lab02

September 5, 2024

[2]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab02.ipynb")
```

1 Lab 2: Table operations

Welcome to Lab 2! This week, we'll learn how to import a module and practice table operations!

Recommended Reading: * Introduction to tables

First, set up the tests and imports by running the cell below.

[71]:
```
# Just run this cell

import numpy as np
from datascience import *
```

2 Lab Warm Up!

We will work together as a class in the following coding cells to prepare you for all sections of this lab.

Make sure to come to lab on time so you don't miss points for this warm-up!

[5]:
```
# Review Python Building Blocks (prep for section 1 and 2):
# What does the following line of python code do?
my_result = min(abs(4*2-3**2), abs(4-30))
#First Python will evaluate the innermost expression,
#math inside abs, then both abs, then use min
my_result
```

[5]: 1

[6]:
```
# Importing a library

import matplotlib
```

[7]:
```python
# Run the following cell to visualize a table
myTable = Table().with_columns("Name", make_array('Jose', 'Lucia', 'Nathan',␣
↪'Annie', 'Vanessa','Ryan', 'Tom'),
                               "Number of pets", make_array(2,0,1,1,3,2,0))
myTable
```

[7]:
```
Name     | Number of pets
Jose     | 2
Lucia    | 0
Nathan   | 1
Annie    | 1
Vanessa  | 3
Ryan     | 2
Tom      | 0
```

[21]:
```python
# Let's practice a couple of table functions (prep for section 3 & 4)
# If needed refer to table in section 5 (and predicates table, end of section 3␣
↪)

#how to load a csv file into a table
#sample = Table.read_table('sample.csv')

#how do we find the number of colums in a table?
myTable.num_columns

#how about the number of rows
myTable.num_rows

#what if I want to turn myTable into a one column table that just has
#the names of students?
myTable.select('Name')
myTable.drop("Number of pets")

#waht if I want to sort the table so that the name of the student
#with tht emost pets is at the top
myTable.sort("Number of pets", descending = True)
```

[21]:
```
Name     | Number of pets
Vanessa  | 3
Jose     | 2
Ryan     | 2
Nathan   | 1
Annie    | 1
Lucia    | 0
Tom      | 0
```

[8]:
```
# Can you use python to answer the question: What is the proportion of␣
  ↪individuals who have pets? (prep for section 4)
# we need a numerator (count te nummer of pets that aren't 0)
#we need a denominator (number of students)
denom = myTable.num_rows
numer = myTable.where('Number of pets', are.above(0)).num_rows
numer/denom
```

[8]:
```
0.7142857142857143
```

3 1. Review: The building blocks of Python code

The two building blocks of Python code are *expressions* and *statements*. An **expression** is a piece of code that

- is self-contained, meaning it would make sense to write it on a line by itself, and
- usually evaluates to a value.

Here are two expressions that both evaluate to 3:

```
3
5 - 2
```

One important type of expression is the **call expression**. A call expression begins with the name of a function and is followed by the argument(s) of that function in parentheses. The function returns some value, based on its arguments. Some important mathematical functions are listed below.

Function	Description
`abs`	Returns the absolute value of its argument
`max`	Returns the maximum of all its arguments
`min`	Returns the minimum of all its arguments
`pow`	Raises its first argument to the power of its second argument
`round`	Rounds its argument to the nearest integer

Here are two call expressions that both evaluate to 3:

```
abs(2 - 5)
max(round(2.8), min(pow(2, 10), -1 * pow(2, 10)))
```

The expression `2 - 5` and the two call expressions given above are examples of **compound expressions**, meaning that they are actually combinations of several smaller expressions. `2 - 5` combines the expressions `2` and `5` by subtraction. In this case, `2` and `5` are called **subexpressions** because they're expressions that are part of a larger expression.

A **statement** is a whole line of code. Some statements are just expressions. The expressions listed above are examples.

Other statements *make something happen* rather than *having a value*. For example, an **assignment statement** assigns a value to a name.

A good way to think about this is that we're **evaluating the right-hand side** of the equals sign and **assigning it to the left-hand side**. Here are some assignment statements:

```
height = 1.3
the_number_five = abs(-5)
absolute_height_difference = abs(height - 1.688)
```

An important idea in programming is that large, interesting things can be built by combining many simple, uninteresting things. The key to understanding a complicated piece of code is breaking it down into its simple components.

For example, a lot is going on in the last statement above, but it's really just a combination of a few things. This picture describes what's going on.

Question 1.1. In the next cell, assign the name `new_year` to the larger number among the following two numbers:

1. the **absolute value** of $2^5 - 2^{11} - 2^1 + 1$, and
2. $5 \times 13 \times 31 + 5$.

Try to use just one statement (one line of code). Be sure to check your work by executing the test cell afterward.

[11]:
```
new_year = max(abs(2**5-2**11-2**1+1), abs(5*13*31+5))
new_year
```

[11]: 2020

[12]:
```
grader.check("q11")
```

[12]: q11 results: All test cases passed!

We've asked you to use one line of code in the question above because it only involves mathematical operations. However, more complicated programming questions will more require more steps. It isn't always a good idea to jam these steps into a single line because it can make the code harder to read and harder to debug.

Good programming practice involves splitting up your code into smaller steps and using appropriate names. You'll have plenty of practice in the rest of this course!

4 2. Importing code

source

Most programming involves work that is very similar to work that has been done before. Since writing code is time-consuming, it's good to rely on others' published code when you can. Rather than copy-pasting, Python allows us to **import modules**. A module is a file with Python code that has defined variables and functions. By importing a module, we are able to use its code in our own notebook.

Python includes many useful modules that are just an `import` away. We'll look at the `math` module as a first example. The `math` module is extremely useful in computing mathematical expressions in Python.

Suppose we want to very accurately compute the area of a circle with a radius of 5 meters. For that, we need the constant π, which is roughly 3.14. Conveniently, the `math` module has `pi` defined for us:

[13]:
```
import math
radius = 5  # = 5 centimeters
area_of_circle = radius**2 * math.pi # in centimeters squared
area_of_circle
```

[13]: 78.53981633974483

In the code above, the line `import math` imports the math module. This statement creates a module and then assigns the name `math` to that module. We are now able to access any variables or functions defined within `math` by typing the name of the module followed by a dot, then followed by the name of the variable or function we want.

`<module name>.<name>`

Question 2.1. Now your turn to try! Compute the circumference of the same circle as above (radius equal to 5 cm), giving it the name `circum_of_circle`. (circumference = 2 π radius)

Remember: You can access `pi` from the `math` module like the area example above!

[14]:
```
import math
radius = 5
circum_of_circle = 2*math.pi*radius
circum_of_circle
```

[14]: 31.41592653589793

[15]:
```
grader.check("q21")
```

[15]: q21 results: All test cases passed!

4.1 2.2. Accessing functions

In the question above, you accessed variables within the `math` module.

Modules also define **functions**. For example, `math` provides the name `floor` for the floor function. Having imported `math` already, we can write `math.floor(7.5)` to compute the floor of 7.5. (Note that the floor function returns the largest integer less than or equal to a given number.)

Question 2.2. Compute the floor of pi using `floor` and `pi` from the `math` module. Give the result the name `floor_of_pi`.

[19]:
```
import math
floor_of_pi = math.floor(math.pi)
floor_of_pi
```

[19]: 3

[20]:
```
grader.check("q22")
```

[20]: q22 results: All test cases passed!

For your reference, below are some more examples of functions from the `math` module.

Notice how different functions take in different numbers of arguments. Often, the documentation of the module will provide information on how many arguments are required for each function.

Hint: If you press **`shift+tab`** *while next to the function call, the documentation for that function will appear*

[21]:
```
# Calculating logarithms (the logarithm of 8 in base 2).
# The result is 3 because 2 to the power of 3 is 8.
math.log(8, 2)
```

[21]: 3.0

[22]:
```
# Calculating square roots.
math.sqrt(5)
```

[22]: 2.23606797749979

There are various ways to import and access code from outside sources. The method we used above — `import <module_name>` — imports the entire module and requires that we use `<module_name>.<name>` to access its code.

We can also import a specific constant or function instead of the entire module. Notice that you don't have to use the module name beforehand to reference that particular value. However, you do have to be careful about reassigning the names of the constants or functions to other values!

[23]:
```
# Importing just cos and pi from math.
# We don't have to use `math.` in front of sqrt or pi
from math import sqrt, pi
print(sqrt(pi))

# We do have to use it in front of other functions from math though, like this:
math.log(100,10) #the result is 2 because 10 to the power of 2 is 100
```

```
1.7724538509055159
```

[23]: 2.0

Don't worry too much about which type of import to use. It's often a coding style choice left up to each programmer. In this course, you'll always import the necessary modules when you run the setup cell (like the first code cell in this lab).

Let's move on to practicing some of the table operations you've learned in lecture!

5 3. Table operations

The table `farmers_markets.csv` contains data on farmers' markets in the United States (data collected by the USDA). Each row represents one such market.

Run the next cell to load the `farmers_markets` table.

[29]:
```
# Just run this cell

farmers_markets = Table.read_table('farmers_markets.csv')
```

Let's examine our table to see what data it contains.

Question 3.1. Use the method `show` to display the first 5 rows of `farmers_markets`.

Note: The terms "method" and "function" are technically not the same thing, but for the purposes of this course, we will use them interchangeably.

[31]:
```
farmers_markets.show(5)
```

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

Notice that some of the values in this table are missing, as denoted by "nan." This means either that the value is not available (e.g. if we don't know the market's street address) or not applicable (e.g. if the market doesn't have a street address). You'll also notice that the table has a large number of columns in it!

5.0.1 `num_columns`

The table property `num_columns` returns the number of columns in a table. (A "property" is just like a method that doesn't need to be called by adding parentheses.)

Example call: `<tbl>.num_columns`

Question 3.2. Use `num_columns` to find the number of columns in our farmers' markets dataset.

Assign the number of columns to `num_farmers_markets_columns`.

[35]:
```
num_farmers_markets_columns =farmers_markets.num_columns
print("The table has", num_farmers_markets_columns, "columns in it!")
```

```
The table has 59 columns in it!
```

[36]:
```
grader.check("q32")
```

[36]:
```
q32 results: All test cases passed!
```

5.0.2 `num_rows`

Similarly, the property `num_rows` tells you how many rows are in a table.

[72]:
```
# Just run this cell

num_farmers_markets_rows = farmers_markets.num_rows
print("The table has", num_farmers_markets_rows, "rows in it!")
```

```
The table has 8546 rows in it!
```

5.0.3 select

Most of the columns are about particular products – whether the market sells tofu, pet food, etc. If we're not interested in that information, it just makes the table difficult to read. This comes up more than you might think, because people who collect and publish data may not know ahead of time what people will want to do with it.

In such situations, we can use the table method `select` to choose only the columns that we want in a particular table. It takes any number of arguments. Each should be the name of a column in the table. It returns a new table with only those columns in it. The columns are in the order *in which they were listed as arguments.*

For example, the value of `farmers_markets.select("MarketName", "State")` is a table with only the name and the state of each farmers' market in `farmers_markets`.

Question 3.3. Use `select` to create a table with only the name, city, state, latitude (`y`), and longitude (`x`) of each market. Call that new table `farmers_markets_locations`.

[73]:
```
# HINT: Make sure to be exact when using column names with `select`;␣
↪double-check capitalization!
farmers_markets_locations = farmers_markets.select("MarketName", "city",␣
↪"State", "y", "x")
farmers_markets_locations
```

[73]:
```
MarketName                                         | city       | State
| y       | x
 Caledonia Farmers Market Association - Danville | Danville   | Vermont
| 44.411  | -72.1403
 Stearns Homestead Farmers' Market               | Parma      | Ohio
| 41.3751 | -81.7286
100 Mile Market                                    | Kalamazoo  | Michigan
| 42.296  | -85.5749
106 S. Main Street Farmers Market                  | Six Mile   | South Carolina
| 34.8042 | -82.8187
10th Steet Community Farmers Market                | Lamar      | Missouri
| 37.4956 | -94.2746
112st Madison Avenue                               | New York   | New York
| 40.7939 | -73.9493
12 South Farmers Market                            | Nashville  | Tennessee
| 36.1184 | -86.7907
125th Street Fresh Connect Farmers' Market         | New York   | New York
| 40.809  | -73.9482
```

```
12th & Brandywine Urban Farm Market                  | Wilmington | Delaware
| 39.7421 | -75.5345
14&U Farmers' Market                                 | Washington | District of
Columbia | 38.917  | -77.0321
… (8536 rows omitted)
```

[74]:
```
grader.check("q33")
```

[74]:
```
q33 results: All test cases passed!
```

5.0.4 drop

`drop` serves the same purpose as `select`, but it takes away the columns that you provide rather than the ones that you don't provide. Like `select`, `drop` returns a new table.

Question 3.4. Suppose you just didn't want the `FMID` and `updateTime` columns in `farmers_markets`. Create a table that's a copy of `farmers_markets` but doesn't include those columns. Call that table `farmers_markets_without_fmid`.

[42]:
```
farmers_markets_without_fmid = farmers_markets.drop("FMID", "updateTime")
farmers_markets_without_fmid
```

[42]:
```
MarketName                                           | street
| city        | County                 | State                  | zip   | x          |
y        | Website                                                    | Facebook
| Twitter                                  | Youtube | OtherMedia
| Organic | Tofu | Bakedgoods | Cheese | Crafts | Flowers | Eggs | Seafood |
Herbs | Vegetables | Honey | Jams | Maple | Meat | Nursery | Nuts | Plants |
Poultry | Prepared | Soap | Trees | Wine | Coffee | Beans | Fruits | Grains |
Juices | Mushrooms | PetFood | WildHarvested | Location
| Credit | WIC   | WICcash | SFMNP | SNAP | Season1Date                  |
Season1Time                                        | Season2Date | Season2Time |
Season3Date | Season3Time | Season4Date | Season4Time
 Caledonia Farmers Market Association - Danville | nan
| Danville    | Caledonia              | Vermont                | 05828 | -72.1403 |
44.411   | https://sites.google.com/site/caledoniafarmersmarket/ |
https://www.facebook.com/Danville.VT.Farmers.Market/         | nan
| nan      | nan                                                          | Y
| N     | Y            | Y       | Y       | Y         | Y     | N         | Y      | Y
| Y      | Y     | Y      | Y     | N         | N     | Y       | Y         | Y          | Y
| Y      | N     | Y       | Y      | Y        | N        | Y        | N            | Y
| N               | nan                                                          | Y
| Y     | N         | Y      | N     | 06/08/2016 to 10/12/2016 | Wed: 9:00 AM-1:00
PM;                          | nan           | nan           | nan           | nan
| nan           | nan
 Stearns Homestead Farmers' Market                  | 6975 Ridge Road
| Parma       | Cuyahoga               | Ohio                   | 44130 | -81.7286 |
41.3751 | http://Stearnshomestead.com                                | nan
```

| nan | nan | nan
| - | N | Y | N | N | Y | Y | N | Y
| Y | Y | Y | Y | Y | N | N | Y | N |
N | N | N | N | N | N | Y | N | N | N
| Y | N | nan
| Y | Y | N | Y | Y | 06/25/2016 to 10/01/2016 | Sat: 9:00
AM-1:00 PM; | nan | nan | nan |
nan | nan | nan
100 Mile Market | 507 Harrison St
| Kalamazoo | Kalamazoo | Michigan | 49007 | -85.5749 |
42.296 | http://www.pfcmarkets.com |
https://www.facebook.com/100MileMarket/?fref=ts | nan
| nan | https://www.instagram.com/100milemarket/ | N
| N | Y | Y | N | Y | Y | N | Y | Y
| Y | Y | Y | Y | N | N | N | Y | Y | Y
| N | Y | N | N | Y | Y | N | N | N
| N | nan | Y
| Y | N | Y | Y | 05/04/2016 to 10/12/2016 | Wed: 3:00 PM-7:00
PM; | nan | nan | nan | nan
| nan | nan
106 S. Main Street Farmers Market | 106 S. Main Street
| Six Mile | nan | South Carolina | 29682 | -82.8187 |
34.8042 | http://thetownofsixmile.wordpress.com/ | nan
| nan | nan | nan
| - | N | N | N | N | N | N | N | N
| N | N | N | N | N | N | N | N | N |
N | N | N | N | N | N | N | N | N | N
| N | N | nan
| Y | N | N | N | N | nan | nan
| nan | nan | nan | nan | nan | nan
10th Steet Community Farmers Market | 10th Street and Poplar
| Lamar | Barton | Missouri | 64759 | -94.2746 |
37.4956 | nan | nan
| nan | nan | http://agrimissouri.com/mo-
grown/grodetail.php?type=mo-g … | - | N | Y | N | Y
| N | Y | N | Y | Y | Y | Y | N | Y |
N | N | Y | Y | Y | Y | N | N | N | N
| Y | N | N | N | N | N | nan
| Y | N | N | N | N | 04/02/2014 to 11/30/2014 | Wed: 3:00
PM-6:00 PM;Sat: 8:00 AM-1:00 PM; | nan | nan | nan |
nan | nan | nan
112st Madison Avenue | 112th Madison Avenue
| New York | New York | New York | 10029 | -73.9493 |
40.7939 | nan | nan
| nan | nan | nan
| - | N | Y | N | Y | Y | N | N | Y
| Y | Y | Y | N | N | N | Y | N | N |

Y | Y | N | N | N | N | N | N | N | N
| N | N | Private business parking lot
| N | N | Y | Y | N | July to November | Tue:8:00
am - 5:00 pm;Sat:8:00 am - 8:00 pm; | nan | nan | nan |
nan | nan | nan
12 South Farmers Market | 3000 Granny White Pike
| Nashville | Davidson | Tennessee | 37204 | -86.7907 |
36.1184 | http://www.12southfarmersmarket.com |
12_South_Farmers_Market | @12southfrmsmkt
| nan | @12southfrmsmkt | Y
| N | Y | Y | N | Y | Y | N | Y | Y
| Y | Y | Y | Y | N | N | N | Y | Y | Y
| N | N | Y | N | Y | N | Y | Y | Y
| N | nan | Y
| N | N | N | Y | 05/05/2015 to 10/27/2015 | Tue: 3:30 PM-6:30
PM; | nan | nan | nan | nan
| nan | nan
125th Street Fresh Connect Farmers' Market | 163 West 125th Street and
Adam Clayton Powell, Jr. Blvd. | New York | New York | New York
| 10027 | -73.9482 | 40.809 | http://www.125thStreetFarmersMarket.com
| https://www.facebook.com/125thStreetFarmersMarket |
https://twitter.com/FarmMarket125th | nan | Instagram-->
125thStreetFarmersMarket | Y | N | Y |
Y | Y | Y | Y | N | Y | Y | Y | Y |
Y | Y | N | Y | N | Y | Y | Y | N | Y
| Y | N | Y | N | Y | N | N | N
| Federal/State government building grounds | Y | Y | N
| Y | Y | 06/10/2014 to 11/25/2014 | Tue: 10:00 AM-7:00 PM;
| nan | nan | nan | nan | nan | nan
12th & Brandywine Urban Farm Market | 12th & Brandywine Streets
| Wilmington | New Castle | Delaware | 19801 | -75.5345 |
39.7421 | nan |
https://www.facebook.com/pages/12th-Brandywine-Urban-Far … | nan
| nan | https://www.facebook.com/delawareurbanfarmcoalition | N
| N | N | N | N | N | N | N | Y | Y
| N | N | N | N | N | N | N | N | N | N
| N | N | N | N | Y | N | N | N | N
| N | On a farm from: a barn, a greenhouse, a tent, a stand, etc | N
| N | N | N | Y | 05/16/2014 to 10/17/2014 | Fri: 8:00 AM-11:00
AM; | nan | nan | nan | nan
| nan | nan
14&U Farmers' Market | 1400 U Street NW
| Washington | District of Columbia | District of Columbia | 20009 | -77.0321 |
38.917 | nan |
https://www.facebook.com/14UFarmersMarket |
https://twitter.com/14UFarmersMkt | nan | nan
| Y | N | Y | Y | N | Y | Y | N | Y

```
| Y          | Y      | Y     | N      | Y     | N         | Y     | Y        | Y          |
N        | N      | N       | N      | N        | Y       | Y        | Y        | Y        | N
| N       | N               | Other
| Y       | Y      | Y         | Y       | Y     | 05/03/2014 to 11/22/2014 | Sat: 9:00
AM-1:00 PM;                       | nan           | nan           | nan           |
nan           | nan           | nan
… (8536 rows omitted)
```

[43]:
```
grader.check("q34")
```

[43]:
```
q34 results: All test cases passed!
```

Now, suppose we want to answer some questions about farmers' markets in the US. For example, which market(s) have the largest longitude (given by the `x` column)?

To answer this, we'll sort `farmers_markets_locations` by longitude.

[44]:
```
farmers_markets_locations.sort('x')
```

[44]:
```
MarketName                                          | city          | State   | y
| x
Trapper Creek Farmers Market                        | Trapper Creek | Alaska |
53.8748 | -166.54
Kekaha Neighborhood Center (Sunshine Markets)       | Kekaha        | Hawaii |
21.9704 | -159.718
Hanapepe Park (Sunshine Markets)                    | Hanapepe      | Hawaii |
21.9101 | -159.588
Kalaheo Neighborhood Center (Sunshine Markets)      | Kalaheo       | Hawaii |
21.9251 | -159.527
Hawaiian Farmers of Hanalei                         | Hanalei       | Hawaii |
22.2033 | -159.514
Hanalei Saturday Farmers Market                     | Hanalei       | Hawaii |
22.2042 | -159.492
Kauai Culinary Market                               | Koloa         | Hawaii |
21.9067 | -159.469
Koloa Ball Park (Knudsen) (Sunshine Markets)        | Koloa         | Hawaii |
21.9081 | -159.465
West Kauai Agricultural Association                 | Poipu         | Hawaii |
21.8815 | -159.435
Kilauea Neighborhood Center (Sunshine Markets)      | Kilauea       | Hawaii |
22.2112 | -159.406
… (8536 rows omitted)
```

Oops, that didn't answer our question because we sorted from smallest to largest longitude. To look at the largest longitudes, we'll have to sort in reverse order.

[45]:
```
farmers_markets_locations.sort('x', descending=True)
```

[45]:
```
MarketName                                   | city               | State
| y        | x
Christian "Shan" Hendricks Vegetable Market | Saint Croix        | Virgin
Islands | 17.7449 | -64.7043
La Reine Farmers Market                      | Saint Croix        | Virgin
Islands | 17.7322 | -64.7789
Anne Heyliger Vegetable Market               | Saint Croix        | Virgin
Islands | 17.7099 | -64.8799
Rothschild Francis Vegetable Market          | St. Thomas         | Virgin
Islands | 18.3428 | -64.9326
Feria Agrícola de Luquillo                   | Luquillo           | Puerto Rico
| 18.3782 | -65.7207
El Mercado Familiar                          | San Lorenzo        | Puerto Rico
| 18.1871 | -65.9674
El Mercado Familiar                          | Gurabo             | Puerto Rico
| 18.2526 | -65.9786
El Mercado Familiar                          | Patillas           | Puerto Rico
| 18.0069 | -66.0135
El Mercado Familiar                          | Caguas zona urbana | Puerto Rico
| 18.2324 | -66.039
El Maercado Familiar                         | Arroyo zona urbana | Puerto Rico
| 17.9686 | -66.0617
… (8536 rows omitted)
```

(The `descending=True` bit is called an *optional argument*. It has a default value of `False`, so when you explicitly tell the function `descending=True`, then the function will sort in descending order.)

5.0.5 `sort`

Some details about sort:

1. The first argument to `sort` is the name of a column to sort by.
2. If the column has text in it, `sort` will sort alphabetically; if the column has numbers, it will sort numerically.
3. The value of `farmers_markets_locations.sort("x")` is a *copy* of `farmers_markets_locations`; the `farmers_markets_locations` table doesn't get modified. For example, if we called `farmers_markets_locations.sort("x")`, then running `farmers_markets_locations` by itself would still return the unsorted table.
4. Rows always stick together when a table is sorted. It wouldn't make sense to sort just one column and leave the other columns alone. For example, in this case, if we sorted just the `x` column, the farmers' markets would all end up with the wrong longitudes.

Question 3.5. Create a version of `farmers_markets_locations` that's sorted by **latitude (`y`)**, with the largest latitudes first. Call it `farmers_markets_locations_by_latitude`.

[75]:
```
farmers_markets_locations_by_latitude = farmers_markets_locations.sort('y',␣
↪descending=True)
farmers_markets_locations_by_latitude
```

[75]:
```
MarketName                        | city           | State   | y       | x
Tanana Valley Farmers Market      | Fairbanks      | Alaska | 64.8628 | -147.781
Ester Community Market            | Ester          | Alaska | 64.8459 | -148.01
Fairbanks Downtown Market         | Fairbanks      | Alaska | 64.8444 | -147.72
Nenana Open Air Market            | Nenana         | Alaska | 64.5566 | -149.096
Highway's End Farmers' Market     | Delta Junction | Alaska | 64.0385 | -145.733
MountainTraders                   | Talkeetna      | Alaska | 62.3231 | -150.118
Talkeetna Farmers Market          | Talkeetna      | Alaska | 62.3228 | -150.118
Denali Farmers Market             | Anchorage      | Alaska | 62.3163 | -150.234
Kenny Lake Harvest II             | Valdez         | Alaska | 62.1079 | -145.476
Copper Valley Community Market    | Copper Valley  | Alaska | 62.0879 | -145.444
… (8536 rows omitted)
```

[76]:
```
grader.check("q35")
```

[76]:
```
q35 results: All test cases passed!
```

Now let's say we want a table of all farmers' markets in California. Sorting won't help us much here because California is closer to the middle of the dataset.

Instead, we use the table method `where`.

[77]:
```
california_farmers_markets = farmers_markets_locations.where('State', are.
  ↪equal_to('California'))
california_farmers_markets
```

[77]:
```
MarketName                                 | city          | State      | y       |
x
Adelanto Stadium Farmers Market            | Victorville   | California | 34.5593 |
-117.405
Alameda Farmers' Market                    | Alameda       | California | 37.7742 |
-122.277
Alisal Certified Farmers' Market           | Salinas       | California | 36.6733 |
-121.634
Altadena Farmers' Market                   | Altadena      | California | 34.2004 |
-118.158
Alum Rock Village Farmers' Market          | San Jose      | California | 37.3678 |
-121.833
Amador Farmers' Market-- Jackson           | Jackson       | California | 38.3488 |
-120.774
Amador Farmers' Market-- Pine Grove        | Pine Grove    | California | 38.3488 |
-120.774
Amador Farmers' Market-- Sutter Creek      | Sutter Creek  | California | 38.3488 |
-120.774
Anderson Happy Valley Farmers Market       | Anderson      | California | 40.4487 |
-122.408
Angels Camp Farmers Market-Fresh Fridays | Angels Camp   | California | 38.0722 |
-120.543
```

```
… (745 rows omitted)
```

Ignore the syntax for the moment. Instead, try to read that line like this:

> Assign the name **`california_farmers_markets`** to a table whose rows are the rows in the **`farmers_markets_locations`** table **`where`** the **`'State'`**s **`are equal to California`**.

5.0.6 `where`

Now let's dive into the details a bit more. `where` takes 2 arguments:

1. The name of a column. `where` finds rows where that column's values meet some criterion.
2. A predicate that describes the criterion that the column needs to meet.

The predicate in the example above called the function `are.equal_to` with the value we wanted, 'California'. We'll see other predicates soon.

`where` returns a table that's a copy of the original table, but **with only the rows that meet the given predicate**.

Question 3.6. Use `california_farmers_markets` to create a table called `SB_markets` containing farmers' markets in Santa Barbara, California.

[57]:
```
SB_markets = farmers_markets.where('city', are.equal_to('Santa Barbara'))
SB_markets
```

[57]:
```
FMID    | MarketName                                  | street                    |
city          | County          | State        | zip    | x          | y          |
Website                          | Facebook | Twitter | Youtube | OtherMedia |
Organic | Tofu | Bakedgoods | Cheese | Crafts | Flowers | Eggs | Seafood | Herbs
| Vegetables | Honey | Jams | Maple | Meat | Nursery | Nuts | Plants | Poultry |
Prepared | Soap | Trees | Wine | Coffee | Beans | Fruits | Grains | Juices |
Mushrooms | PetFood | WildHarvested | updateTime            | Location
| Credit | WIC   | WICcash | SFMNP | SNAP | Season1Date          | Season1Time
| Season2Date | Season2Time | Season3Date | Season3Time | Season4Date |
Season4Time
1006694 | Harding Elementary School CFM          | 1625 Robbins Street      |
Santa Barbara | Santa Barbara | California | nan    | -119.722 | 34.4166 | nan
| nan       | nan       | nan       | nan          | -        | N     | N            | N
| N       | N         | N     | N         | N      | N            | N       | N    | N
| N    | N         | N     | N        | N          | N          | N     | N      | N    | N
| N      | N        | N        | N        | N            | N         | N              | 2011
| nan                            | N        | N     | N          | N       | N     | nan
| nan                      | nan          | nan           | nan           | nan
| nan           | nan
1002376 | Montrose Harvest Market Farmers Market | Honolulu & Ocean Avenue |
Santa Barbara | Los Angeles   | California | 93101 | -119.713 | 34.4222 |
http://www.farmernet.com         | nan       | nan       | nan       | nan          | -
| N    | N            | N        | N        | N          | N     | N          | N       | N
| N      | N      | N       | N      | N          | N      | N        | N          | N          | N
```

```
| N     | N    | N      | N     | N      | N      | N      | N        | N
| N              | 2009                   | nan                          | N       |
N    | N       | N     | N    | nan                  | nan                       |
nan          | nan          | nan          | nan          | nan          | nan
1006691 | Santa Barbara Farmers Market           | 600 Santa Barbara St.    |
Santa Barbara | Santa Barbara | California | 93103 | -119.695 | 34.4199 |
http://www.sbfarmersmarket.org | nan        | nan      | nan      | nan         | Y
| N    | Y            | Y       | N       | Y        | N     | Y        | Y      | Y
| Y     | Y    | N      | Y    | N         | Y    | Y       | N        | N          | N
| N     | N    | N      | N     | N      | N      | N      | N        | N
| N              | 1/14/2012 4:29:04 PM | Private business parking lot | N       |
Y    | N       | Y     | N    | January to December | Sat:8:30 am - 1:00 pm; |
nan          | nan          | nan          | nan          | nan          | nan
1006692 | Santa Barbara Farmers Market           | 600 State Street         |
Santa Barbara | Santa Barbara | California | 93103 | -119.697 | 34.4177 |
http://www.sbfarmersmarket.org | nan        | nan      | nan      | nan         | Y
| N    | Y            | Y       | N       | Y        | N     | Y        | Y      | Y
| Y     | Y    | N      | Y    | N         | Y    | Y       | N        | N          | N
| N     | N    | N      | N     | N      | N      | N      | N        | N
| N              | 1/14/2012 4:29:04 PM | Closed-off public street     | N       |
Y    | N       | Y     | N    | January to December | Tue:4:00 pm - 7:30 pm; |
nan          | nan          | nan          | nan          | nan          | nan
1000007 | Santa Barbara La Cumbre Farmers Market | 110 South Hope Avenue    |
Santa Barbara | Santa Barbara | California | 93101 | -119.747 | 34.4378 |
http://www.sbfarmersmarket.org | nan        | nan      | nan      | nan         | -
| N    | N            | N       | N       | N        | N     | N        | N      | N
| N     | N    | N      | N    | N         | N    | N       | N        | N          | N
| N     | N    | N      | N     | N      | N      | N      | N        | N
| N              | 2009                   | nan                          | N       |
N    | N       | N     | N    | nan                  | nan                       |
nan          | nan          | nan          | nan          | nan          | nan
```

[58]:
```python
grader.check("q36")
```

[58]:
```
q36 results: All test cases passed!
```

Recognize any of them?

So far we've only been using `where` with the predicate that requires finding the values in a column to be *exactly* equal to a certain value. However, there are many other predicates. Here are a few:

Predicate	Example	Result
`are.equal_to`	`are.equal_to(50)`	Find rows with values equal to 50
`are.not_equal_to`	`are.not_equal_to(50)`	Find rows with values not equal to 50
`are.above`	`are.above(50)`	Find rows with values above (and not equal to) 50

Predicate	Example	Result
`are.above_or_equal_to`	`are.above_or_equal_to(50)`	Find rows with values above 50 or equal to 50
`are.below`	`are.below(50)`	Find rows with values below 50
`are.between`	`are.between(2, 10)`	Find rows with values above or equal to 2 and below 10

5.1 4. Analyzing a dataset

Now that you're familiar with table operations, let's answer an interesting question about a dataset!

Run the cell below to load the `imdb` table. It contains information about the 250 highest-rated movies on IMDb.

[78]:
```
# Just run this cell

imdb = Table.read_table('imdb.csv')
imdb
```

[78]:
```
Votes   | Rating | Title                   | Year | Decade
88355   | 8.4    | M                       | 1931 | 1930
132823  | 8.3    | Singin' in the Rain     | 1952 | 1950
74178   | 8.3    | All About Eve           | 1950 | 1950
635139  | 8.6    | Léon                    | 1994 | 1990
145514  | 8.2    | The Elephant Man        | 1980 | 1980
425461  | 8.3    | Full Metal Jacket       | 1987 | 1980
441174  | 8.1    | Gone Girl               | 2014 | 2010
850601  | 8.3    | Batman Begins           | 2005 | 2000
37664   | 8.2    | Judgment at Nuremberg   | 1961 | 1960
46987   | 8      | Relatos salvajes        | 2014 | 2010
… (240 rows omitted)
```

Often, we want to perform multiple operations - sorting, filtering, or others - in order to turn a table we have into something more useful. You can do these operations one by one, e.g.

```
first_step = original_tbl.where("col1", are.equal_to(12))
second_step = first_step.sort('col2', descending=True)
```

However, since the value of the expression `original_tbl.where("col1", are.equal_to(12))` is itself a table, you can just call a table method on it:

```
original_tbl.where("col1", are.equal_to(12)).sort('col2', descending=True)
```

You should organize your work in the way that makes the most sense to you, using informative names for any intermediate tables you create.

Question 4.1. Create a table of movies released between 2010 and 2016 (inclusive) with ratings above 8. The table should only contain the columns `Title` and `Rating`, **in that order**.

Assign the table to the name `above_eight`.

[64]:
```
# HINT: Think about the steps you need to take, and try to put them in an order
 ↪that make sense.
# Feel free to create helper code lines with intermediate tables for each step,
 ↪but make sure you assign your final table the name `above_eight`!

above_eight = imdb.where("Year", are.between_or_equal_to(2010,2016)).
 ↪where("Rating", are.above(8)).select("Title", "Rating")
above_eight
```

[64]:
```
Title                    | Rating
Gone Girl                | 8.1
Warrior                  | 8.2
Intouchables             | 8.5
Shutter Island           | 8.1
12 Years a Slave         | 8.1
Inside Out (2015/I)      | 8.5
Jagten                   | 8.2
Toy Story 3              | 8.3
How to Train Your Dragon | 8.1
Interstellar             | 8.6
… (10 rows omitted)
```

[65]:
```
grader.check("q41")
```

[65]:
```
q41 results: All test cases passed!
```

Question 4.2. Use `num_rows` (and arithmetic) to find the *proportion* of movies in the dataset that were released 1900-1999, and the *proportion* of movies in the dataset that were released in the year 2000 or later.

Assign `proportion_in_20th_century` to the proportion of movies in the dataset that were released 1900-1999, and `proportion_in_21st_century` to the proportion of movies in the dataset that were released in the year 2000 or later.

[69]:
```
#STEP 1: Get the total number of movies in the dataset
num_movies_in_dataset = imdb.num_rows
#STEP 2: Get the number of movies in the 20th century (that does NOT include
 ↪2000)
num_in_20th_century = imdb.where('Year', are.between(1900,2000)).num_rows
#STEP 3: Get the number of movies in the 21st century (movies released 2000 or
 ↪later)
num_in_21st_century = imdb.where('Year', are.above_or_equal_to(2000)).num_rows

# STEP 4: Calculate the proportions
# HINT: The *proportion* of movies released in the 1900's is the *number* of
 ↪movies released in the 1900's, divided by the *total number* of movies.
proportion_in_20th_century = num_in_20th_century/num_movies_in_dataset
```

```python
proportion_in_21st_century = num_in_21st_century/num_movies_in_dataset

print("Proportion in 20th century:", proportion_in_20th_century)
print("Proportion in 21st century:", proportion_in_21st_century)
```

```
Proportion in 20th century: 0.684
Proportion in 21st century: 0.316
```

[70]:
```python
grader.check("q42")
```

[70]:
```
q42 results: All test cases passed!
```

5.2 5. Summary

For your reference, here's a table of all the functions and methods we saw in this lab. We'll learn more methods to add to this table in the coming week!

Name	Example	Purpose
`sort`	`tbl.sort("N")`	Create a copy of a table sorted by the values in a column
`where`	`tbl.where("N", are.above(2))`	Create a copy of a table with only the rows that match some *predicate*
`num_rows`	`tbl.num_rows`	Compute the number of rows in a table
`num_columns`	`tbl.num_columns`	Compute the number of columns in a table
`select`	`tbl.select("N")`	Create a copy of a table with only some of the columns
`drop`	`tbl.drop("2*N")`	Create a copy of a table without some of the columns

Alright! You're finished with lab 2! Be sure to… - run all the cells and the grader checks and verify that they all pass, - choose **Download as PDF via LaTeX** from the **File** menu - choose **Download as Notebook** from the **File** menu - correctly name the files and submit them on **canvas**.