**Computers in the Cronkite building that have Anaconda:**

**-Room 318, computers 07 and 08**

**-Room 320, computers 10, 11 and 12**

**To run each cell in your jupyter notebook, hit shift return (or shift enter if you’re on a PC)**

**Getting Started**

Before we start working with data, we need to tell the computer what python libraries we are going to use. Think of libraries as toolkits. Some examples (to name just a few!):

* Pandas: filtering, sorting, aggregating data (think pivot tables but in here they’re called ‘group bys’), math, merging datasets, appending datasets and more!
* Geopandas: analyze spatial files that feed into maps. Merging together two datasets based on the location of your records. Creates spatial files.
* Numpy: good for more complex math. We’ll use this for if/then statements
* Beautiful Soup: good for scraping.

You open these libraries by simply writing *import* followed by the name of the library.

*import pandas*

But you’ll want to shorten the name so that every time you reference it you’re not typing out the whole name. (Especially if the name of the library is long, like Beautiful Soup.) To do that, simply write “as” and call it whatever you want to call it. Pandas is typically shortened to ‘pd’ but you could actually call it anything.

*import pandas as pd*

Loading your datasets

Just as you would open a spreadsheet to start analyzing it in Excel, you need to open the files you plan to analyze in your jupyter notebooks. You do this not by clicking on a file name. Instead, we tell the computer to read the file.

Make sure you’ve opened your jupyter notebook in the same folder where your data lives, or the computer won’t be able to find it!

pd.read\_csv('public\_150k\_plus.csv')

You will notice when you paste that code into your notebooks, the computer will spit out a view of your data. (The first five rows and the last five rows.) It’s nice to see what our data looks like, but if we want to continue working with it, we need to assign it a name. In other words, we need to make it a variable.

**Variables**

A Python variable is a reserved memory location to store values. Think of variables as names you assign to represent a dataframe, a column within that dataframe, a list of numbers or words or even a math formula.

* Variables must start with a letter
* No special characters
* Case sensitive

Once you assign a variable, you can use that name in future commands and the computer will understand that it represents the values you assigned to it. You do this by assigning a name, typing a single = and then following with whatever it represents.

For example: df = pd.read\_csv('public\_150k\_plus.csv')

Note that the value of a variable, if it is text, should be in quotes or single quotes. If the value is a number, you don’t need to put it in quotes. But we’ll get to that later.

Try that code your jupyter notebook. Notice how it looks like nothing happens? That usually means the command worked! The computer only does what it is told, so you have to tell it that you want to see your results if that’s what you want.

In order to do that, type:

df.head()

That will return the first five rows. You can see more rows if you specify how many you’d like to see inside the parentheses.

Try:

df.head(10)

You should see 10 rows. Now, if you want to see the last 10 rows of your data, simply type:

df.tail(10)

df.sample() will give you a random sampling your data

**Commands**

Any time you tell the computer to do something with a dataset, you have to tell it what dataset you want it to do that thing on. In the case above, we’re using df. If you had two datasets, you might name the other one df2 and want to run the same commands on it.

Let’s try that with a list of industries that correspond with numeric codes found in the PPP loan data. (What we’re calling df.)

df2 = pd.read\_csv('NAICS.csv', dtype={'2017 NAICS Code': str})

Now ask the computer to display the first 10 rows and the last 10 rows of df2.

You can do this with any many datasets as you are trying to analyze at once. In fact, from here, we will be creating new variables (in this case dataframes) based on the analysis we do. Think about in Excel, when you make a pivot table, it opens a new sheet. Or, if you filter a subset of your data and you open a new sheet and paste the results of your filter. We will be doing that in python, but think of each sheet as a new dataframe, or variable.

**Try out a few basic commands**

You may want to start by telling the computer that you want to see all your columns when you ask to display your data. If you’ve noticed, so far, there are likely ellipses covering up a lot of the columns in the middle when you do the df.head() command.

Try this

#set your notebook so you see all column headers

pd.set\_option('display.max\_columns', None)

And then try df.head()

This may or may not be necessary depending on how many columns you have, but it will save you a headache if you go ahead and use this code before trying to analyze anything.

The following are commands that you’ll likely use to get familiar with your data.

#find out how many rows you have

df.shape()

The df.shape() command is important so you know how many records you have. Always keep an eye on that number in every data frame you’re working with.

#get the datatypes for all records

df.info()

This command will tell you how many non-null (meaning cells that are not empty) values are within each column and what type of data lives in every column. This is how you can find if certain columns are missing too many values to be useful and also if the columns that should be numbers are being read as numbers, and if the columns that should be text are read as strings.

Numbers will be either int64 or float64.

Text will show up as ‘object.’

Notice how ‘LoanNumber’ is a number?

LoanNumber is the unique identifier for every single loan. Unique identifiers are often a combination of letters and numbers or just numbers. Sometimes, a numeric unique ID will start with “0”. Excel will almost always interpret such a column as a number format, and automatically cut the zero off, which is bad because then two distinct things might be counted as the same thing. (In this case, a loan with the ID of 08235 and a loan with the ID of 8235 would be counted as the same thing.)

We don’t want that! Using Python with Comma-Separated-Values files (csv) can rescue you from that problem.

An original csv will contain the original values in that spreadsheet. But we need to tell the computer how to read those values. So let’s try the pd.read\_csv command again, but this time specify the type of data in that LoanNumber column.

Try this:

Df = pd.read\_csv('public\_150k\_plus.csv', dtype={'LoanNumber': str})

You could also convert data types after import. That would look like:

df['LoanNumber'] = df['LoanNumber'].astype(str)

Here, we are writing over the Loan Number column with text values instead of numeric. You could actually create a new column instead of overwriting an existing one by saying df[‘LoanNumberText’] = (or whatever you want to call it). Then you would have two columns for LoanNumber with one being read as text and the other read as a number.

(In the future, when you want to convert something to a number, you would just replace ‘str’ with ‘numeric in the above code. If you’re specifying upon impor, you’d replace ‘str’ with ‘float’ or ‘int.)

Now check the data types.

…

Now it’s an object!

Keep in mind that another big reason you may want to specify your datatypes is when you’re merging two datasets based on one common field, and one dataframe characterises that field as text while the other characterises it as a number. In order to merge the two datasets, the columns in both datasets need to be the same type. (We’re not going to get to merging yet, so just put this thought on the backburner.)

Before moving on, we should also make sure the computer knows that our DateApproved field is a date. See how it’s reading it as an object? If you wanted to sort by date, you wouldn’t be able to.

Here’s how you can fix that.

df['DateApproved'] = pd.to\_datetime(df['DateApproved'])

With this, again we are overwriting the original date column. If you didn’t want to do that, you could call your variable something else and have two date fields. One that is formatted as a date and one that is not.

**MATH**

Now let’s have the computer do some math for us.

Simply type the name of your dataframe, followed by the name of whatever column you want to work on, followed by the command.

This, for example, will give us the average approval amount:

df.CurrentApprovalAmount.mean()

You can also try:

* .sum()
* .max()
* .min()
* .median()

Now let’s try sorting your data so the largest value is at the top.

df.sort\_values('CurrentApprovalAmount', ascending=False).head()

The anatomy of this code is:

* **df** the name of the dataframe we want to work with
* **.sort\_values** the command that we want to use on this data frame
* **(‘CurrentApprovalAmount’** the column we want to sort by
* **,ascending=False** tells the computer to sort largest to smallest. The default is that it will sort smallest to largest. So you have to tell it False so that it knows to do the opposite. If you replace False with True, the computer will sort with the smallest values first.
* **)** closes your parenthese so the computer knows that’s the end of that command
* **.head()** will show you the data

One last thing.

This dataset is unwieldy, so let’s trim it down to just the columns we care about.

To do that, you will make a new variable.

Let’s just call it df3.

Df3 = df[['BorrowerName','BorrowerCity','InitialApprovalAmount','CurrentApprovalAmount', 'RuralUrbanIndicator','Gender','JobsReported']]

Every column name should be in quotes or single quotes and separated by a comma. Note the double brackets here. You need those in order to pull out the columns you want.

You can also rename these. This will come in handy when column headers are too long or complicated. This time, just one set of brackets. Your names still need to be in quotes and separated by commas. You also have to make sure you name each column, even if you keep the original name. You will get an error if you’re missing one.

df3.columns = ['business', 'city', 'initial\_anount', 'current\_amount', 'rural\_urban', 'gender', 'jobs']

Now let’s save our new dataframe as a separate csv, which you can open in excel.

df3.to\_csv('loans.csv')

\*\*You may get an error message if you copy the code that involves quotes directly into jupyter notebooks. Occasionally the ‘ ‘ will get weirdly formatted when you paste them. To fix this, in the jupyter notebook cell, retype your quotes. (Only if you get the error message.)