hw04-solution

March 31, 2020

1 Homework 4: Probability and Sampling

Reading: Textbook chapter 8.

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

[1]:
```
# Don't change this cell; just run it.

import numpy as np
from datascience import *

%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')

from client.api.notebook import Notebook
ok = Notebook('hw04.ok')
_ = ok.auth(inline=True)
```

```
=====================================================================
Assignment: Homework 4: Probability and Sampling
OK, version v1.14.20
=====================================================================

Successfully logged in as m.zareei@tec.mx
```

Important: The `ok` tests don't usually tell you that your answer is correct. More often, they help catch careless mistakes. It's up to you to ensure that your answer is correct. If you're not sure, ask someone (not for the answer, but for some guidance about your approach).

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission.

[26]:
```
_ = ok.submit()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>


Saving notebook… Could not automatically save 'hw04.ipynb'
Make sure your notebook is correctly named and saved before submitting to OK!
Making a best attempt to submit latest 'hw04.ipynb', last modified at Wed Jan 29
13:47:22 2020
Submit… 100% complete
Submission successful for user: m.zareei@tec.mx
URL: https://okpy.org/tec/tc2031/sp20/hw04/submissions/Lv6w4g
```

1.1 1. Sampling Basketball Players

This exercise uses salary data and game statistics for basketball players from the 2014-2015 NBA season. The data were collected from basketball-reference and spotrac.

Run the next cell to load the two datasets.

[3]:
```
player_data = Table.read_table('player_data.csv')
salary_data = Table.read_table('salary_data.csv')
player_data.show(3)
salary_data.show(3)
```

```
<IPython.core.display.HTML object>


<IPython.core.display.HTML object>
```

Question 1. We would like to relate players' game statistics to their salaries. Compute a table called `full_data` that includes one row for each player who is listed in both `player_data` and `salary_data`. It should include all the columns from `player_data` and `salary_data`, except the `"PlayerName"` column.

[4]:
```
full_data = player_data.join('Name', salary_data, 'PlayerName') #SOLUTION
full_data
```

[4]:
```
Name            | Age  | Team | Games | Rebounds | Assists | Steals | Blocks |
Turnovers | Points | Salary
A.J. Price      | 28   | TOT  | 26    | 32       | 46      | 7      | 0      |
14        | 133    | 62552
Aaron Brooks    | 30   | CHI  | 82    | 166      | 261     | 54     | 15     |
157       | 954    | 1145685
Aaron Gordon    | 19   | ORL  | 47    | 169      | 33      | 21     | 22     |
38        | 243    | 3992040
Adreian Payne   | 23   | TOT  | 32    | 162      | 30      | 19     | 9      |
44        | 213    | 1855320
```

```
Al Horford      | 28   | ATL  | 76    | 544      | 244     | 68     | 98     |
100       | 1156   | 12000000
Al Jefferson    | 30   | CHO  | 65    | 548      | 113     | 47     | 84     |
68        | 1082   | 13666667
Al-Farouq Aminu | 24   | DAL  | 74    | 342      | 59      | 70     | 62     |
55        | 412    | 1100602
Alan Anderson   | 32   | BRK  | 74    | 204      | 83      | 56     | 5      |
60        | 545    | 1276061
Alec Burks      | 23   | UTA  | 27    | 114      | 82      | 17     | 5      |
52        | 374    | 3034356
Alex Kirk       | 23   | CLE  | 5     | 1        | 1       | 0      | 0      | 0
| 4      | 507336
… (482 rows omitted)
```

[5]:
```
_ = ok.grade('q1_1')
```

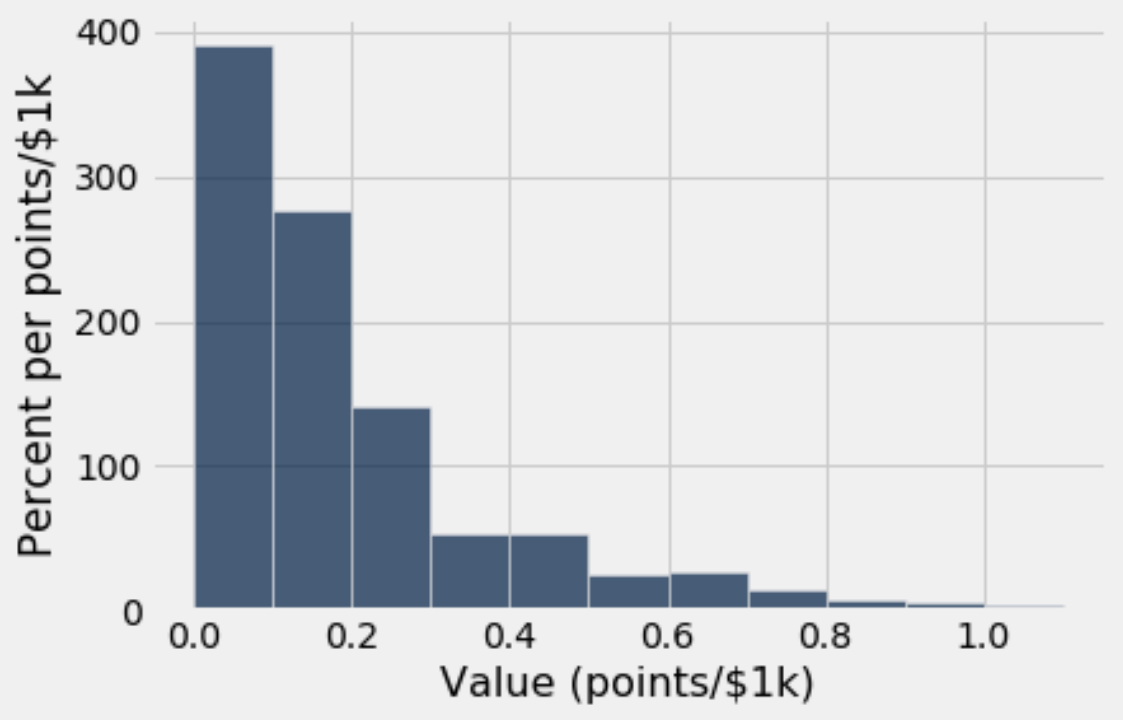
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

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

Basketball team managers would like to hire players who perform well but don't command high salaries. From this perspective, a very crude measure of a player's *value* to their team is the number of points the player scored in a season for every **$1000 of salary** (*Note*: the `Salary` column is in dollars, not thousands of dollars). For example, Al Horford scored 1156 points and has a salary of 12,000 thousands of dollars, so his value is $\frac{1156}{12000}$.

Question 2. Create a table called `full_data_with_value` that's a copy of `full_data`, with an extra column called `"Value"` containing each player's value (according to our crude measure). Then make a histogram of players' values. **Specify bins that make the histogram informative, and don't forget your units!**

[6]:
```
full_data_with_value = full_data.with_column("Value", full_data.
 ↪column("Points") / full_data.column("Salary") * 1000) #SOLUTION
full_data_with_value.hist("Value", bins=np.arange(0, 1.2, .1), unit="points/
 ↪$1k") #SOLUTION
```



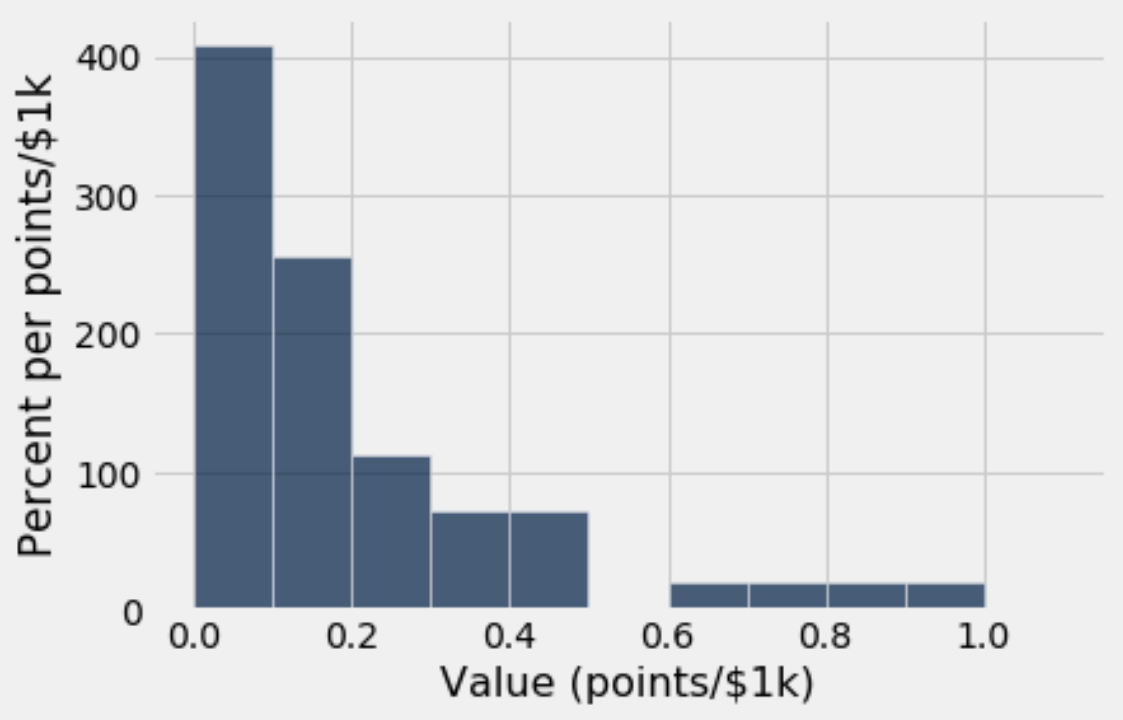
Now suppose we weren't able to find out every player's salary. (Perhaps it was too costly to interview each player.) Instead, we have gathered a *simple random sample* of 100 players' salaries. The cell below loads those data.

[7]:
```
sample_salary_data = Table.read_table("sample_salary_data.csv")
sample_salary_data.show(3)
```

```
<IPython.core.display.HTML object>
```

Question 3. Make a histogram of the values of the players in `sample_salary_data`, using the same method for measuring value we used in question 2. **Use the same bins, too.** *Hint:* This will take several steps.

[8]:
```
# Use this cell to make your histogram.
sample_data = player_data.join('Name', sample_salary_data, 'PlayerName')
sample_data_with_value = sample_data.with_column("Value", sample_data.
 ↪column("Points") / sample_data.column("Salary") * 1000)
sample_data_with_value.hist("Value", bins=np.arange(0, 1.2, .1), unit="points/
 ↪$1k")
```



Now let us summarize what we have seen. To guide you, we have written most of the summary already.

Question 4. Complete the statements below by filling in the [SQUARE BRACKETS]:

The plot in question 2 displayed a(n) **probability** histogram of the population of **492** players. The areas of the bars in the plot sum to **1**.

The plot in question 3 displayed a(n) **empirical** histogram of the population of **100** players. The areas of the bars in the plot sum to **1**.

Question 5. For which range of values does the plot in question 3 better depict the distribution of the **population's player values**: 0 to 0.5, or above 0.5?

SOLUTION: The sample histogram and population histogram look similar for values below 0.5. For values above 0.5, the sample histogram looks less accurate. The players in the population with values above 0.5 are rarer, so the sample gives us a worse estimate of that part of the distribution.

1.2 2. How Many Devices?

When a company produces medical devices, it must be sure that its devices will not fail. Sampling is used ubiquitously in the medical device industry to test how well devices work.

Suppose you work at a company that produces syringes, and you are responsible for ensuring the syringes work well. After studying the manufacturing process for the syringes, you have a hunch that they have a 1% failure rate. That is, you suspect that 1% of the syringes won't work when a doctor uses them to inject a patient with medicine.

To test your hunch, you would like to find at least one faulty syringe. You hire an expert consultant who can test a syringe to check whether it is faulty. But the expert's time is expensive, so you need to avoid checking more syringes than you need to.

Important note: This exercise asks you to compute numbers that are related to probabilities. For all questions, you can calculate your answer using algebra, **or** you can write and run a simulation to compute an approximately-correct answer. (For practice, we suggest trying both.) An answer based on an appropriate simulation will receive full credit. If you simulate, use at least **5,000** trials.

Question 1. Suppose there is indeed a 1% failure rate among all syringes. If you check 20 syringes chosen at random from among all syringes, what is the chance that you find at least 1 faulty syringe? (You may assume that syringes are chosen with replacement from a population in which 1% of syringes are faulty.) Name your answer `chance_to_find_syringe`.

[9]:
```
# For your convenience, we have created a list containing
# 99 copies of the number 0 (to represent good syringes)
# and 1 copy of the number 1 (to represent a bad syringe).
# This may be useful if you run a simulation.  Feel free
# to delete it.
faultiness = np.append(0*np.arange(99), 1)

chance_to_find_syringe = 1 - (1 - 1/100)**20 # SOLUTION
chance_to_find_syringe
```

[9]:
```
0.18209306240276923
```

[7]:
```
# For completeness, here is a simulation, too.
# An array of ninety-nine 0s and a 1. The 1 represents a bad syringe.
faultiness = np.append(0*np.arange(99), 1)
syringe_table = Table().with_column("Faulty", faultiness)
num_simulations = 5000

# We encapsulate our code in a few functions. This makes
# it a little clearer, and it lets us reuse it later.

def found_fault_in_simulation(num_checks):
    """Runs one simulation in which num_checks syringes are checked.

    Returns True if at least one syringe is found to be faulty, and
    False otherwise."""
    num_faults_found = sum(syringe_table.sample(num_checks).column("Faulty"))
    return num_faults_found > 0

def proportion_faults_in_simulations(num_checks):
    """Runs many simulations, each one checking num_checks syringes.

    Returns the proportion of simulations in which at least one of
    the num_checks syringes was found to be faulty."""
```

```
    num_simulations_finding_a_fault = 0
    for i in np.arange(num_simulations):
        if found_fault_in_simulation(num_checks):
            num_simulations_finding_a_fault = num_simulations_finding_a_fault +␣
 ↪1
    return num_simulations_finding_a_fault / num_simulations

chance_to_find_syringe = proportion_faults_in_simulations(20)
chance_to_find_syringe
```

[7]: 0.1778

[10]:
```
_ = ok.grade('q2_1')
```

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

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

Question 2. Continue to assume that there really is a 1% failure rate. Find the smallest number of syringes you can check so that you have at least a 50% chance of finding a faulty syringe. (Your answer should be an integer.) Name that number `num_required_for_50_percent`. **It's okay if your answer is off by as many as 11 for full credit.**

[5]:
```
# We are solving this equation for n:
# .5 = 1 - (1 - 1/100)**n
# The solution to that is:
# n = log(1 - .5) / log(1 - 1/100)
# We have to find an integer number of tries, so we round up using math.ceil.
import math
num_required_for_50_percent = math.ceil(math.log(1 - .5) / math.log(1 - 1/100))␣
 ↪# SOLUTION
num_required_for_50_percent
```

[5]: 69

[8]:
```
# For completeness, here is another solution that uses a simulation. # SOLUTION
# We compute the chance of finding a faulty syringe for a range
# of numbers of checks. We reuse the function we wrote above.
proportions = Table().with_column("Num checks", np.arange(20, 100, 5))
proportions = proportions.with_column(
    "Chance to find fault",
    proportions.apply(proportion_faults_in_simulations, "Num checks"))
```

```
# Then we find the smallest number of checks that gives us at least
# a 50% chance of success. Note that the result of this simulation
# is _random_, since the simulation itself is random. If you run it
# several times, you might occasionally see 65 or 75.
num_required_for_50_percent = min(proportions.where("Chance to find fault", are.
 ↪above_or_equal_to(.5)).column("Num checks"))
num_required_for_50_percent
```

[8]:
```
70
```

[9]:
```
_ = ok.grade('q2_2')
```

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

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

Question 3. A doctor purchased 5 syringes and found 4 of them to be faulty. Assuming that there is indeed a 1% failure rate, what was the probability of **exactly 4** out of 5 syringes being faulty?

[10]:
```
probability_of_four_faulty = 5 * (0.01 ** 4) * 0.99 # SOLUTION
probability_of_four_faulty
```

[10]:
```
4.95e-08
```

[11]:
```
_ = ok.grade('q2_3')
```

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

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

Question 4. Assuming that there is indeed a 1% failure rate, assign `order` to a **list** of the numbers 1 through 7, ordered by the size of the quantities described below from smallest to largest. For example, `order` will start with 2 because list item 2 ("Zero") is the smallest quantity.

1. One half
2. Zero
3. The chance that **zero** out of 5 syringes are faulty.

4. The chance that **at least 1** out of 5 syringes is faulty.
5. The chance that **exactly 4** out of 5 syringes are faulty.
6. The chance that **at least 4** out of 5 syringes are faulty.
7. The chance that **all 5** out of 5 syringes are faulty.

[12]:
```
order = [2, 7, 5, 6, 4, 1, 3] # SOLUTION
```

[13]:
```
_ = ok.grade('q2_4')
```

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

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

1.3 3. Predicting Temperatures

In this exercise, we will try to predict the weather in California using the prediction method discussed in section 7.1 of the textbook. Much of the code is provided for you; you will be asked to understand and run the code and interpret the results.

The US National Oceanic and Atmospheric Administration (NOAA) operates thousands of climate observation stations (mostly in the US) that collect information about local climate. Among other things, each station records the highest and lowest observed temperature each day. These data, called "Quality Controlled Local Climatological Data," are publicly available here and described here.

`temperatures.csv` contains an excerpt of that dataset. Each row represents a temperature reading in Fahrenheit from one station on one day. (The temperature is actually the highest temperature observed at that station on that day.) All the readings are from 2015 and from California stations.

[14]:
```
temperatures = Table.read_table("temperatures.csv")
temperatures
```

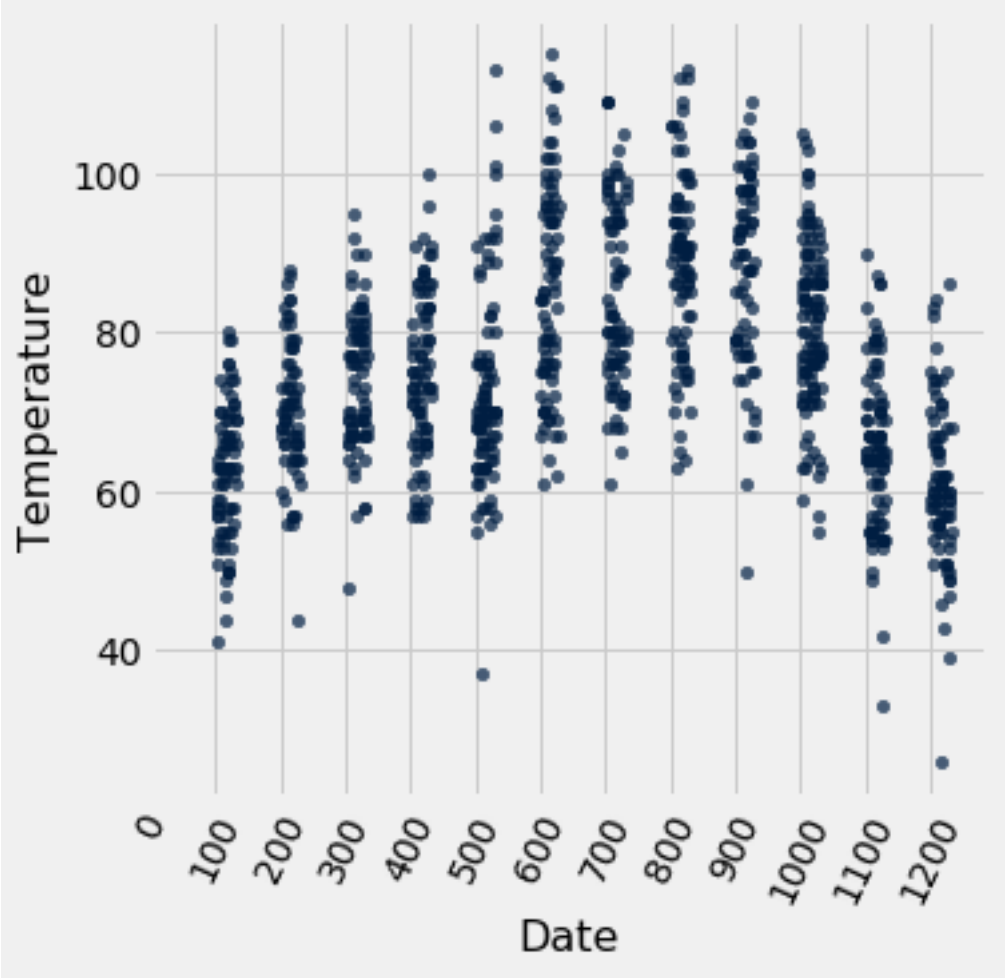
[14]:

Temperature	Date	Latitude	Longitude	Station name
67	1013	40.9781	-124.109	Arcata/Eureka
63	811	38.3208	-123.075	Bodega
94	706	39.1019	-121.568	Marysville
59	1211	36.9358	-121.789	Watsonville
111	620	32.8342	-115.579	Imperial
88	821	33.9	-117.25	Riverside
68	606	33.938	-118.389	Los Angeles
66	205	37.2847	-120.513	Merced
89	902	39.49	-121.618	Oroville

```
105          | 728   | 34.8536   | -116.786   | Daggett
… (990 rows omitted)
```

Here is a scatter plot:

[15]:
```
temperatures.scatter("Date", "Temperature")
_ = plots.xticks(np.arange(0, max(temperatures.column('Date')), 100),␣
 ↪rotation=65)
```



Each entry in the column "Date" is a number in MMDD format, meaning that the last two digits denote the day of the month, and the first 1 or 2 digits denote the month.

Question 1. Why do the data form vertical bands with gaps?

SOLUTION: When we make the scatter plot, the values on the horizontal axis are the numbers in the "Date" column. Days in each month only go up to 30 or 31, so there is a gap between, for example, date 331 (March 31) and date 401 (April 1). Python doesn't know that the numbers in the "Date" column have this special meaning, so it doesn't take out the gaps, and as a result there is roughly a 70-day gap in between each strip.

Let us solve that problem. We will convert each date to the number of days since the start of the year.

Question 2. Implement the `get_day_in_month` function. The result should be an integer. *Hint:* Use the remainder operator.

[16]:
```
def get_month(date):
    """The month in the year for a given date.

    >>> get_month(315)
    3
    """
    return int(date / 100) # Divide by 100 and round down to the nearest integer

def get_day_in_month(date):
    """The day in the month for a given date.

    >>> get_day_in_month(315)
    15
    """
    return date % 100 #SOLUTION
```

[17]:
```
_ = ok.grade('q3_2')
```

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

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

Next, we'll compute the *day of the year* for each temperature reading, which is the number of days from January 1 until the date of the reading.

[18]:
```
# You don't need to change this cell, but you are strongly encouraged
# to read all of the code and understand it.

days_in_month = make_array(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

# A table with one row for each month.  For each month, we have
# the number of the month (e.g. 3 for March), the number of
# days in that month in 2015 (e.g. 31 for March), and the
# number of days in the year before the first day of that month
# (e.g. 0 for January or 59 for March).
days_into_year =  Table().with_columns(
    "Month", np.arange(12)+1,
```

```
    "Days until start of month", np.cumsum(days_in_month) - days_in_month)

# First, compute the month and day-of-month for each temperature.
months = temperatures.apply(get_month, "Date")
day_of_month = temperatures.apply(get_day_in_month, "Date")
with_month_and_day = temperatures.with_columns(
    "Month", months,
    "Day of month", day_of_month
)

# Then, compute how many days have passed since
# the start of the year to reach each date.
t = with_month_and_day.join('Month', days_into_year)
day_of_year = t.column('Days until start of month') + t.column('Day of month')
with_dates_fixed = t.drop(0, 6, 7).with_column("Day of year", day_of_year)
with_dates_fixed
```

[18]:
```
Temperature | Date | Latitude | Longitude | Station name    | Day of year
71          | 127  | 32.7     | -117.2    | San Diego       | 27
61          | 102  | 34.1167  | -119.117  | Point Mugu      | 2
56          | 126  | 40.9781  | -124.109  | Arcata/Eureka   | 26
55          | 111  | 37.3591  | -121.924  | San Jose        | 11
67          | 127  | 36.3189  | -119.629  | Hanford         | 27
69          | 130  | 33.6267  | -116.159  | Palm Springs    | 30
67          | 117  | 32.7     | -117.2    | San Diego       | 17
79          | 124  | 33.8222  | -116.504  | Palm Springs    | 24
73          | 116  | 35.2372  | -120.641  | San Luis Obispo | 16
70          | 128  | 39.1019  | -121.568  | Marysville      | 28
… (990 rows omitted)
```

Question 3. Set `missing` to an array of all the days of the year (integers from 1 through 365) that do not have any temperature readings. *Hint:* One strategy is to start with a table of all days in the year, then use either the predicate `are.not_contained_in` (docs) or the method `exclude` (docs) to eliminate all of the days of the year that do have a temperature reading.

[19]:
```
missing = Table().with_column('Day', np.arange(365)+1).where('Day', are.
 ↪not_contained_in(with_dates_fixed.column('Day of year'))).column(0) #␣
 ↪SOLUTION
missing
```

[19]:
```
array([ 14,  33,  35,  57,  60,  76,  80,  81,  85,  96, 102, 103, 130,
       143, 178, 181, 186, 210, 215, 227, 247, 258, 264, 270, 272, 294,
       319, 344, 354, 358])
```

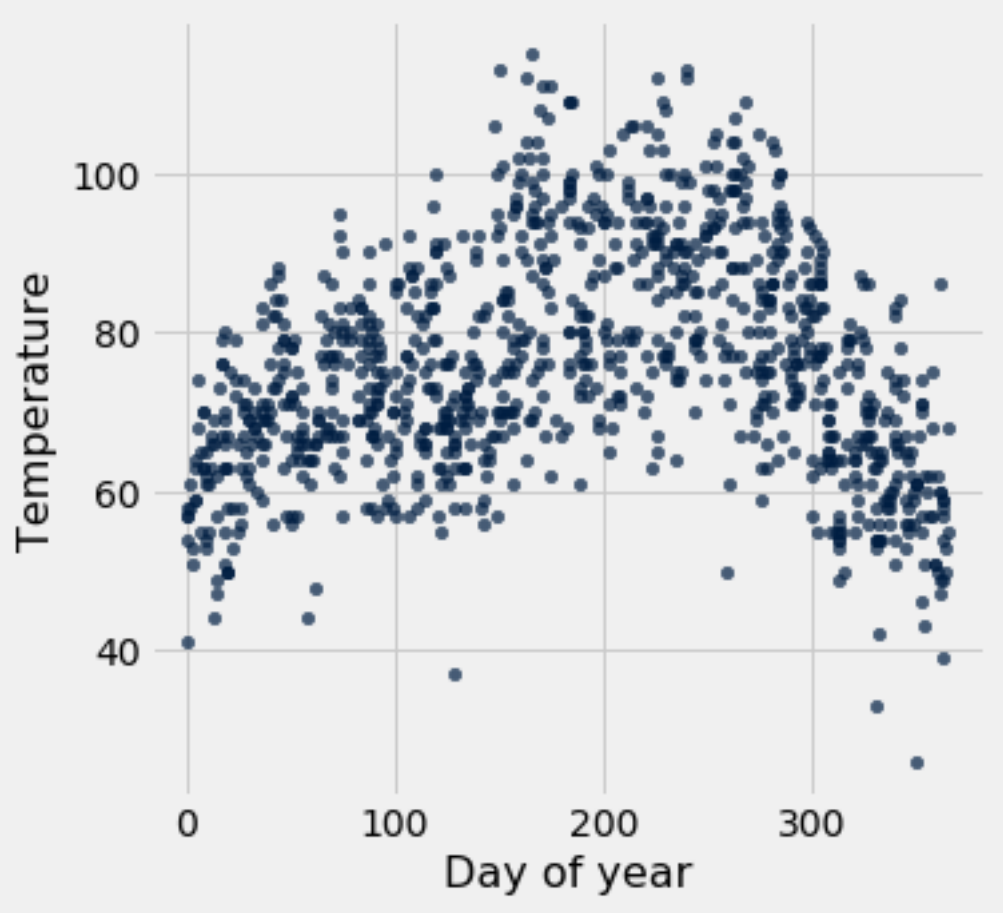
[20]:
```
_ = ok.grade('q3_3')
```

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

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

Using `with_dates_fixed`, we can make a better scatter plot.

[21]:
```
with_dates_fixed.scatter("Day of year", "Temperature")
```



Let's do some prediction. For any reading on any day, we will predict its value using all the readings from the week before and after that day. A reasonable prediction is that the reading will be the average of all those readings. We will package our code in a function.

[22]:
```
def predict_temperature(day):
    """A prediction of the temperature (in Fahrenheit) on a given day at some␣
↪station.
    """
```

```
    nearby_readings = with_dates_fixed.where("Day of year", are.
↪between_or_equal_to(day - 7, day + 7))
    return np.average(nearby_readings.column("Temperature"))
```

Question 4. Suppose you're planning a trip to Yosemite for Thanksgiving break this year, and you'd like to predict the temperature on November 26. Use `predict_temperature` to compute a prediction for a temperature reading on that day.

[23]:
```
thanksgiving_prediction = predict_temperature(days_into_year.where('Month', 11).
↪column(1).item(0) + 26) #SOLUTION
thanksgiving_prediction
```

[23]: 64.31914893617021

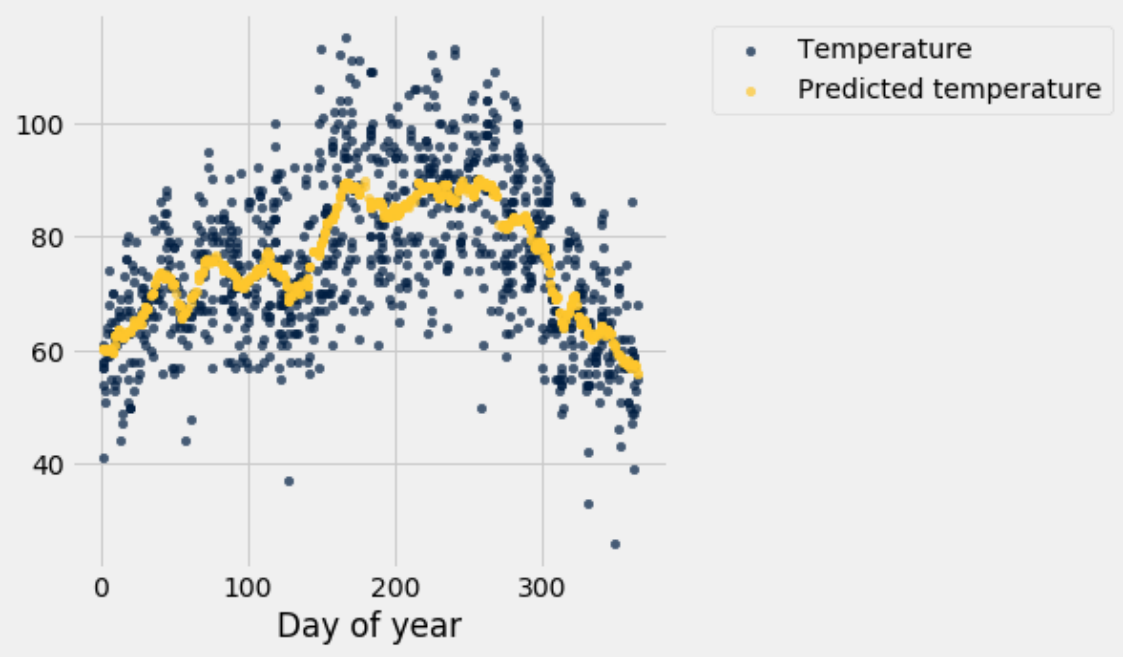
[24]:
```
_ = ok.grade('q3_4')
```

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

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

Below we have computed a predicted temperature for each reading in the table and plotted both. (It may take a **minute or two** to run the cell.)

[25]:
```
with_predictions = with_dates_fixed.with_column(
    "Predicted temperature",
    with_dates_fixed.apply(predict_temperature, "Day of year"))
with_predictions.select("Day of year", "Temperature", "Predicted temperature")\
                .scatter("Day of year")
```



Question 5. The scatter plot is called a *graph of averages*. In the example in the textbook, the graph of averages roughly followed a straight line. Is that true for this one? Using your knowledge about seasons, explain why or why not.

SOLUTION: No, it doesn't follow a straight line. Temperatures go up in the summer and down in the winter, so we would expect them to have a shape like this.

Question 6. According to the Wikipedia article on California's climate, "[t]he climate of California varies widely, from hot desert to subarctic." Suppose we limited our data to weather stations in a smaller area whose climate varied less from place to place (for example, the state of Vermont, or the San Francisco Bay Area).

If we made the same graph for that dataset, in what ways would you expect it to look different? Be specific.

SOLUTION: For any day, the temperature readings would probably be close to each other, since they would all come from the same area. So probably the blue dots (the actual readings) would be lessscattered vertically. That means they would also be scattered more tightly around the yellow dots (the predictions).

[ ]: