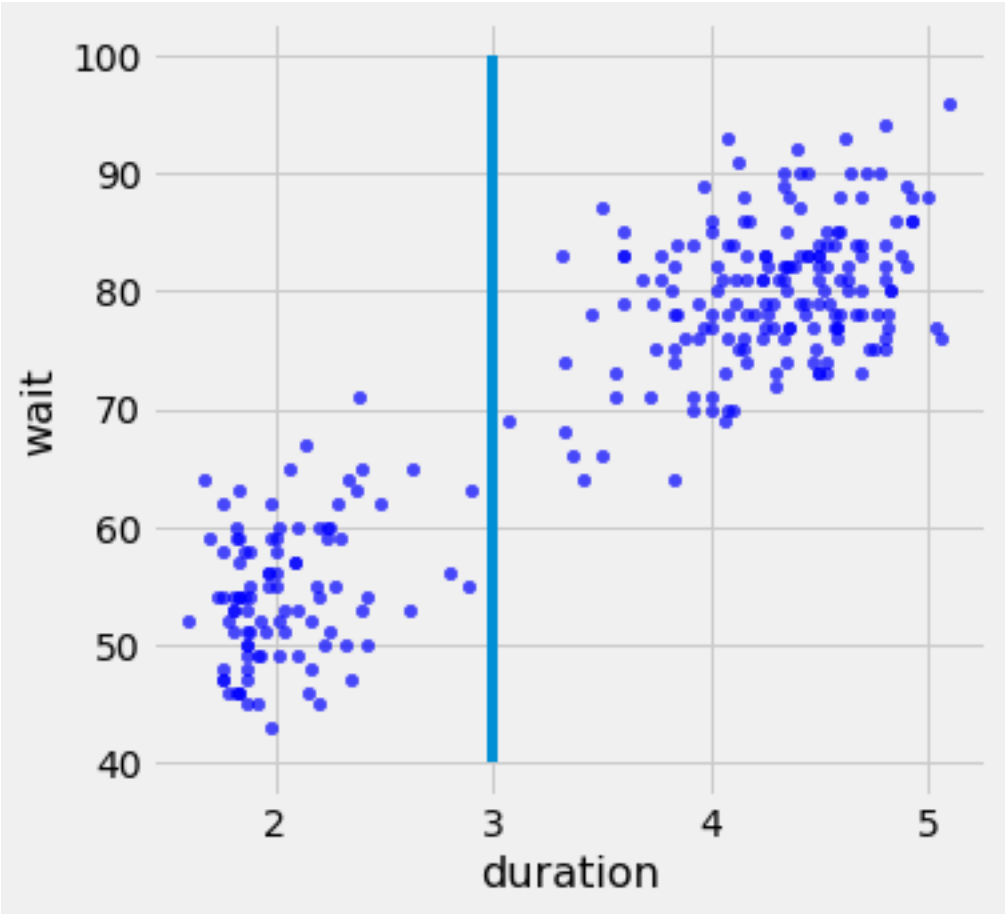
```
After an eruption lasting 2 minutes, we predict you'll wait 54.933679813020404
minutes until the next eruption.
After an eruption lasting 5 minutes, we predict you'll wait 87.12260399842098
minutes until the next eruption.
After an eruption lasting 0 minutes, we predict you'll wait 33.47439702275335
minutes until the next eruption.
After an eruption lasting 2.5 minutes, we predict you'll wait 60.29850051058717
minutes until the next eruption.
After an eruption lasting 60 minutes, we predict you'll wait 677.252880730765
minutes until the next eruption.
For points below 3, r is 0.2901895264925431 ; for points above 3, r is
0.3727822255707511
['tests/q2_1.py', 'tests/q3_1.py', 'tests/q3_2.py', 'tests/q3_3.py',
'tests/q4_1.py', 'tests/q5_1.py', 'tests/q5_2.py', 'tests/q5_3.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7f40d0ffc4d0>

Question 2:

<gofer.ok.OKTestsResult at 0x7f40d0ed4490>

Question 3:

<gofer.ok.OKTestsResult at 0x7f40d0dbe4d0>

Question 4:

<gofer.ok.OKTestsResult at 0x7f40d0dbe390>

Question 5:

<gofer.ok.OKTestsResult at 0x7f40d0dc9c50>

Question 6:

<gofer.ok.OKTestsResult at 0x7f40d0d15d10>

Question 7:

<gofer.ok.OKTestsResult at 0x7f40d0fa95d0>

Question 8:

<gofer.ok.OKTestsResult at 0x7f40d0d3d510>

1.0
```

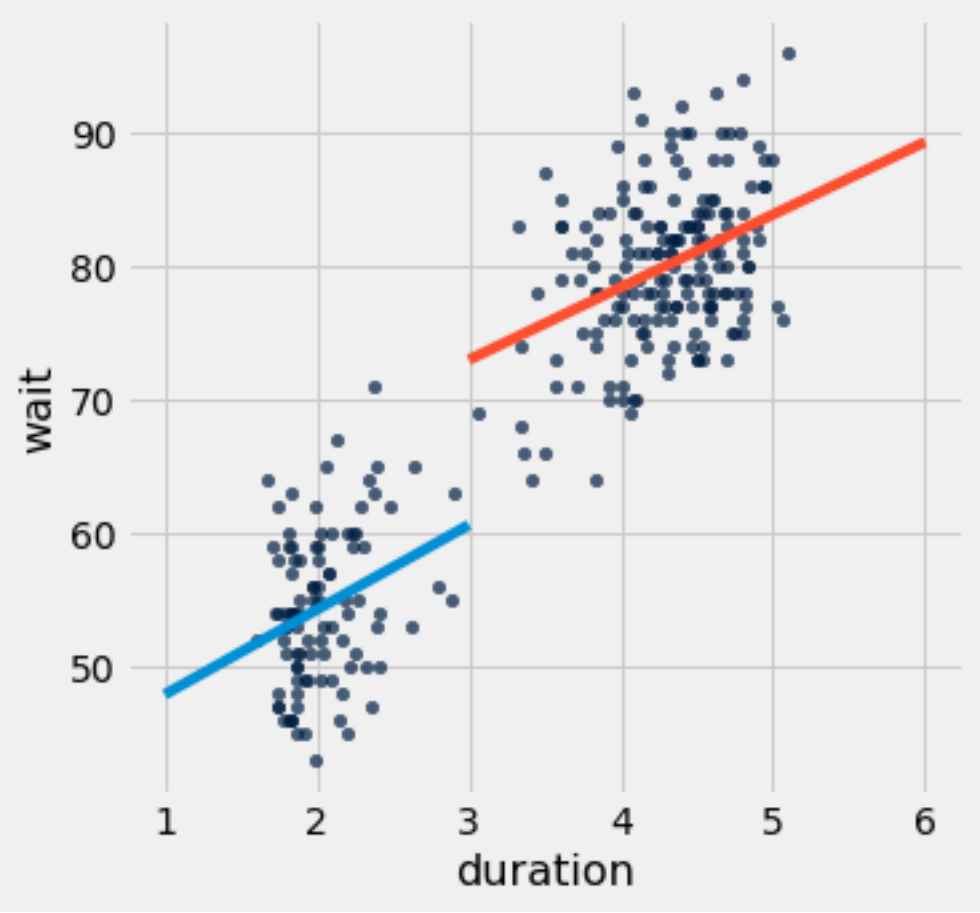

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
wait (standard units)
−2
−1
0
1
2
duration (standard units)
data
regression line

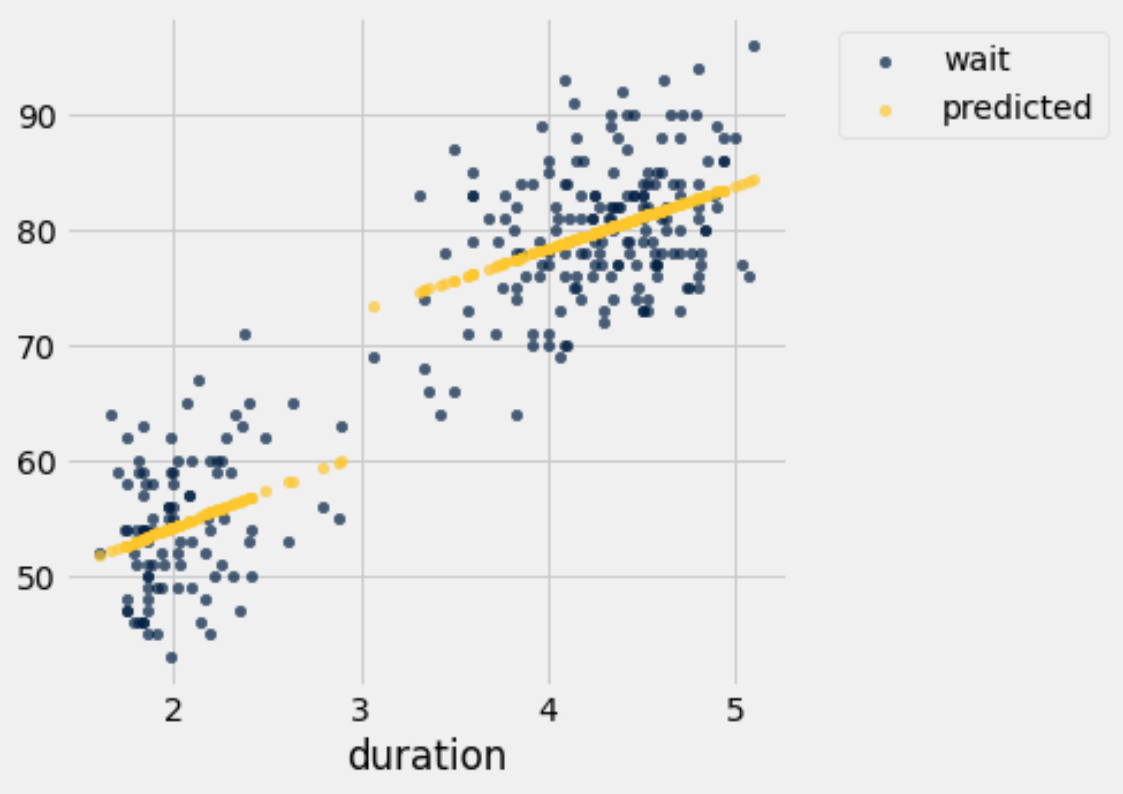

90
80
70
60
50
wait
2
3
4
5
duration
data
regression line

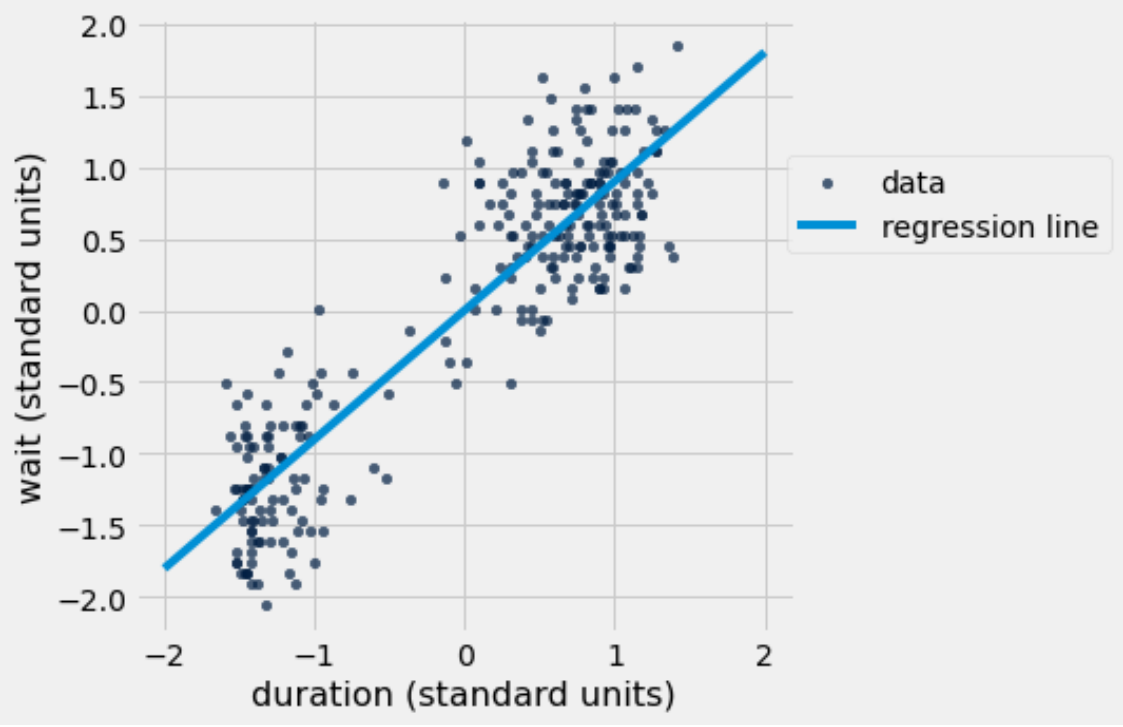

15
10
5
0
−5
−10
residual
2
3
4
5
duration

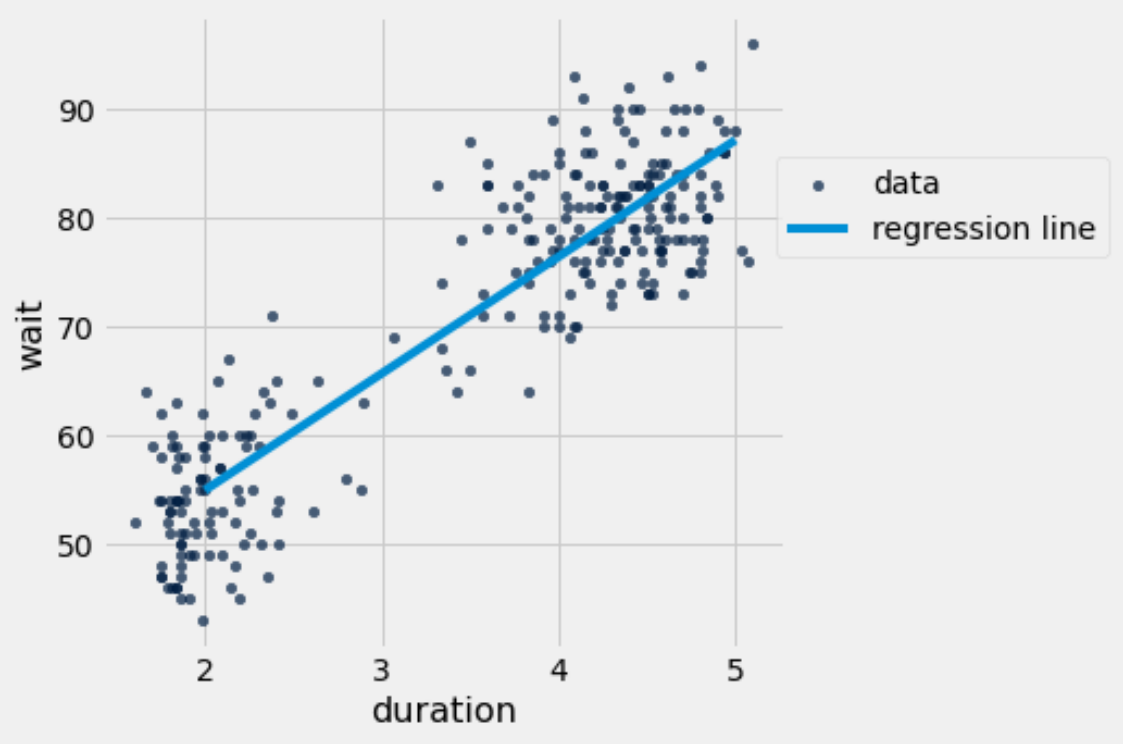

100
80
60
40
20
0
wait
2
3
4
5
duration
actual waiting time
residual
regression line

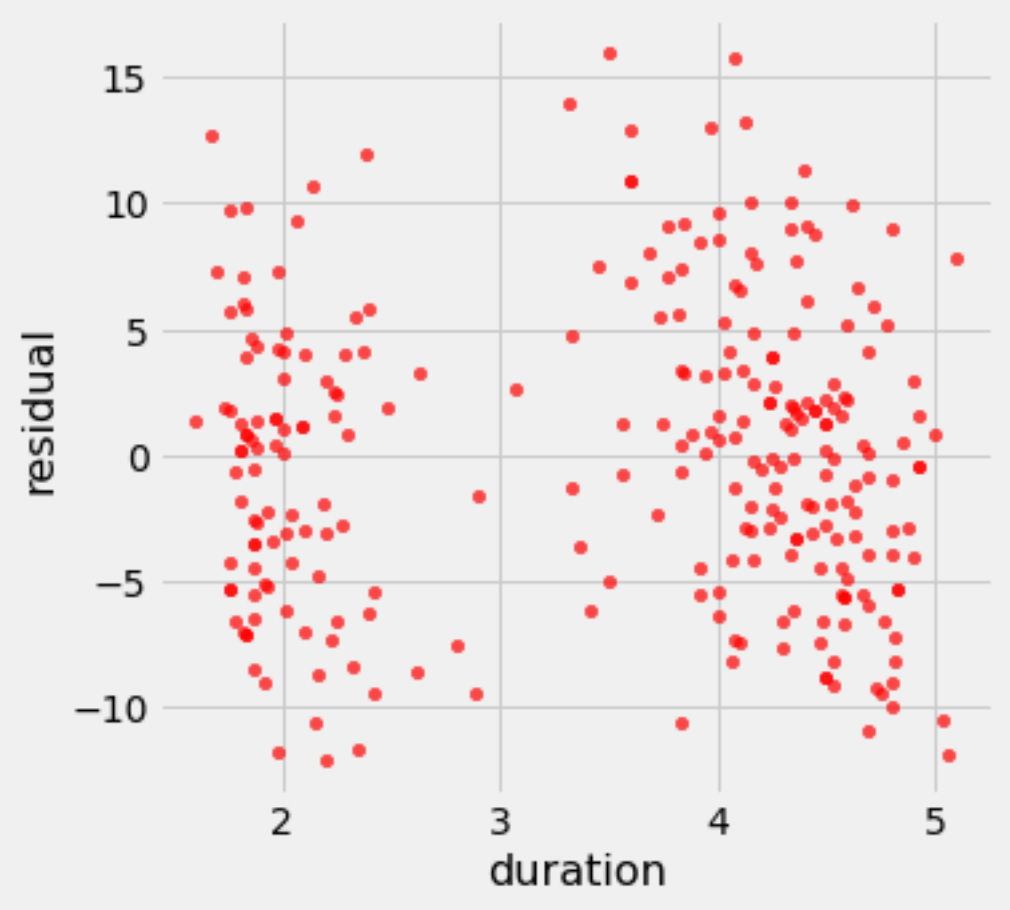

100
90
80
70
60
50
40
wait
2
3
4
5
duration

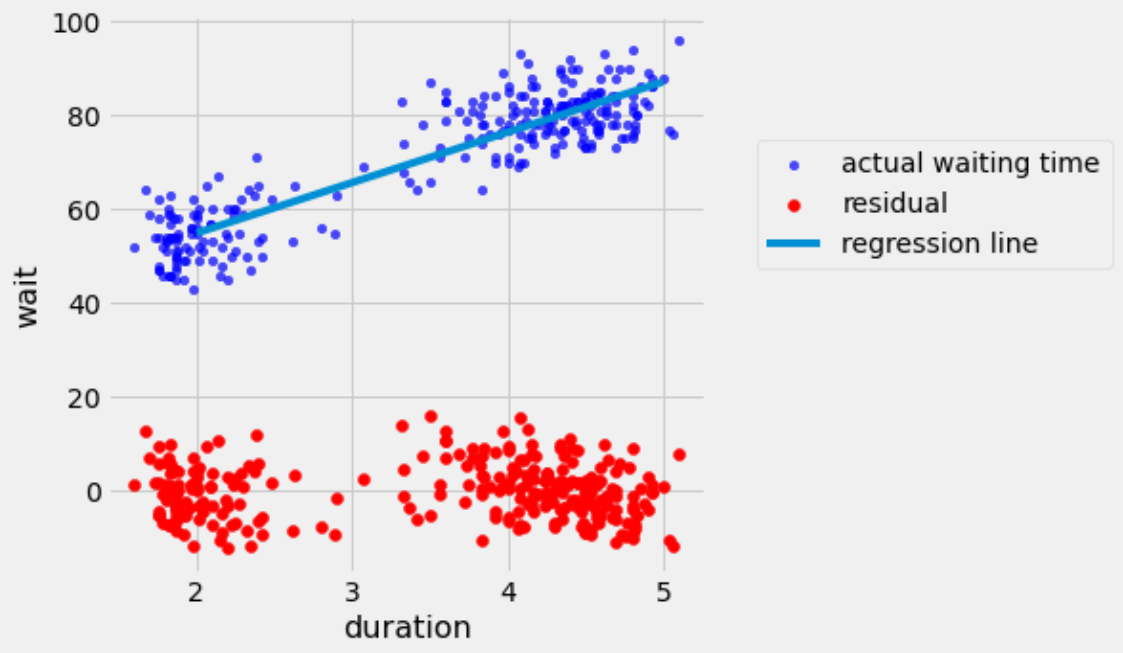

90
80
70
60
50
wait
1
2
3
4
5
6
duration

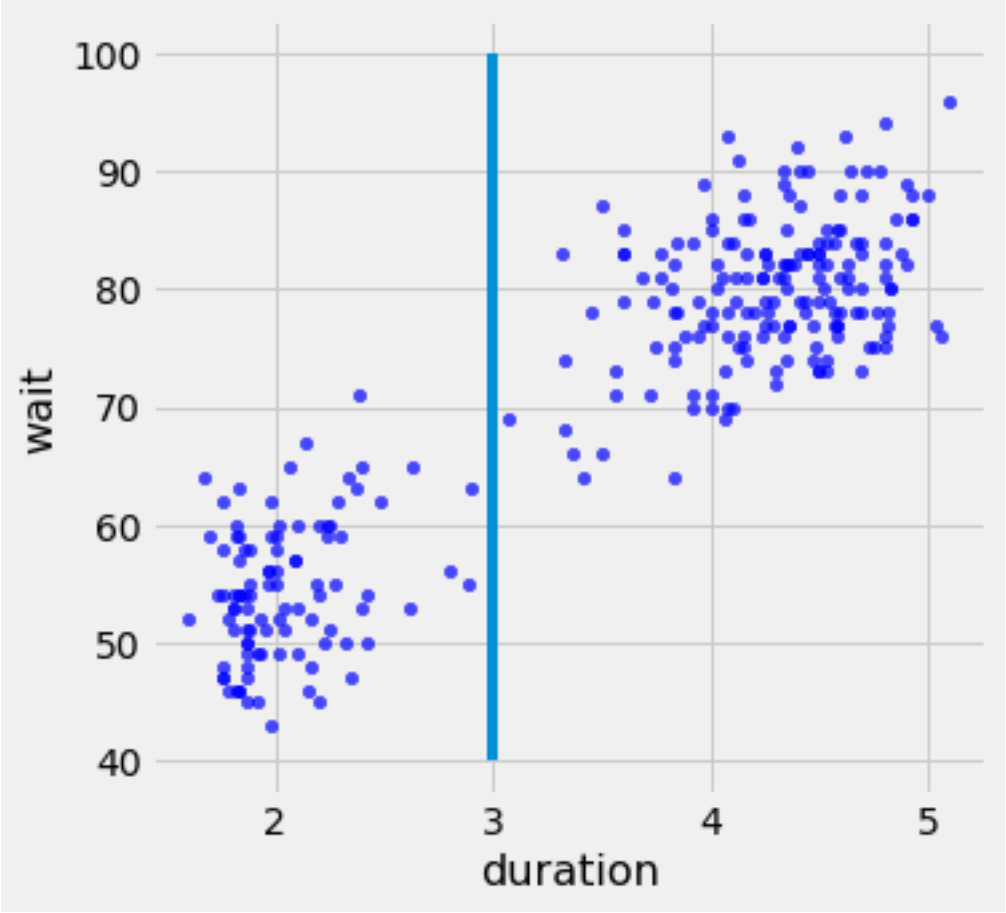

wait
predicted
90
80
70
60
50
2
3
4
5
duration

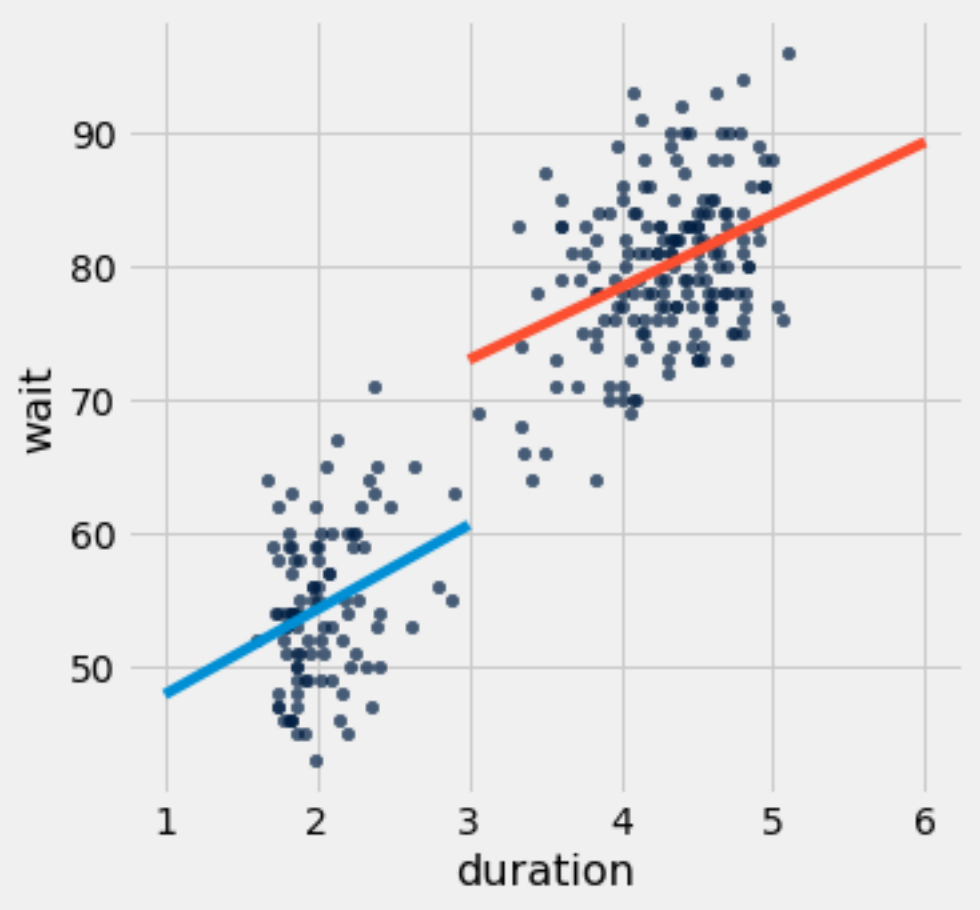

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
wait (standard units)
−2
−1
0
1
2
duration (standard units)
data
regression line

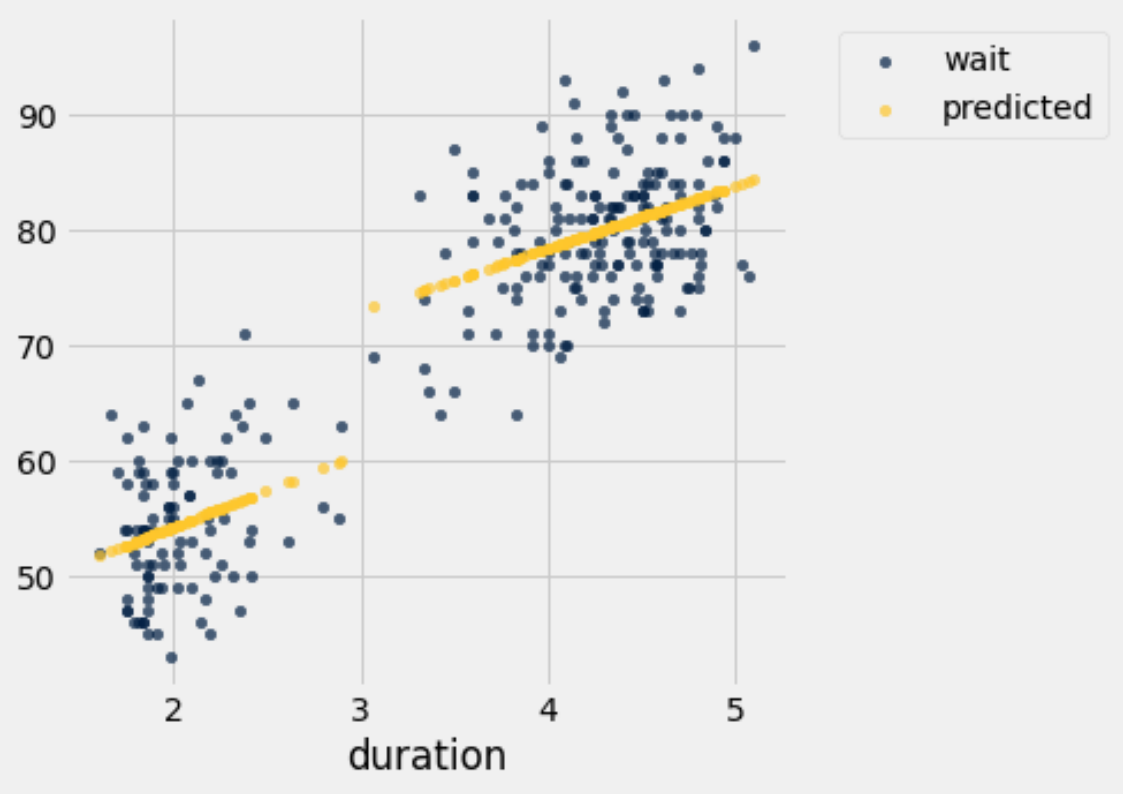

90
80
70
60
50
wait
2
3
4
5
duration
data
regression line


15
10
5
0
−5
−10
residual
2
3
4
5
duration


100
80
60
40
20
0
wait
2
3
4
5
duration
actual waiting time
residual
regression line


100
90
80
70
60
50
40
wait
2
3
4
5
duration


90
80
70
60
50
wait
1
2
3
4
5
6
duration



Name: Allan Gongora

Section: 0131