Python for Data Analysis

R has taken over academic data analysis

- Used to be SPSS
- Then SPSS and STATA (less coding, drop down menus)
- R is free, has more applications, and is constantly updated by open source contributions
  - Scripting language, automate processes, reusable code, etc.
  - Open source allows for rapid model release for academic research advancements
  - R has taken over other software usage for more advanced modeling and analytics
- However, Python is becoming the dominant language of data science

Python dominates outside academia (Social Science)

- Plays well with cloud resources
- Developed by computer scientists
  - Code is cleaner/structurally similar across libraries

- Python is becoming the dominant language of data science

  - Annual Survey from Kaggle Users:
    - https://www.kaggle.com/kaggle-survey-2022

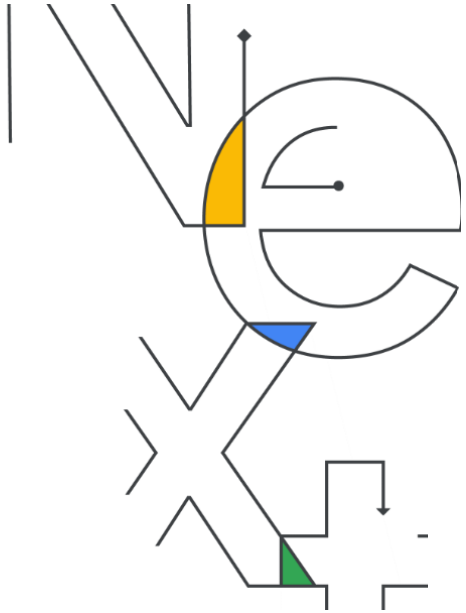
  Kaggle – huge data science and machine learning community – data, code, and other resources. Host machine learning competitions; can submit predictions for competitions

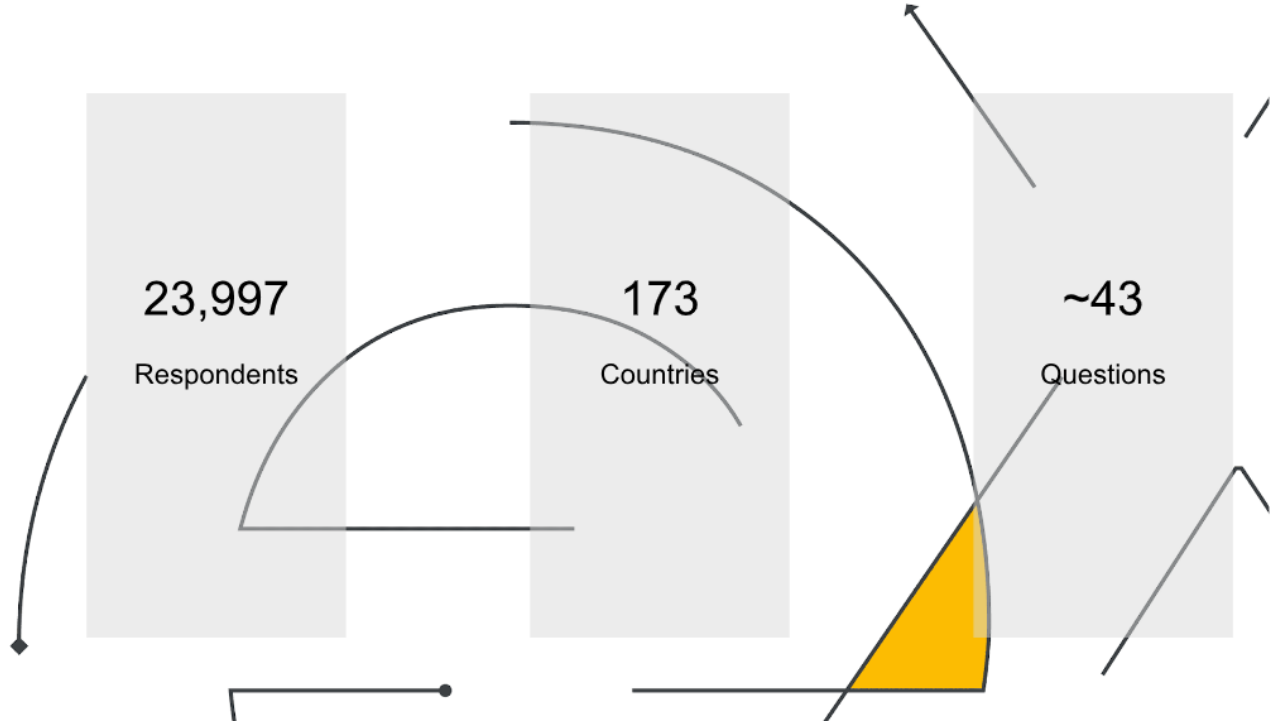
Google Cloud
Next '22

2022 Kaggle Data Science & ML Survey

Data Scientists' backgrounds, preferred technologies, and techniques

Oct/
11–13



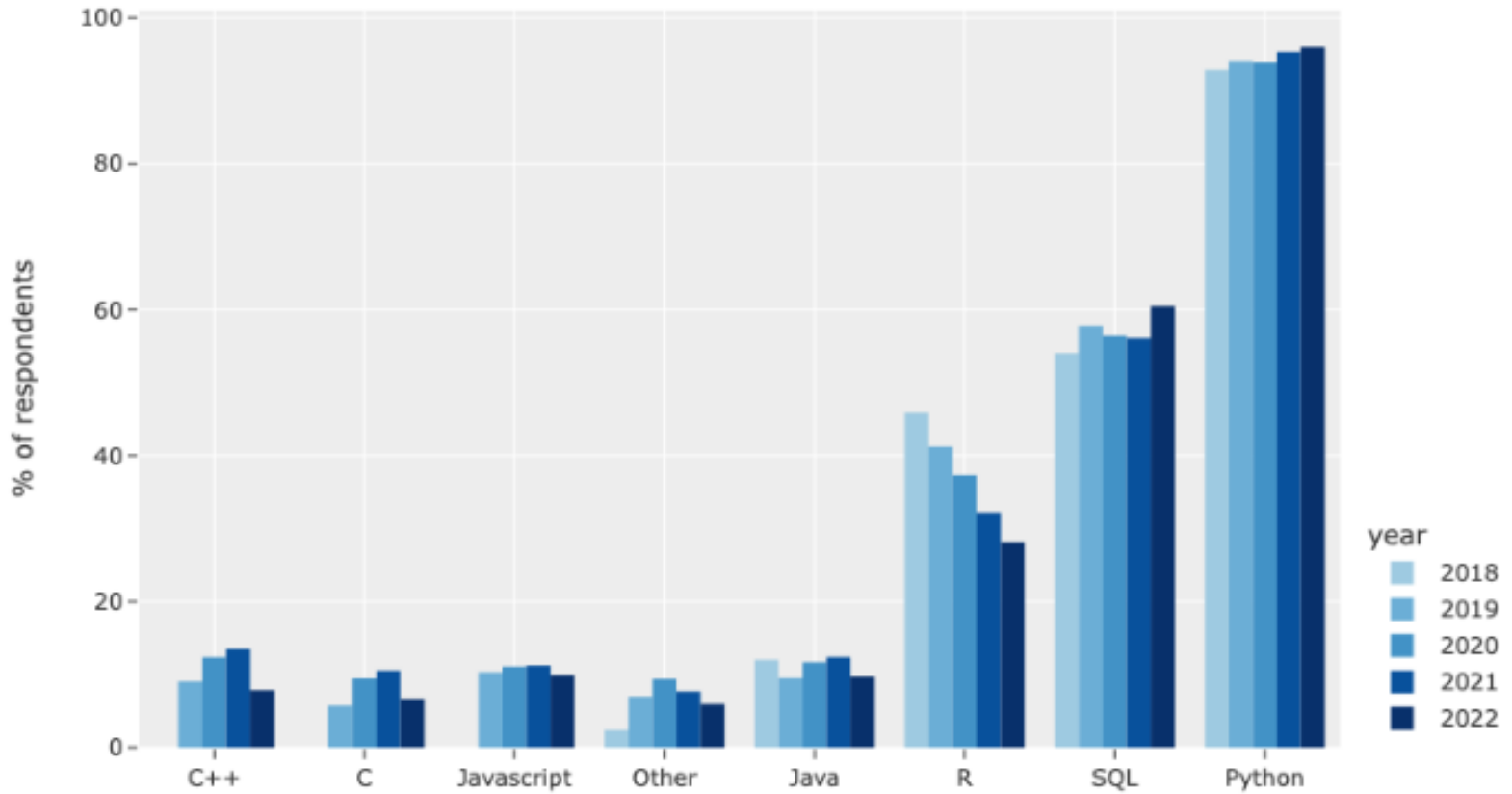


Download full survey results:

https://www.kaggle.com/kaggle-survey-2022

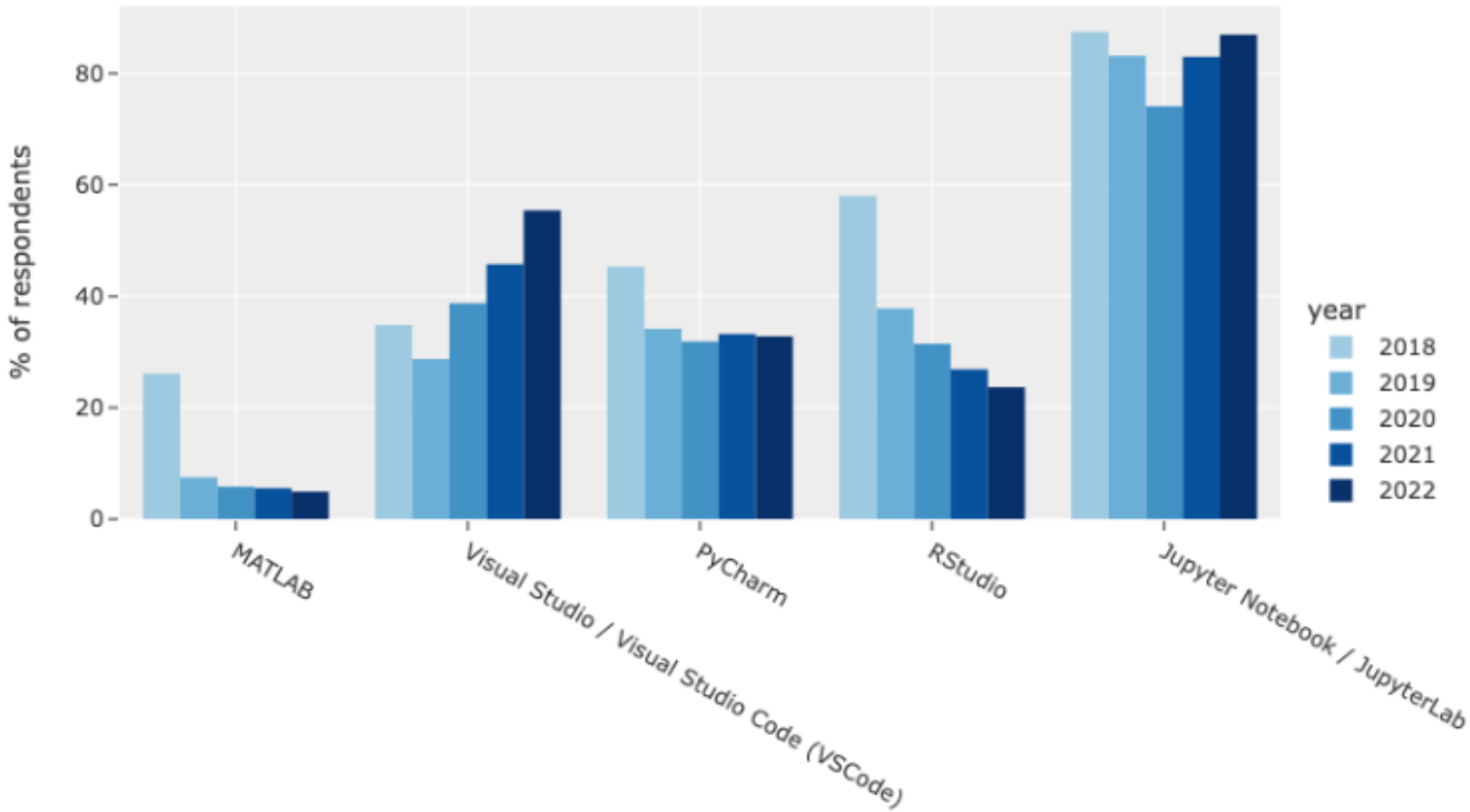
Kaggle DS & ML Survey 2022

Python and SQL remain the two most common programming skills for data scientists

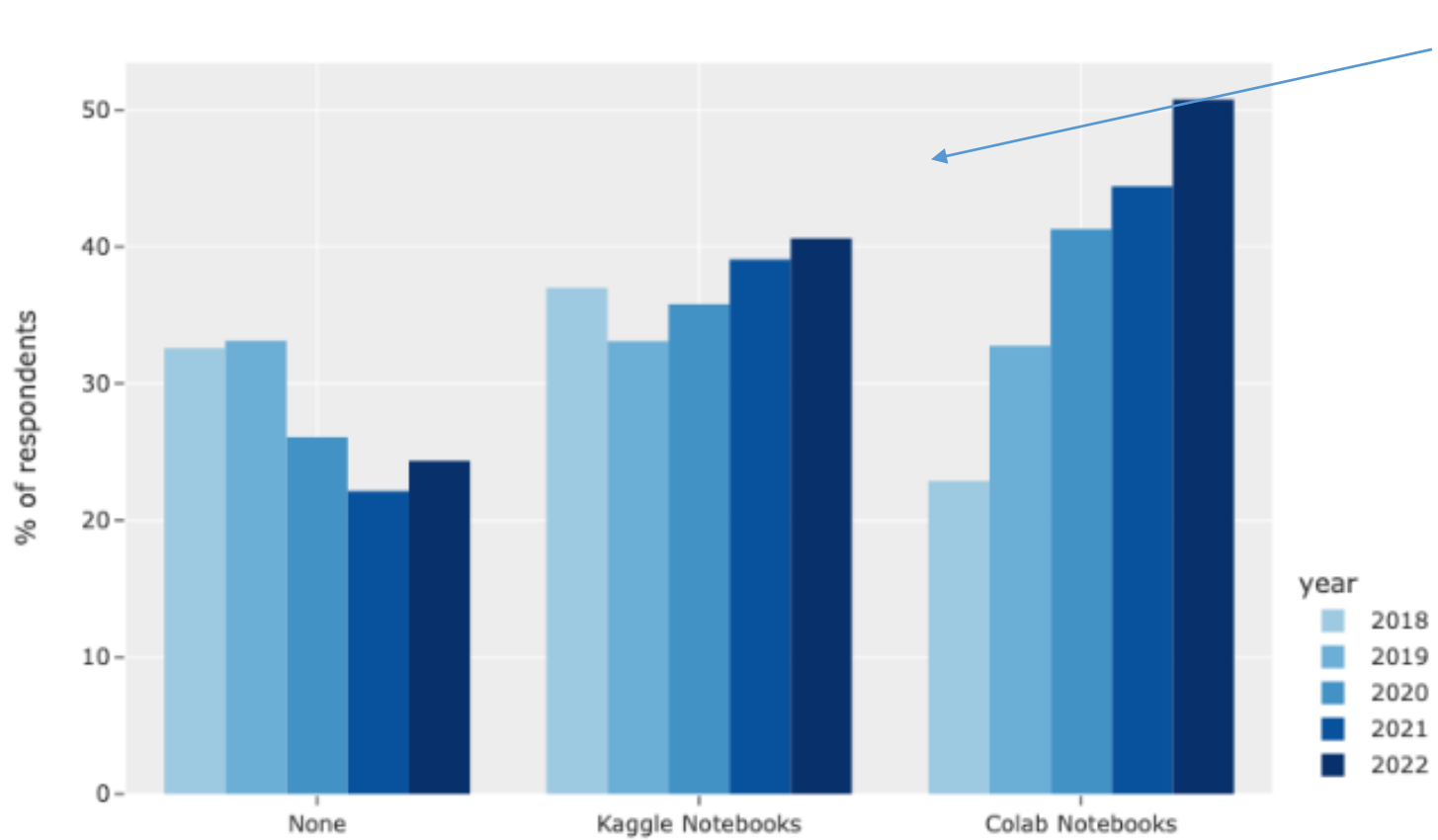


Kaggle DS & ML Survey 2022

VSCode is now used by over 50% of working data scientists

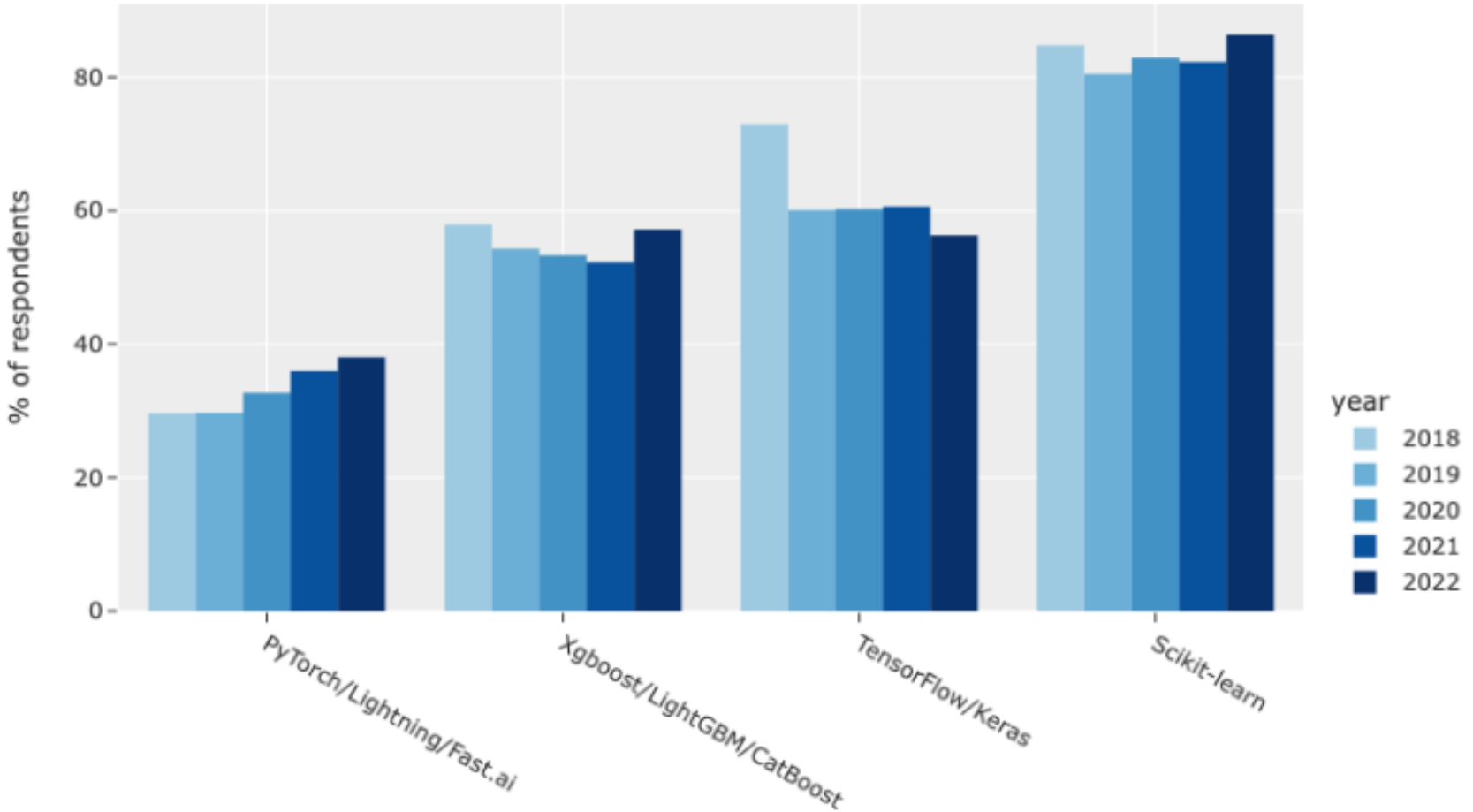


Colab notebooks are the most popular cloud-based Jupyter notebook environment



Colab and Kaggle Notebooks both built over Jupyter Notebooks

Scikit-learn is the most popular ML framework while PyTorch has been growing steadily year-over-year

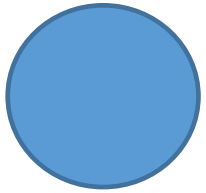


Python use is growing for data analysis

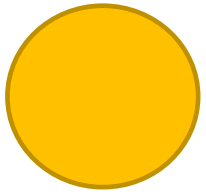
Table 1: Top Analytics/Data Science/ML Software in 2018 KDnuggets Poll

Software	2018 % share	% change 2018 vs 2017
Python	65.6%	11%
RapidMiner	52.7%	65%
R	48.5%	-14%
SQL	39.6%	1%
Excel	39.1%	24%
Anaconda	33.4%	37%
Tensorflow	29.9%	32%
Tableau	26.4%	21%
scikit-learn	24.4%	11%
Keras	22.2%	108%

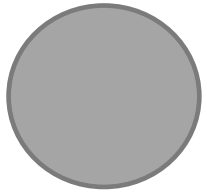
Tutorial Content



Overview of Python Libraries for Data Scientists



Reading Data; Selecting and Filtering the Data; Data manipulation, sorting, grouping, rearranging (focus on tabular data)



Intro to Plotting

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

A library is essentially a collection of code that can be imported, stored and reused. Within the base software there is a standard library and then you can install other user-contributed libraries.

Visualization libraries

- matplotlib
- Seaborn *(built over matplot lib and is easier to use)*

and many more …

Python Libraries for Data Science

NumPy:

- Efficiently organize raw data so that we can do math on the structured data
- Introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects

- Powers many other libraries – i.e., many other Python libraries are built on NumPy

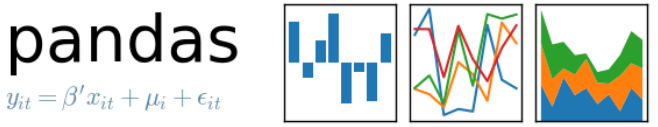
Link: http://www.numpy.org/

Python Libraries for Data Science

SciPy:

- built on NumPy
- uses NumPy to do advanced mathematical equations in an efficient way
- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- will be used by other libraries but we won't use it directly that much this semester (a little when we do tree visualizations and cluster techniques for unsupervised learning)

Link: https://www.scipy.org/scipylib/



Python Libraries for Data Science

Pandas:

- Uses NumPy
- it's how we get tabular data into Python - provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Link: http://pandas.pydata.org/

Python Libraries for Data Science

SciKit-Learn:

- Python library used for classic machine learning
- provides machine learning algorithms: classification, regression, clustering, model validation etc.
- the most used machine learning library

- built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

Python Libraries for Data Science

matplotlib:

- basic visualizations
- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- based on matplotlib
- easier to use - allows you to use less lines of code
- provides high level interface for drawing attractive statistical graphics
- similar (in style) to the popular ggplot2 library in R

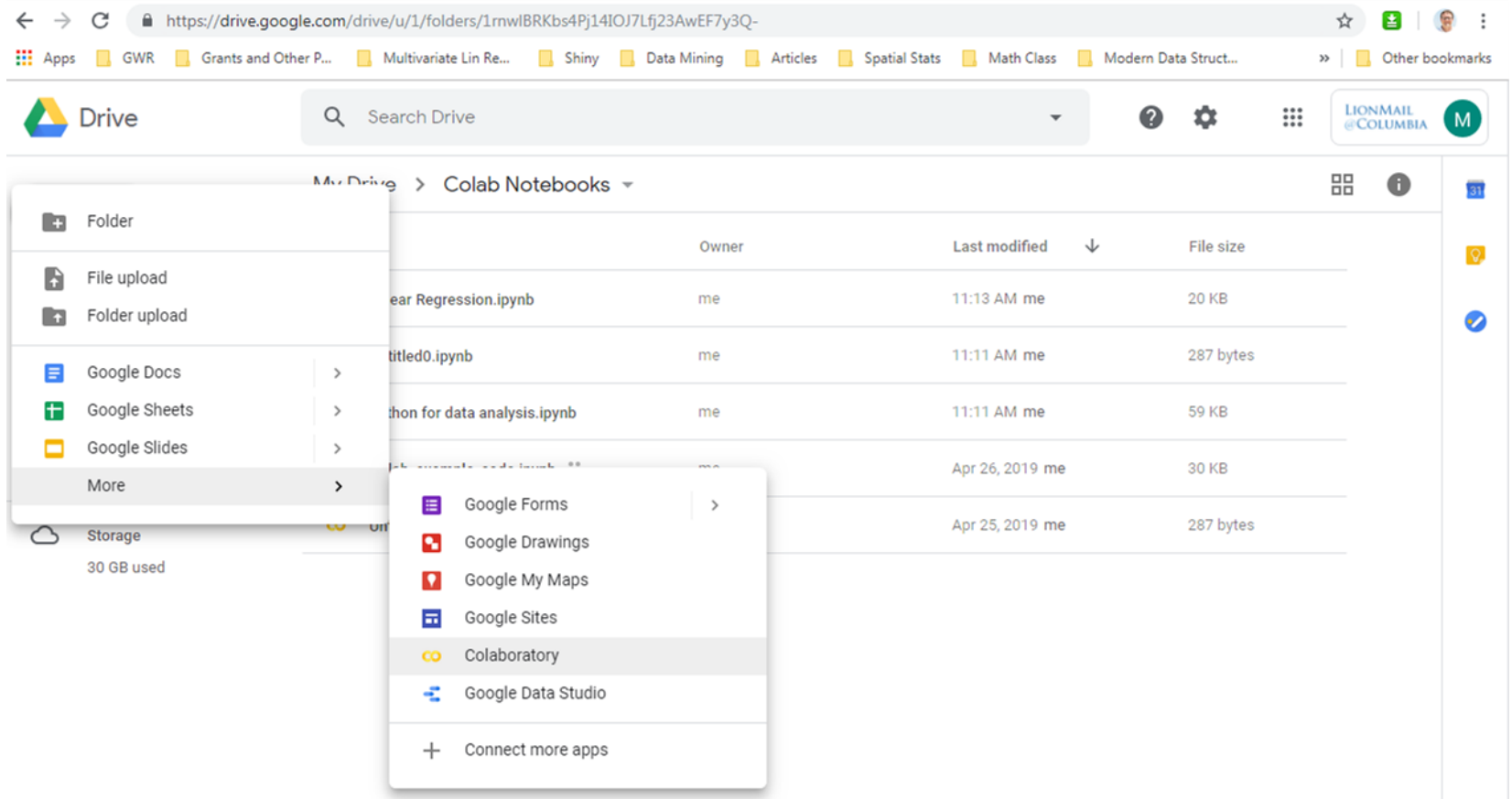
Link: https://seaborn.pydata.org/

2 pathways to Python

- Google Drive - Colab Notebook *(required for this semester)*
  - Notebook interface that connects to cloud computer for free (doesn't use local processors)

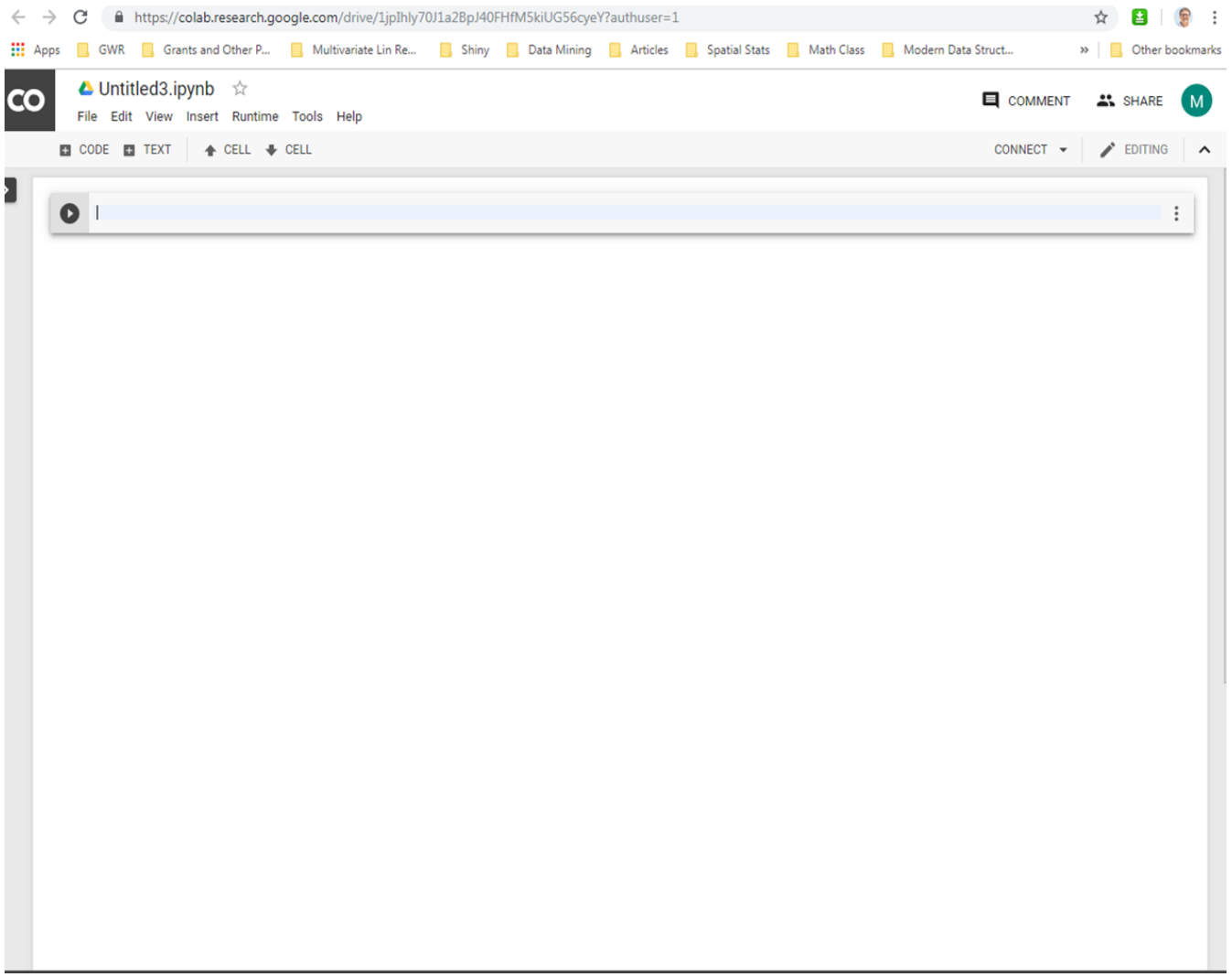
- Local installation through Anaconda

Start Google Colab Jupyter notebook from Google Drive:



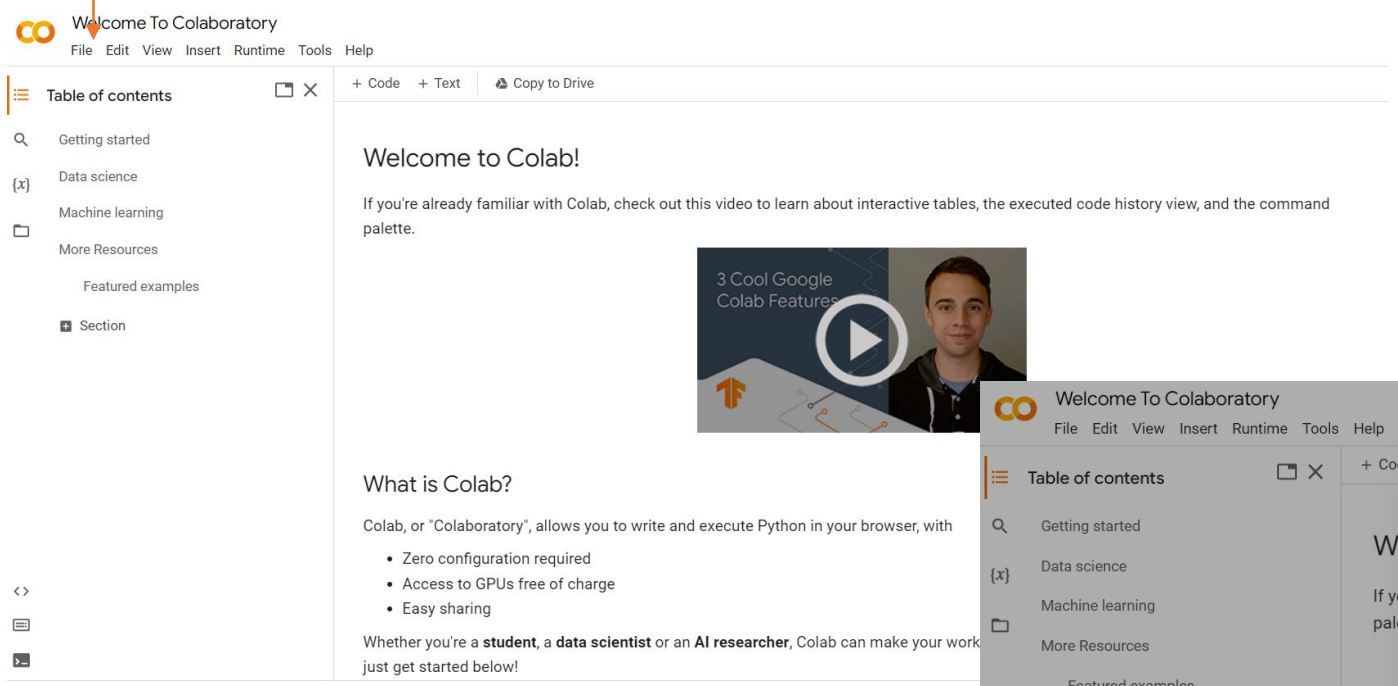
+New -> More -> Google Colaboratory

Start Google Colab Jupyter notebook from Google Drive:

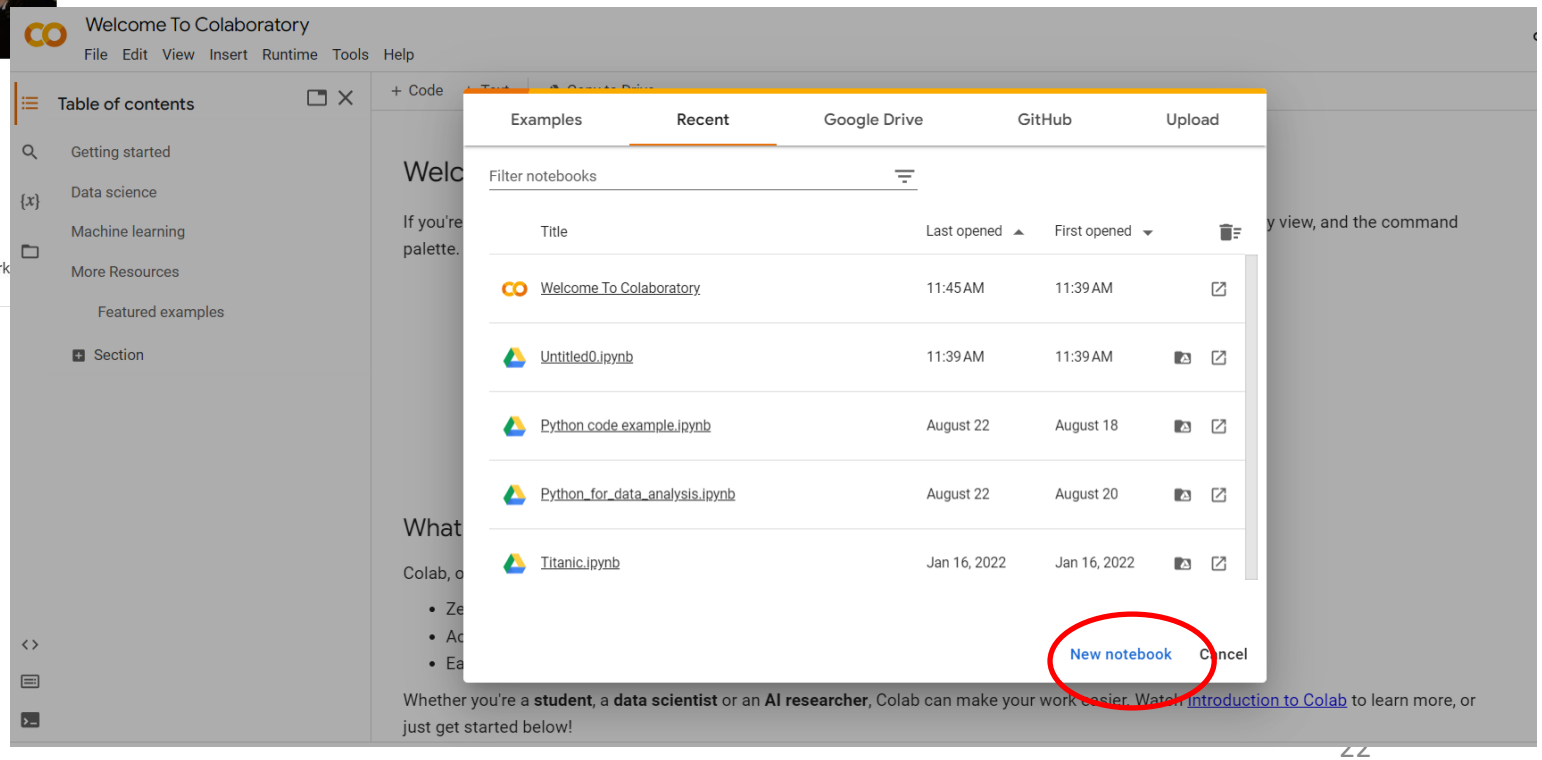


https://colab.research.google.com/

File -> New Notebook



*Or just search for Google Colab if it doesn't come up in your drive.



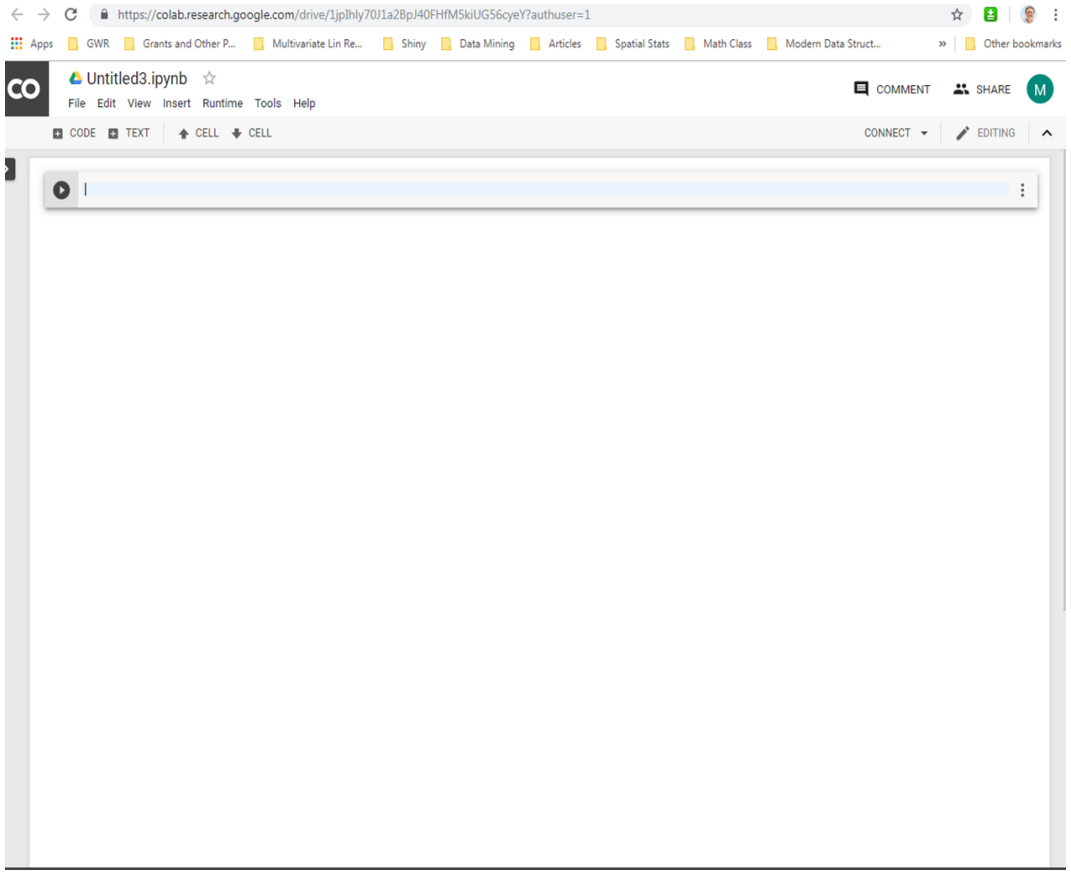
Hands-on exercises

- ✓ Make sure you can open up a Jupyter Notework using Google Colab

Tour of Jupyter Notebook

<u>The Basics</u>

- Two types of cells (code and text)
- Organizing notebooks with markdown
- Saving – download to local computer to upload to Courseworks
- Download .ipynb – the file extension for any Jupyter notebook (Interactive Python Notebook)
  - .py is raw Python code
- Can also share in email and will automatically open
- Upload files for session storage – will be deleted when done unless you save them to drive or locally
  - CSV file to read in with Pandas



Formatting text cells

- Jupyter Notebook supports Markdown, which is a lightweight markup language that allows you to format your text.
- Powered by HTML but simplifies it

Headings

Use number sign (#) followed by blank space for titles and headings:

for titles
for major headings
for subheadings
for 4th level subheadings

Emphasis

Use the following code to emphasize text:

Bold text: __string__ or **string**
Italic text: _string_ or *string*

Line breaks

Sometimes markdown doesn't make line breaks when you want them.
To force a linebreak, use the following code:

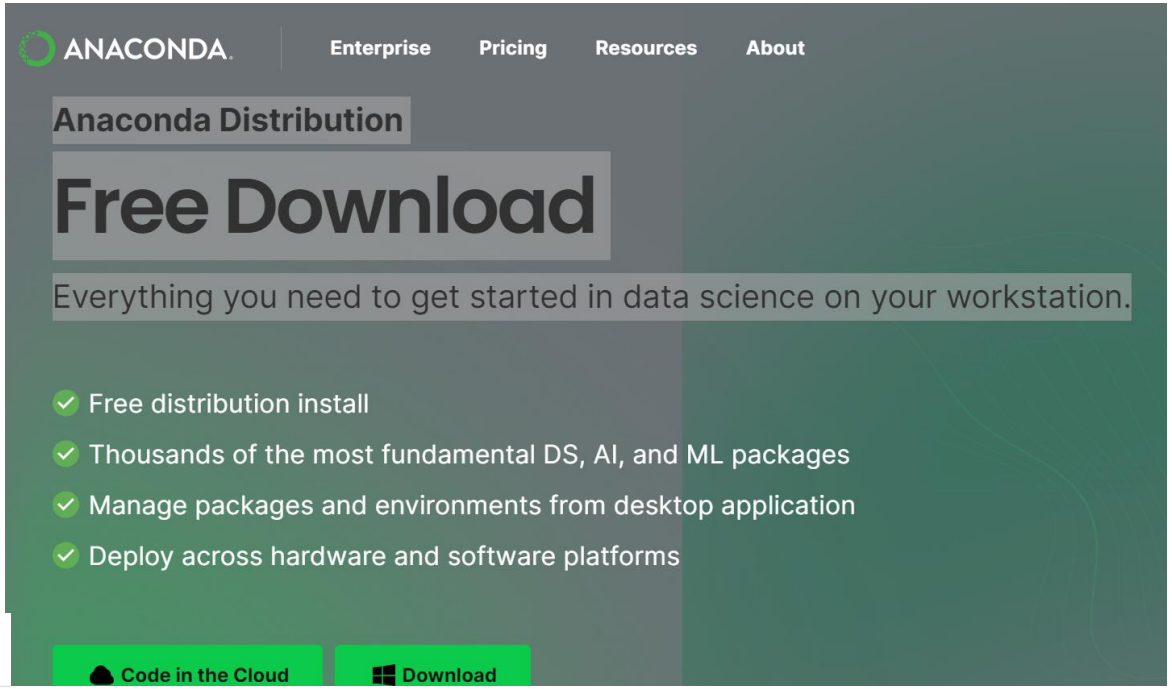
Hands-on exercises

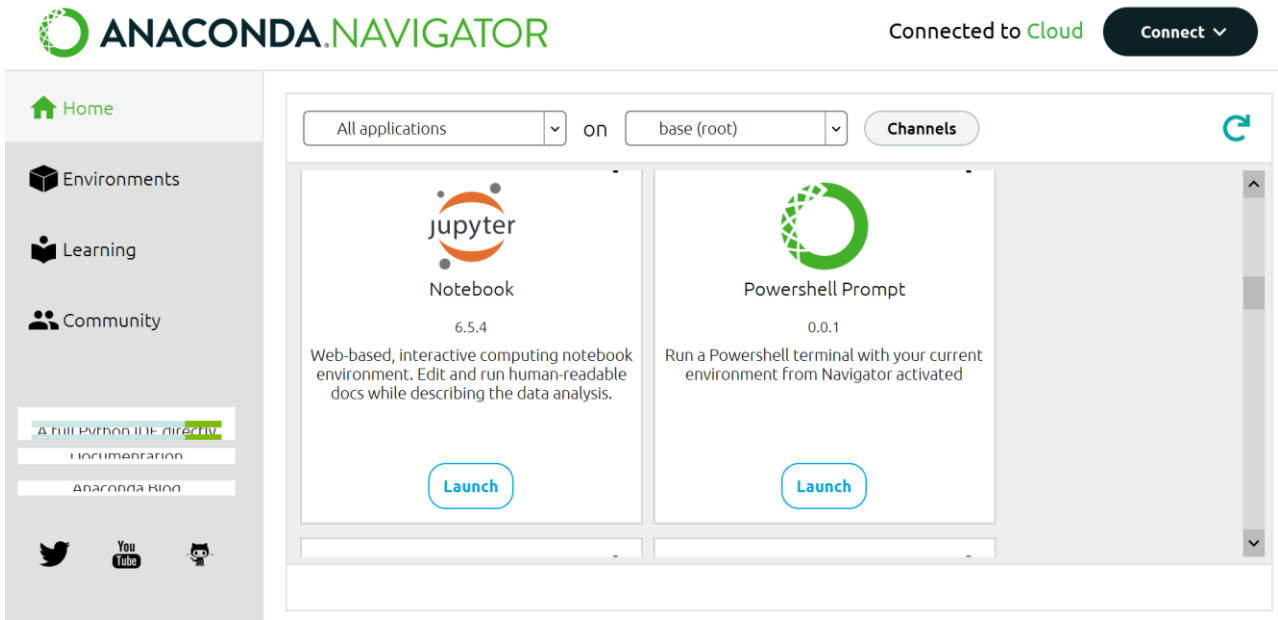
- ✓ Play around with Text and Code cells (add, delete, reorganize, format text, etc.)
- ✓ Name your notebook
- ✓ Download .ipynb

Local Python installation

https://www.anaconda.com/download

- Most popular way is Anaconda Distribution
- Free download, comes already set up with many core libraries pre-installed
- Access to Jupyter Notebooks
- Plugged into local computer





Loading Python Libraries

```
In [ ]: #Import Python Libraries
        import numpy as np
        import scipy as sp
        import pandas as pd
        import matplotlib as mpl
        import seaborn as sns
```

-Load up all code from a library. We can rename to make it easier to refer to the library when we need to (different than R)

-Good practice to import libraries at top of notebook

Press `Shift+Enter` to execute the *jupyter* cell

Reading data using pandas

```
In [ ]: #Read csv file
        df=pd.read_csv("https://raw.githubusercontent.com/Apress/data-analysis-and-visualization-using-python/master/Ch07/Salaries.csv")
```

Note: The above command can include other optional arguments to fine-tune the data import process.

There are several pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

`read_csv` is a function that takes inputs and processes them into outputs (functions are referred to as methods in Python)

The CSV file is the input. The output is the object we've named df.

Single equal sign (=) assigns the name df to the object The double equal sign (==) is the equality operator. It is used to compare the values of two expressions or objects. If the values are equal, the operator returns True; otherwise, it returns False.

Exploring data frames

```
In [3]: #List first 5 records
        df.head()
```

Out[3]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Note: Rows start with an index value of 0.
Columns in dataframes have names.

✏ Hands-on exercises (five minutes)

✓ Learn more about the method (i.e.-function) with question marks

  ✓ Run ?df.head() to learn about head method args. (in Colab you will need to select and hover over the function)

✓ Try to read the first 10, 20, 50 records (by adding and changing an input argument)

✓ Can you guess how to view the last few records;

✓ *Hint: Flip a coin and get heads or ???*

df.head ×

```
def head(n: int=5) -> NDFrameT
```

Return the first n rows.

This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

For negative values of n, this function returns all rows except the last |n| rows, equivalent to df[:n].

If n is larger than the number of rows, this function returns all rows.

Parameters

n : int, default 5
Number of rows to select.

Returns

same type as caller
The first n rows of the caller object.

See Also

DataFrame.tail: Returns the last n rows.

Examples

```
>>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
...                    'monkey', 'parrot', 'shark', 'whale', 'zebra']})
>>> df
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
5     parrot
6      shark
7      whale
8      zebra
```

Viewing the first 5 lines

```
>>> df.head()
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
```

Viewing the first n lines (three in this case)

```
>>> df.head(3)
      animal
0  alligator
1        bee
2     falcon
```

For negative values of n

```
>>> df.head(-3)
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
5     parrot
```

Data Frame data types

Data frames can have different types of data in each column.

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the number of character it can hold.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types – Code to explore the data frame

```
In [4]: #Check a particular column type
        df['salary'].dtype
```

```
Out[4]: dtype('int64')
```

`['salary']` will isolate the salary column.

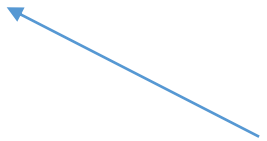
- dtype is an attribute, not a function (or method). dtype prints out data for a particular column. Follows this structure: dot notation and without ( ).

- An attribute is a variable that is stored on an object. A method is a function associated with an object. Methods can be used to perform actions on objects or to access the attributes of objects.
- Main difference is that attributes store data, while methods perform actions. Attributes can be accessed using the dot notation.

Data Frame data types – Code to explore the data frame

```
In [5]: #Check types for all the columns
        df.dtypes
```

```
Out[5]: rank          object
        discipline    object
        phd           int64
        service       int64
        sex           object
        salary        int64
        dtype: object
```



dtype: object
Pandas includes the type of information that you are printing out. This output is text.

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns `df.dtypes`
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions (data frames and matrices have 2, not as useful)
size	number of elements (how many total cells are there)
shape	return a tuple (collection of objects separated by commas) representing the dimensionality (i.e. how many rows and columns)
values	numpy representation of the data (just the raw data, without columns and row names)

Hands-on exercises (5 minutes)

✓ Find how many records this data frame has

✓ How many elements are there?

✓ What are the column names?

✓ What types of columns we have in this data frame?

Hands-on exercises

✓ Find how many records this data frame (df) has `df.shape` will give # of rows

✓ How many elements are there? `df.size`

✓ What are the column names? `df.columns`

✓ What types of columns we have in this data frame? `df.dtypes`

Data Frames methods

Unlike attributes, python methods have *parenthesis.*
All attributes and methods can be listed with a *dir()* function: `dir(df)`

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Hands-on exercises (5 minutes)

✓ Give the summary for the numeric columns in the dataset

✓ Calculate standard deviation for all numeric columns;

✓ What are the mean values of the first 50 records in the dataset? ***Hint:*** use head() method to subset the first 50 records and then calculate the mean

Hands-on exercises

✓ Give the summary for the numeric columns in the dataset `df.describe()`

✓ Calculate standard deviation for all numeric columns; `df.std()`

✓ What are the mean values of the first 50 records in the dataset? ***Hint:*** use head() method to subset the first 50 records and then calculate the mean

```
df2=df.head(50)
df2.mean(numeric_only=True)
```

Subsetting data

- Next week, we'll need to get data ready for our prediction models
- SciKit-Learn – requires data to be separated into two objects
  - Single Y variable (dependent variable, target feature)
  - X data – explanatory variables, control variables

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

df['sex'] ← Isolate data in a column by using this structure:
`['column name']`

Method 2: Use the column name as an attribute:

df.sex ← Column names are automatically attributes

Note: there is an attribute *rank* for pandas data frames, so to select a column with a name "rank" we should use method 1

Hands-on exercises (5 minutes)

✓ Calculate the basic statistics for the *salary* column;

✓ Find how many values in the *salary* column (use *count* method);

✓ Calculate the average salary;

Hands-on exercises

✓ Calculate the basic statistics for the *salary* column; `y=df['salary']`

✓ Find how many values in the *salary* column (use *count* method); `y.count()`

✓ Calculate the average salary; `y.mean()`

Explore categories and subsets of data

- Similar to pivot tables in excel
- Organize data based on specific categories in a column
- For example, organize data by the different values in the Rank Column (Prof, Assoc Prof, Asst Prof, etc.) and calculate average salary by the ranks

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group
- Similar to dplyr() function in R

In [ ]:
```
#Group data using rank
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#Calculate mean value for each numeric column per each group
df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Data Frames *groupby* method

Once groupby object is created we can calculate various statistics for each group:

```
In [ ]: #Calculate mean salary for each professor rank:
        df.groupby('rank')[['salary']].mean()
```

rank	salary
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping

- by default the group keys are sorted during the *groupby* operation. You may want to pass sort=False for potential speedup:

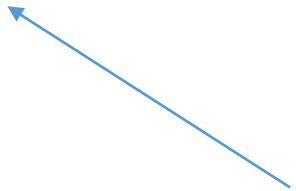
In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby(['rank'], sort=False)[['salary']].mean()
```

Data Frame: filtering

Subsetting on rows

To subset the data we can apply a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

```
In [ ]: #Identifies rows in which salary is greater than 120000:
        df_sub = df[ df['salary'] > 120000 ]
```



True or False depending on whether this condition is met. Anything True is subsetted. Output is all rows with salary > 120000

Data Frame: filtering

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female' ]
```

Hands-on exercises (5 minutes)

- ✓ Go through some examples of subsetting on columns / filtering on row
- ✓ Change the operators that define your subset
- ✓ Use shape or other attributes to compare subset to original data to make sure you've filtered out the data appropriately

Data Frames: Slicing

Removing columns, subsetting rows, outputting objects. Motivation is how to slice up data to get it ready for the models.

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

In [ ]:
```
#Select column salary:
df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

In [ ]:
```
#Select columns rank and salary:
df[['rank','salary']]
```

Data Frames: Selecting rows

If we need to select a range of rows, we can specify the range using ":"

```
In [ ]: #Select rows by their position:
        df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted.

So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9.

Data Frames: method loc

If we need to select a range of rows, using their labels we can use method loc:

In [ ]:
```
#Select rows by their labels:
df_sub.loc[:,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

- Uses [ ] instead of ( ) that we'd normally use with a function
- Index for rows and names for columns
- **:** means you want to select all rows (if you wanted only some rows you would do something like 0:20
- And only want Rank, Sex and Salary columns

Data Frames: method iloc

Use index values for both rows and columns
If we need to select a range of rows and/or columns, using their positions we can use method iloc:

In [ ]:
```
#Select rows by their labels:
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

0 is the first column

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # All rows and First column
df.iloc[:, -1] # All rows and Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #All rows and First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  #1st and 6th rows and 2nd and 4th columns
```

Data Frames: Sorting

We can sort the data by a value in the column using the `sort_value` method. By default the sorting will occur in ascending order and a new data frame is returned.

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Missing Values

Missing values are marked as NaN

In [ ]:
```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

Out[ ]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Missing Values

There are a number of methods to deal with missing values in the data frame (will have a class dedicated to missing data later in the semester:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have *skipna* option to control if missing data should be excluded . This value is set to *True* by default (unlike R)

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max
count, sum, prod
mean, median, mode, mad
std, var

Aggregation Functions in Pandas

agg() method are useful when multiple statistics are computed per column:

In [ ]:
```
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

Graphics to explore the data

Seaborn package is built on matplotlib but provides high level interface for drawing attractive statistical graphics, similar to ggplot2 library in R.

It specifically targets statistical data visualization

Matplotlib introductory examples

Can also import subfolders of a library (may need to save memory)

(Libraries are also known as Modules and sub-folders are sub-modules)

Line chart code

In [ ]:

```
from matplotlib import pyplot as plt

years=[1950,1960,1970,1980,1990,2000,2010]
gdp=[300.2,543.3,1075.9,2862.5,5979.6,10289.7,14958.3]

# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years,gdp,color='green',marker='o',linestyle='solid')

# add a title

plt.title("Nominal GDP")

# add a label to the y-axis
plt.ylabel("Billions of $")

plt.show() # code to print out final chart
```

Two inputs for a line chart.
These are both Python lists

Check out other examples in the Notebook