Lab 9: Regression

Welcome to Lab 9!

Today we will get some hands-on practice with linear regression. You can find more information about this topic in [section 15.2 (https://www.inferentialthinking.com/chapters/15/2/Regression_Line.html#the-regression-line)](https://www.inferentialthinking.com/chapters/15/2/Regression_Line.html#the-regression-line).

In [1]:

```
# 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)

# These lines load the tests.
from client.api.notebook import Notebook
ok = Notebook('lab08.ok')
_ = ok.submit()
```

```
===================================================================
==
Assignment: Regression
OK, version v1.14.20
===================================================================
==



Saving notebook... No valid file sources found

ERROR  | auth.py:102 | {'error': 'invalid_grant'}

Performing authentication
Please enter your bCourses email:
Successfully logged in as austenzhu@berkeley.edu
Submit... 0.0% complete
Could not submit: Assignment does not exist
Backup... 0.0% complete
Could not backup: Assignment does not exist
```

1. How Faithful is Old Faithful?

Old Faithful is a geyser in Yellowstone National Park that is famous for eruption on a fairly regular schedule. Run the cell below to see Old Faithful in action!

In [2]:
```python
# For the curious: this is how to display a YouTube video in a
# Jupyter notebook.  The argument to YouTubeVideo is the part
# of the URL (called a "query parameter") that identifies the
# video.  For example, the full URL for this video is:
#   https://www.youtube.com/watch?v=wE8NDuzt8eg
from IPython.display import YouTubeVideo
YouTubeVideo("wE8NDuzt8eg")
```

Out[2]:

Some of Old Faithful's eruptions last longer than others. Whenever there is a long eruption, it is usually followed by an even longer wait before the next eruption. If you visit Yellowstone, you might want to predict when the next eruption will happen, so that you can see the rest of the park instead of waiting by the geyser.

Today, we will use a dataset on eruption durations and waiting times to see if we can make such predictions accurately with 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.

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

Out[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

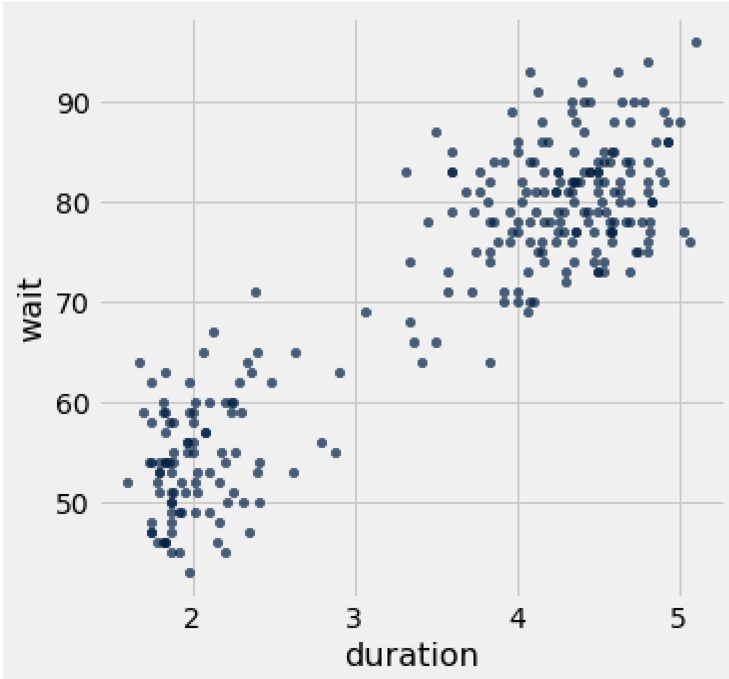
... (262 rows omitted)

We would like to use linear regression to make predictions, but that won't work well if the data aren't roughly linearly related. To check that, we should look at the data.

Question 1.1. Make a scatter plot of the data. It's conventional to put the column we want to predict on the vertical axis and the other column on the horizontal axis.

```
BEGIN QUESTION
name: q1_1
```

In [4]:
```
faithful.scatter("duration") #SOLUTION
```



Question 1.2. Are eruption duration and waiting time roughly linearly related based on the scatter plot above? Is this relationship positive?

```
BEGIN QUESTION
name: q1_2
```

SOLUTION: Yes, they are roughly linearly related. The eruption durations seem to cluster; there are a bunch of short eruptions and a bunch of longer ones. But the data in both clusters fall roughly on a line, and that's what's important when it comes to predicting waiting times with linear regression. The relationship is positive, meaning that longer eruptions have longer waiting times, as we claimed.

We're going to continue with the assumption that they are linearly related, so it's reasonable to use linear regression to analyze this data.

We'd next like to plot the data in standard units. If you don't remember the definition of standard units, textbook section [14.2 (https://www.inferentialthinking.com/chapters/14/2/Variability.html#standard-units)](https://www.inferentialthinking.com/chapters/14/2/Variability.html#standard-units) might help!

Question 1.3. Compute the mean and standard deviation of the eruption durations and waiting times. **Then** create a table called `faithful_standard` containing the eruption durations and waiting times in standard units. The columns should be named `duration (standard units)` and `wait (standard units)`.

```
BEGIN QUESTION
name: q1_3
```

In [5]:

```
# BEGIN SOLUTION NO PROMPT
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") - dura
    "wait (standard units)", (faithful.column("wait") - wait_mean) /
faithful_standard
# END SOLUTION
""" # BEGIN PROMPT
duration_mean = ...
duration_std = ...
wait_mean = ...
wait_std = ...

faithful_standard = Table().with_columns(
    "duration (standard units)", ...,
    "wait (standard units)", ...)
faithful_standard
"""; # END PROMPT
```

In [6]:

```
# TEST
abs(sum(faithful_standard.column(0))) <= 1e-8
```

Out[6]: True

In [7]:

```
# TEST
int(round(duration_std))
```

Out[7]: 1

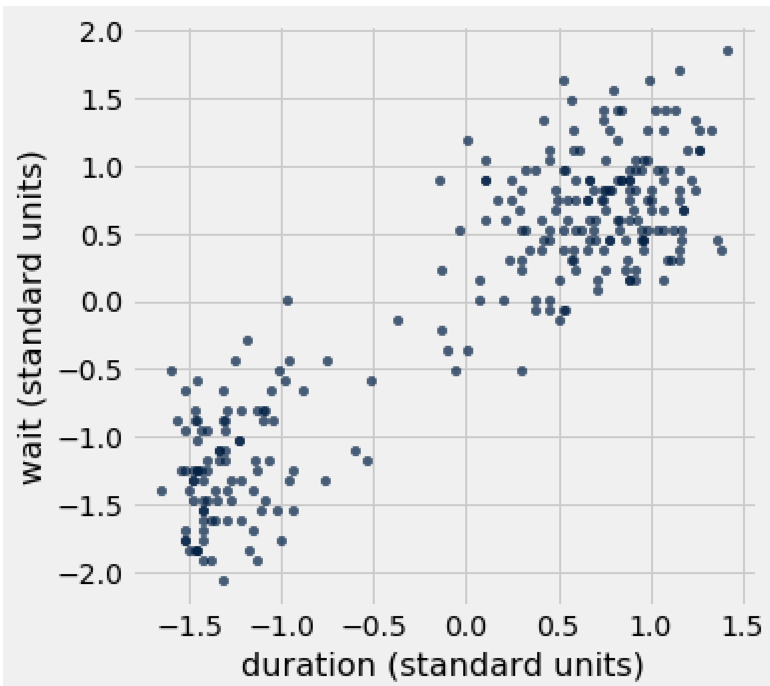
In [8]:
```
# TEST
int(round(wait_std))
```

Out[8]: 14

Question 1.4. Plot the data again, but this time in standard units.

```
BEGIN QUESTION
name: q1_4
```

In [9]:
```
faithful_standard.scatter(0) #SOLUTION
```



You'll notice that this plot looks the same as the last one! However, the data and axes are scaled differently. So it's important to read the ticks on the axes.

Question 1.5. Among the following numbers, which would you guess is closest to the correlation between eruption duration and waiting time in this dataset?

1. -1
2. 0
3. 1

Assign `correlation` to the number corresponding to your guess.

```
BEGIN QUESTION
name: q1_5
```

In [10]:
```
correlation = 3 # SOLUTION
```

In [11]:
```
# TEST
correlation == 3
```

Out[11]: True

Question 1.6. Compute the correlation `r`.

Hint: Use `faithful_standard`. Section [15.1 (https://www.inferentialthinking.com/chapters/15/1/Correlation.html#calculating-r)](https://www.inferentialthinking.com/chapters/15/1/Correlation.html#calculating-r) explains how to do this.

```
BEGIN QUESTION
name: q1_6
```

In [12]:
```
r = np.mean(faithful_standard.column(0) * faithful_standard.column(1)
r
```

Out[12]: 0.9008111683218132

In [13]:
```
# TEST
"%.2f" % round(r,2)
```

Out[13]: '0.90'

2. The regression line

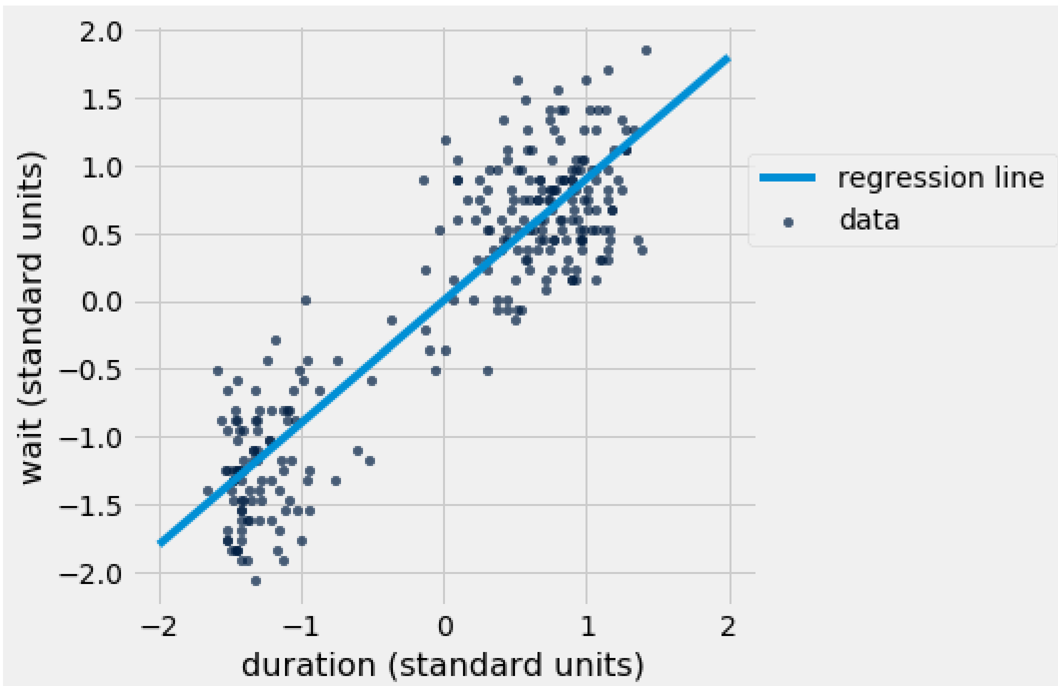
Recall that the correlation is the **slope of the regression line when the data are put in standard units**.

The next cell plots the regression line in standard units:

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

Then, it plots the data in standard units again, for comparison.

In [14]:
```
def plot_data_and_line(dataset, x, y, point_0, point_1):
    """Makes a scatter plot of the dataset, along with a line passing
    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. Calculate the slope of the regression line in original units, and assign it to `slope`.

(If the "stretching" explanation is unintuitive, consult section [15.2 (https://www.inferentialthinking.com/chapters/15/2/Regression_Line.html#the-equation-of-the-regression-line)](https://www.inferentialthinking.com/chapters/15/2/Regression_Line.html#the-equation-of-the-regression-line) in the textbook.)

```
BEGIN QUESTION
name: q2_1
```

In [15]:
```
slope = (wait_std/duration_std) * r #SOLUTION
slope
```

Out[15]: 10.729641395133527

In [16]:
```
# TEST
(slope*13 - 100)/98 <= 0.5
```

Out[16]: True

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 therefore:

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

The rearranged equation becomes:

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

Question 2.2. Calculate the intercept in original units and assign it to `intercept`.

```
BEGIN QUESTION
name: q2_2
```

In [17]:
```
intercept = slope*(-duration_mean) + wait_mean
intercept
```

Out[17]: 33.47439702275335

In [18]:
```
# TEST
(18 + intercept*(-4)) / 201 >= 0.3
```

Out[18]: False

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.

```
BEGIN QUESTION
name: q3_1
```

In [19]:

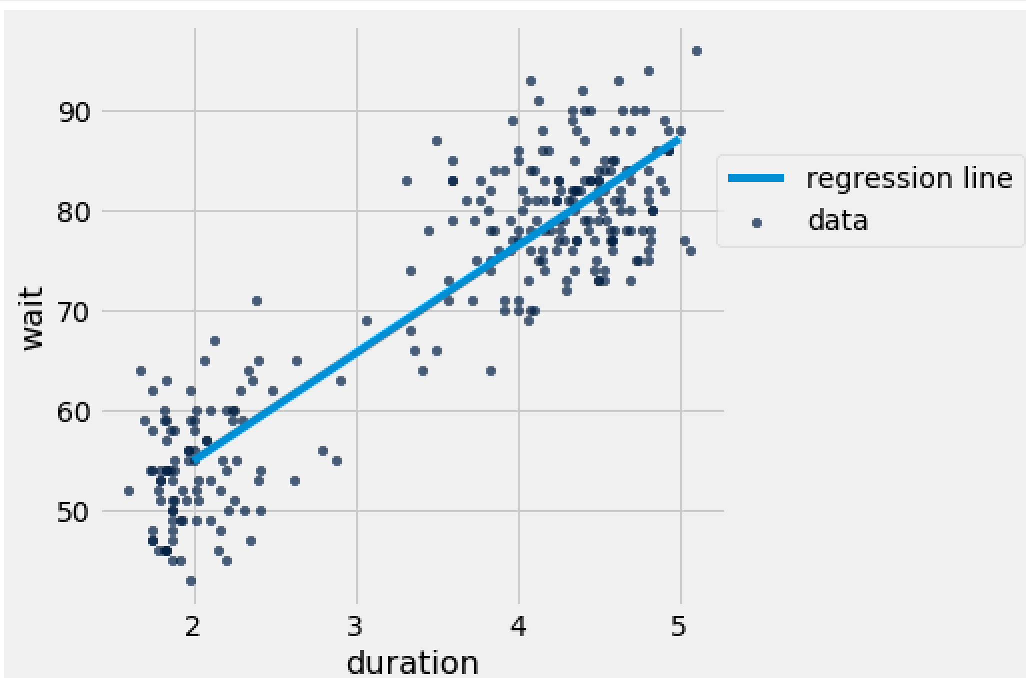
```
two_minute_predicted_waiting_time = slope*2 + intercept #SOLUTION
five_minute_predicted_waiting_time = slope*5 + intercept #SOLUTION

# Here is a helper function to print out your predictions.
# Don't modify the code below.
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 54.9336
79813020404 minutes until the next eruption.
After an eruption lasting 5 minutes, we predict you'll wait 87.1226
0399842098 minutes until the next eruption.
```

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

In [20]:

```
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.

```
BEGIN QUESTION
name: q3_2
```

In [21]:
```
faithful_predictions = faithful.with_column("predicted wait", slope*
faithful_predictions
```

Out[21]:

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)

In [22]:
```
# TEST
# Make sure your column labels are correct.
set(faithful_predictions.labels) == set(['duration', 'wait', 'predic
```

Out[22]: True

In [23]:
```
# TEST
abs(1 - np.mean(faithful_predictions.column(2))/100) <= 0.35
```

Out[23]: True

Question 3.3. How close were we? Compute the *residual* for each eruption in the dataset. The residual is the actual waiting time minus the predicted waiting time. Add the residuals to `faithful_predictions` as a new column called `residual` and name the resulting table `faithful_residuals`.

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

```
BEGIN QUESTION
name: q3_3
```

In [27]:
```
faithful_residuals = faithful_predictions.with_column("residual", fa
faithful_residuals
```

Out[27]:

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

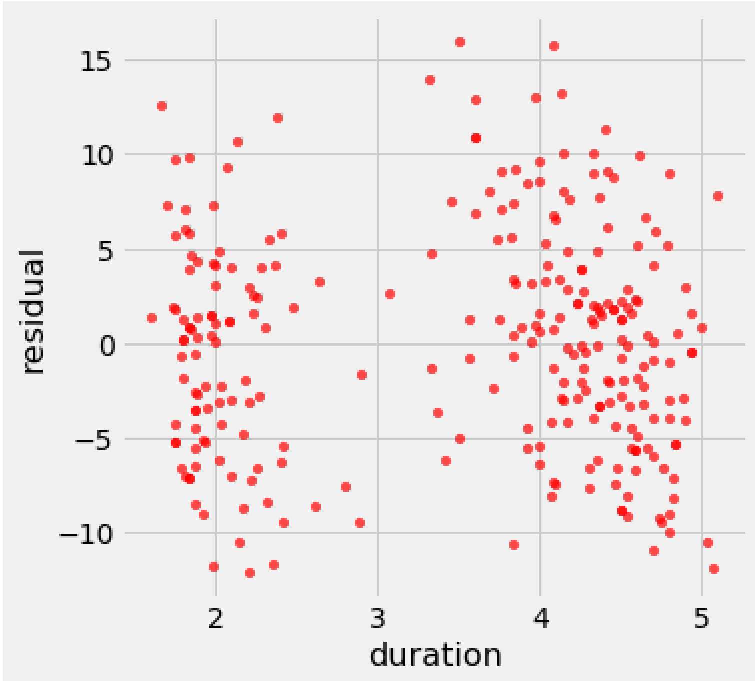
... (262 rows omitted)

In [28]:
```
# TEST
abs(sum(faithful_residuals.column(3))) <= 1e-8
```

Out[28]: True

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.

In [29]:
```
faithful_residuals.scatter("duration", "residual", color="r")
```



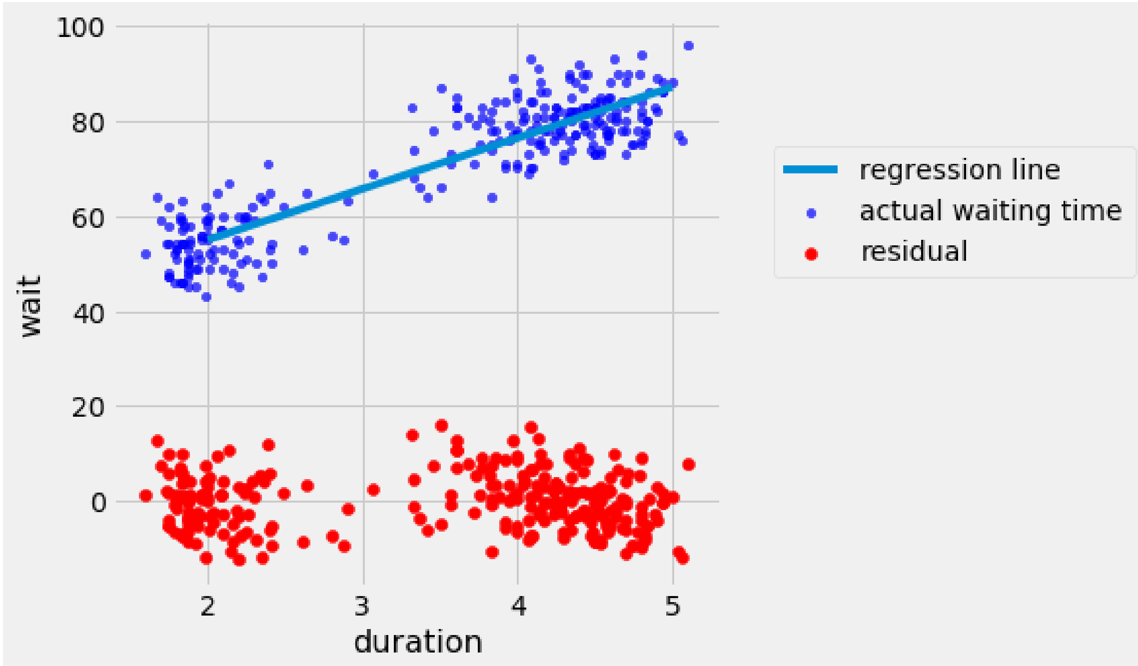
There isn't really a pattern in the residuals, which confirms that it was reasonable to try linear regression. It's true that there are two separate clouds; the eruption durations seemed to fall into two distinct clusters. But that's just a pattern in the eruption durations, not a pattern in the relationship between eruption durations and waiting times.

4. How accurate are different predictions?

Earlier, you should have found that the correlation is fairly close to 1, so the line fits fairly well on the training data. That means 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:

In [30]:
```
faithful_residuals.scatter("duration", "wait", label="actual waiting
plots.scatter(faithful_residuals.column("duration"), faithful_residua
plots.plot([2, 5], [two_minute_predicted_waiting_time, five_minute_pr
plots.legend(bbox_to_anchor=(1.7,.8));
```



However, unless you have a strong reason to believe that the linear regression model is true, you should be wary of applying your prediction model to data that are very different from the training data.

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?

```
BEGIN QUESTION
name: q4_1
```

In [31]:
```
zero_minute_predicted_waiting_time = intercept #SOLUTION
two_point_five_minute_predicted_waiting_time = slope * 2.5 + intercep
hour_predicted_waiting_time = slope * 60 + intercept #SOLUTION

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 33.4743
9702275335 minutes until the next eruption.
After an eruption lasting 2.5 minutes, we predict you'll wait 60.29
850051058717 minutes until the next eruption.
After an eruption lasting 60 minutes, we predict you'll wait 677.25
2880730765 minutes until the next eruption.
```

In [32]:
```
# TEST
12 - zero_minute_predicted_waiting_time*1.4/4 <= 0.35
```

Out[32]: True

In [33]:
```
# TEST
2 - two_point_five_minute_predicted_waiting_time/35 <= 0.4
```

Out[33]: True

In [34]:
```
# TEST
(26 - hour_predicted_waiting_time/30)/10 <= 0.43
```

Out[34]: True

Question 2. For each prediction, state whether you think it's reliable and explain your reasoning.

```
BEGIN QUESTION
name: q4_2
```

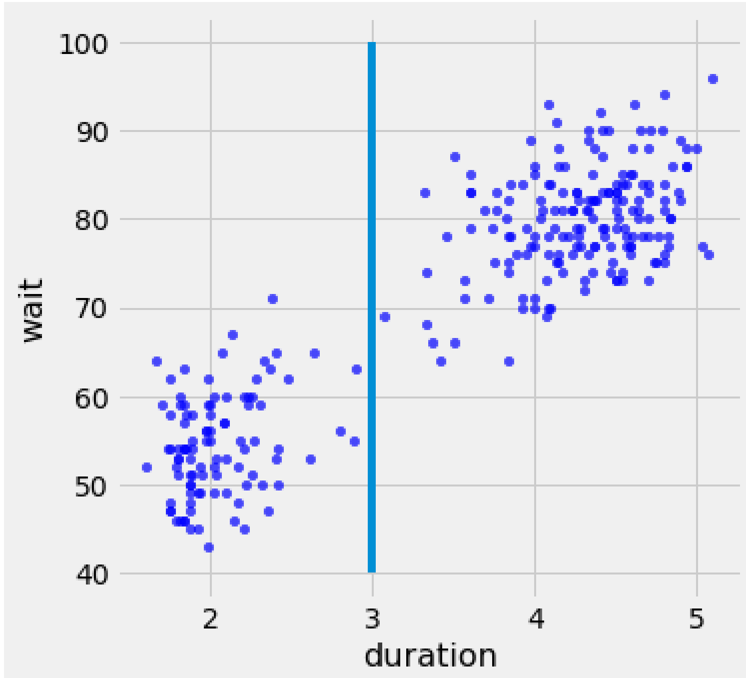
SOLUTION: The prediction for 2.5 is believable, since the dataset has eruptions that are *around* that long. A 0 minute eruption is physically impossible, so the predicted waiting time is meaningless. A 60 minute eruption might be possible, but since we never saw one nearly that long, it would probably be very different in character than the ones in `faithful` . So we probably shouldn't trust that prediction, either.

5. Divide and Conquer

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.

In [35]:

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



The `standardize` function from lecture appears below, which takes in a table with numerical columns and returns the same table with each column converted into standard units.

In [36]:

```
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 sta
    t_su = Table()
    for label in t.labels:
        t_su = t_su.with_column(label + ' (su)', standard_units(t.co
    return t_su
```

Question 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`.

```
BEGIN QUESTION
name: q5_1
```

In [37]:
```
def reg_coeff(t):
    """Return the regression coefficient for columns 0 & 1."""
    t_su = standardize(t)
    return np.mean(t_su.column(0) * t_su.column(1)) # SOLUTION

below_3 = faithful.where('duration', are.below(3)) # SOLUTION
above_3 = faithful.where('duration', are.above(3)) # SOLUTION
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
```

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

In [38]:
```
# TEST
np.allclose([below_3_r, above_3_r], [0.290189526493, 0.372782225571]
```

Out[38]:
```
True
```

In [39]:
```
# TEST
[below_3.num_rows, above_3.num_rows]
```

Out[39]:
```
[97, 175]
```

Question 5.2. Complete the 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. The second function should use the regression line for all points with duration above 3.

```
BEGIN QUESTION
name: q5_2
```

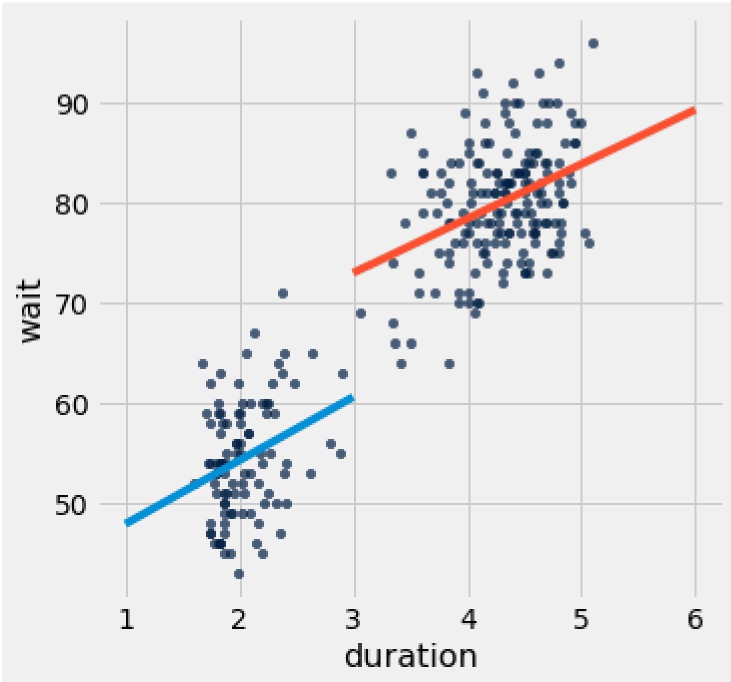
In [40]:
```
def slope_of(t, r):
    """Return the slope of the regression line for t in original uni

    Assume that column 0 contains x values and column 1 contains y va
    r is the regression coefficient for x and y.
    """
    return r * np.std(t.column(1)) / np.std(t.column(0)) # SOLUTION

def intercept_of(t, r):
    """Return the slope of the regression line for t in original uni
    s = slope_of(t, r)
    return s * (-np.mean(t.column(0))) + np.mean(t.column(1)) # SOLU

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
```

In [41]:
```
ok.grade('q5_2');
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

---------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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

In [42]:
```
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 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.

In [43]:
```
def predict_wait(duration):
    """Return the wait predicted by the appropriate one of the two re
    ...
```

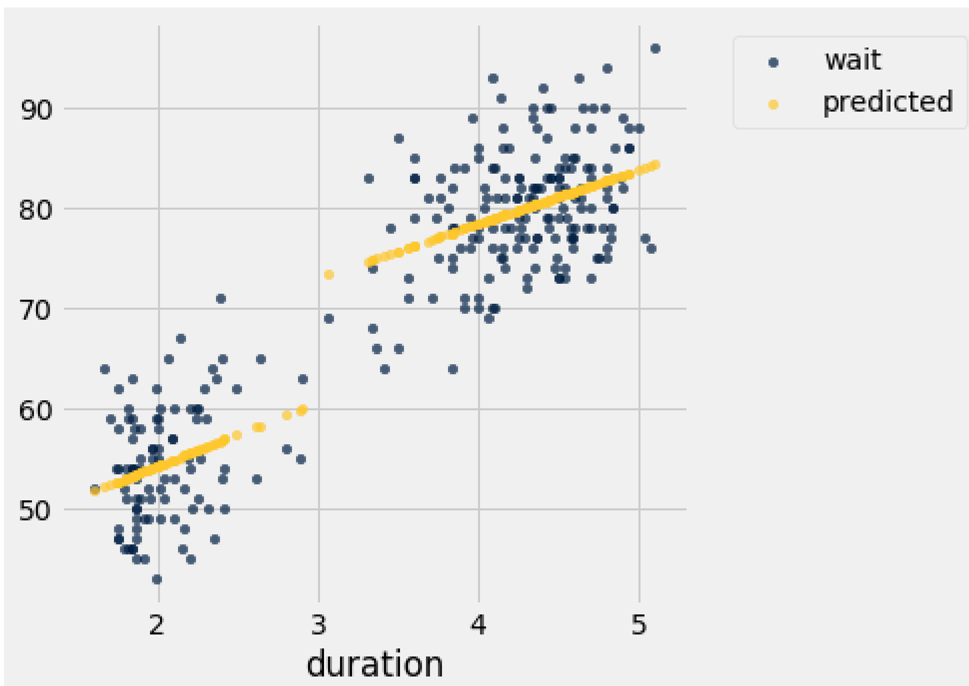
In [44]:
```
def predict_wait(duration):
    if duration < 3:
        return wait_below_3(duration)
    else:
        return wait_above_3(duration)
```

In [45]:
```
ok.grade('q5_3');
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

---------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

The predicted wait times for each point appear below.

In [46]:
```
faithful.with_column('predicted', faithful.apply(predict_wait, 'dura
```



Question 4. Do you think the predictions produced by `predict_wait` would be more or less accurate than the predictions from the regression line you created in section 2? How could you tell?

SOLUTION: More accurate, because each line is specific to the values in its cluster. To verify, we could measure the average magnitude of the residual values.

In [47]:

```python
# For your convenience, you can run this cell to run all the tests a
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q
print("Finished running all tests.")
```

```
Running all tests...
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests
```

```
-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
Running tests

-------------------------------------------------------------------
--
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

Finished running all tests.
```

In [48]:
```
# Run this cell to submit your work *after* you have passed all of t
# It's ok to run this cell multiple times. Only your final submissio

_ = ok.submit()
```

```
Saving notebook... No valid file sources found
Submit... 0.0% complete
Could not submit: Assignment does not exist
Backup... 0.0% complete
Could not backup: Assignment does not exist
```

In [ ]: