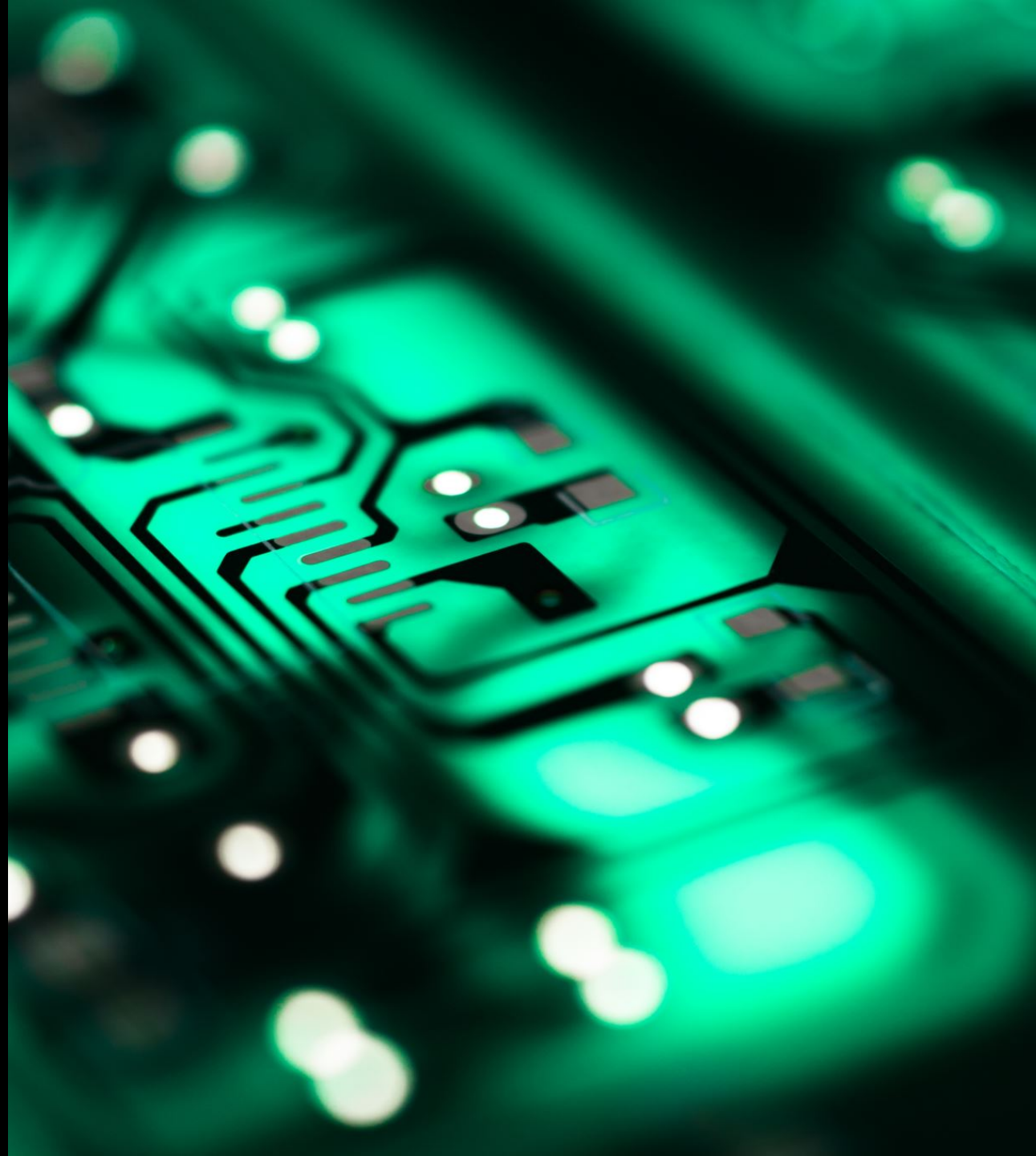
Introduction to biomedical data science

Serghei Mangul, Ph.D

Assistant Professor of Clinical Pharmacy and Biological Sciences, University of Southern California



Data take the center stage

editorial

Data take centre stage

We are updating our editorial policies to further encourage authors to make their data publicly accessible. Publishing Extended Data figures and source data online will also ensure that data are given a more prominent role.

A Data availability statement for accepted articles has been requested by the Nature Research journals for the past three years[1]. In this statement, our authors declare how the data behind their published research can be accessed by interested readers, and disclose any potential restriction limiting data sharing. This initiative, aiming at increasing the reproducibility of our papers, has been seamlessly accepted by researchers, who have promptly added this declaration to their manuscripts submitted to the Nature titles. However, in the vast majority of cases data have remained locked up in the authors' drawers, and allowed to see the light only 'upon request'.

In their contribution to the Why it Matters column, Natasha Noy, computer scientist at Google Research, and Aleksandr Noy, materials scientist at the Lawrence Livermore National Laboratory in California, USA, reveal that a disappointingly small fraction of the papers published in the first 10 issues of *Nature Materials* in 2019 provided publicly accessible links to data repositories. Yet for materials scientists there are plenty of good reasons to share data — for instance, their results can be merged with those of other labs to create larger datasets, and used for further statistical analysis or for a better training of machine learning algorithms for materials discovery. In addition, several research funders already mandate the public availability of data generated by the researchers they financially support; similarly, publicly funded large-scale facilities require their users to give unrestricted access to their results. Above all, sharing promotes transparency and pushes scientists to scrutinize their own laboratory practices related to the production, presentation and storage of research output.

Public repositories, which provide digital identifiers to facilitate proper credit attribution to authors and ensure permanent preservation of the data, are widespread on the Internet. For instance, a list of repositories endorsed by the relevant scientific communities for specific datasets can be found in the Nature Research policies on availability of data (https://www.nature.com/nature-research/editorial-policies/reporting-standards#availability-of-data) — and the list is rapidly growing. Discovery of these datasets is also becoming increasingly easier thanks to the development of dedicated search engines, such as Google Dataset Search (https://toolbox.google.com/datasetsearch), developed by Natasha Noy and her colleagues.

Credit: Peter Cade/Getty

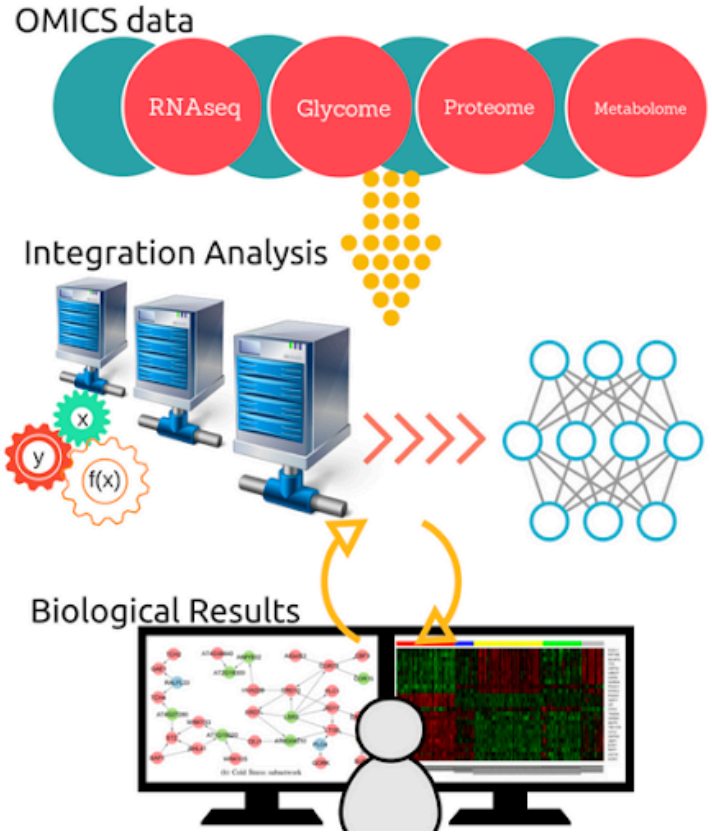
To make the whole process work efficiently, an extra step is needed from researchers. They have to prepare their data in a shareable form and add metadata that describe the data measured, the techniques and experimental conditions used, and other information key to understanding the content provided. Yet Natasha and Aleksandr Noy explain that several repositories are streamlining this procedure by generating such metadata automatically from the details provided by the authors during upload, and other tools are available to generate them independently.

Now, to further encourage our authors to make that extra step, our journals have modified the requirements for the preparation of the Data availability statement on acceptance of scientific articles for publication. If data, as well as custom computer code used in the article, can be made available only upon request, we will ask for an explanation of the reasons in the published statement. We hope that reflecting on any assumed barriers to sharing will help authors remove them whenever possible, and transparently motivate those cases where specific roadblocks exist that cannot be overcome (for instance, for the presence of sensitive information or personal data that cannot be anonymized).

We are also providing more space for data in the web pages of *Nature Materials*. Following the example of *Nature*, we will now allow the publication of up to ten Extended Data figures, which will be integrated in the HTML version of the paper and added to the main PDF online (but not included in the printed version). This will give these figures more visibility with respect to the results presented in the Supplementary Information, and as such researchers are welcome to use them to report key supporting evidence that strengthens the claims of the paper. Importantly, authors are also encouraged to provide 'source data' for all the figures reported in the main manuscript and Extended Data. These could be unprocessed source images for electrophoretic gels and blots (which we already asked to include in the Supplementary Information, https://www.nature.com/nature-research/editorial-policies/image-integrity) or microscopy data, as well as tabulated numerical data underlying the graphs presented in the article. A link to the related source data, which are published online as individual supplementary files, will be provided at the bottom of each figure in the HTML version of the paper. Hopefully this will make life easier for researchers that wish to perform additional analyses on published data and that, until now, had to use dedicated software to extract data points from our charts. Both Extended Data figures and source data will not be mandated — authors will decide whether to make use of these features or not. Like with the use of public repositories, however, we expect a large uptake from the whole materials science community.

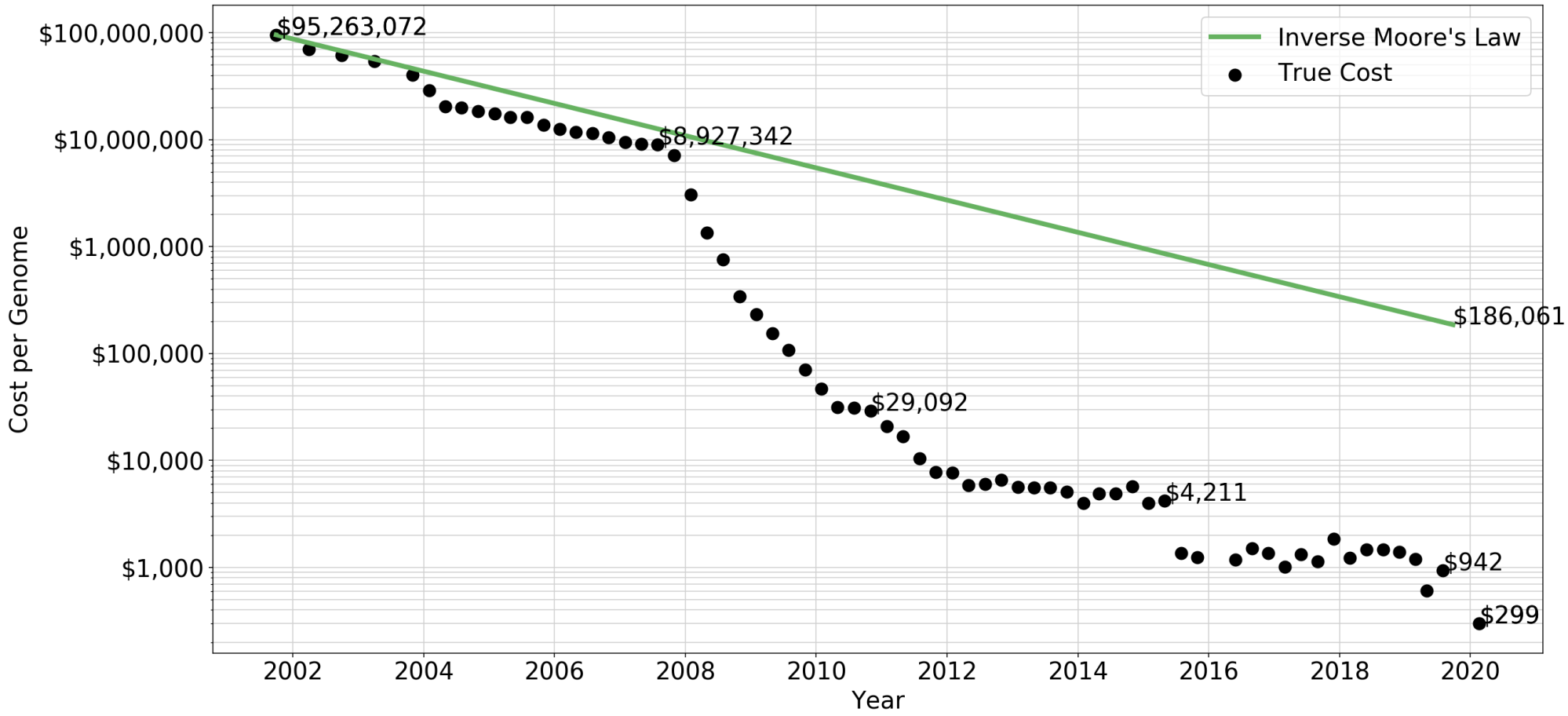
Data are the backbone of every scientific discovery; they are the ground on which researchers confront opinions, debate theories and build collaborations. Unlock them from your drawers and hard drives, and give them the visibility they deserve. ❐

Published online: 18 December 2019
https://doi.org/10.1038/s41563-019-0574-2

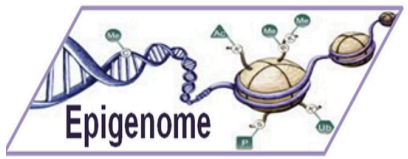


http://cellmanufacturingusa.org/thrust-1

Advances in Next Generation Sequencing



Flood of omics data



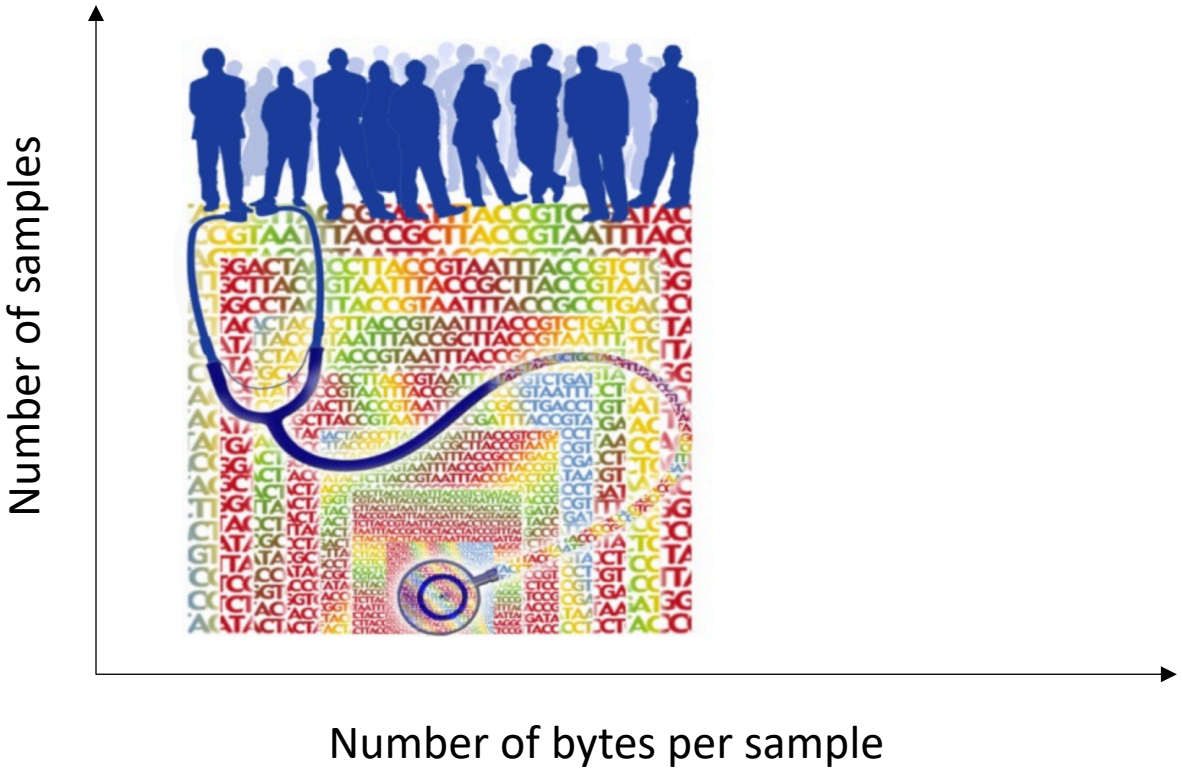
Big omics data

Big data represented by massive datasets represent a substantial challenge for the analysis due to an increased computational footprint associated with handling, processing, and moving information

Applying computational methods to the big data will provide the power to make **novel biological, translational and clinical** discoveries and push the boundaries of current knowledge of the biology of disease

S. Mangul, Interpreting and integrating big data in the life sciences, *Emerging Topics in Life Sciences*

Importance of computational tools for (big) data



Introduction to large-scale biomedical data analysis and visualization

Short description

- This workshop is specifically designed for beginners and researchers with **no prior coding experience**. It covers the basics of biomedical data analysis and visualization using **Python**. The workshop will focus on training students with computational techniques for biomedical data analysis and visualization to ensure reproducibility.

Goal

- Train graduate students with state-of-the-art computational techniques for biomedical data analysis and visualization.

Learning objectives

- Understand and identify the difference between key terms of data science, (e.g. data, code, repository)
- Effectively visualize the data using modern visualization techniques. Understand distributions, sampling, and perform appropriate statistical tests
- Effectively use git for version control and collaborative work. Learn how to share the code and data to ensure the reproducibility of data analysis and visualization
- Use UNIX command line to access and operate a high-performance USC cluster. Perform basic data cleansing, data manipulations, and transformations
- Understand the basics of RNA-Seq analysis. Use state of the art tools to obtain gene expressions and detect differentially expressed genes (e.g. Salmon, DeSeq2)
- Apply the learned techniques to available datasets

Module 1

Interactive notebooks

Perform basic data cleansing, data manipulations, and transformations

Understand and identify the difference between key terms of data science, (e.g. data, code, repository)

Biomedical data science

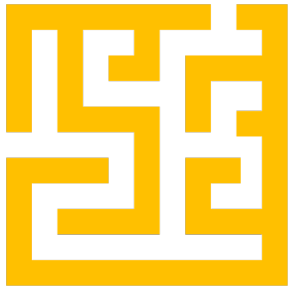
Biomedical data science is an interdisciplinary field that combines computer science and biology to research, analyze and interpret **large scale** of **biomedical data**

Usually is large and requires code-based solutions to process, interpret and visualize

Many samples or data points – need to repeat the same procedures multiple times

Why do data science?

As data becomes larger, the human capacity to manually analyze the data becomes limited



The more data there is to analyze, the more chance is that one would need code-based solutions

I don’t plan to run big data analysis

- Is this workshop still relevant to me?

Code availability

All code required to produce the figures and analysis performed in this paper are freely available at https://github.com/Mangul-Lab-USC/ImReP_publication .

Data availability

- All data required to produce the figures and analysis performed in this paper are freely available at https://github.com/Mangul-Lab-USC/ImReP_publication and are available in the Source Data zip file, including the data used to produce Figs. 2a–f, 3, 4a, c, 5a, b, and 6a, b, and Supplementary Figs. 4, 5, 6d, 7, 8, 9a–c, 10–12, 13a– c, and 14a–c. Source data are provided with this paper.

1. Comply with strict journal requirement on data and code availability

Data availability

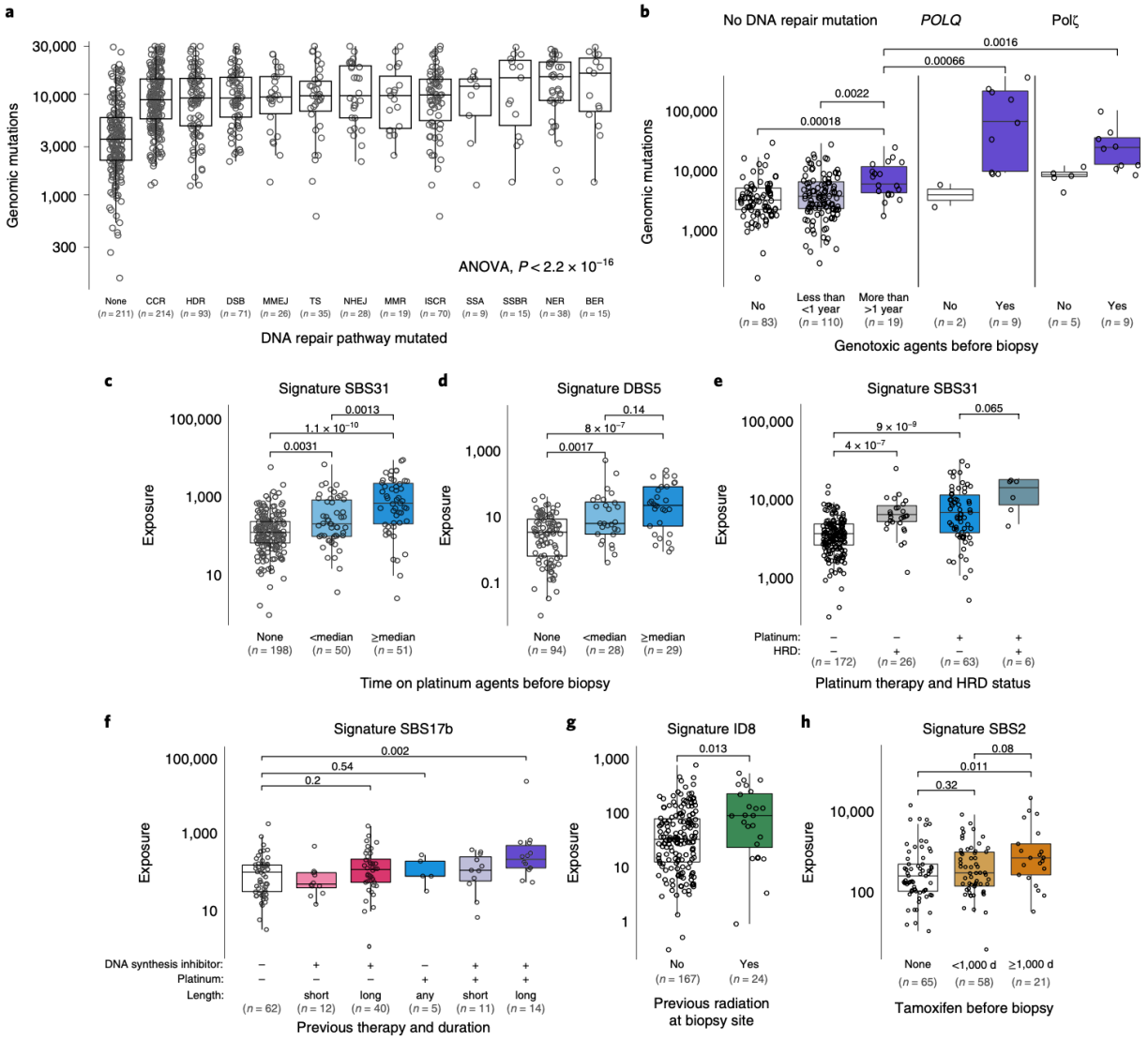
Genomic and transcriptomic sequence datasets, including metadata with library construction and sequencing approaches have been deposited at the European Genome–phenome Archive (EGA, http://www.ebi.ac.uk/ega/) as part of the study EGAS00001001159 with accession numbers as listed in Supplementary Table 1. Data on mutations, copy changes and expression from tumor samples in the POG program organized by OncoTree classification (http://oncotree.mskcc.org) are also accessible from https://www.personalizedoncogenomics.org/cbioportal/. The complete small mutation catalog and gene expression TPMs are available for download from http://bcgsc.ca/downloads/POG570/. Previously published TCGA and PCAWG data that were re-analyzed here are available from data portals (https://portal.gdc.cancer.gov/ and https://dcc.icgc.org/) with sample barcodes as listed in Supplementary Table 9. All other data supporting the findings of this study are available from the corresponding author on reasonable request.

Code availability

The bioinformatics analyses were performed using open-source software, including Burrows–Wheeler alignment tool (v.0.5.7 for up to 125 bp reads and v.0.7.6a for 150 bp reads), CNAseq[91] (v.0.0.6), APOLLOH[92] (v.0.1.1), SAMtools[93] (v.0.1.17), MutationSeq[94] (v.1.0.2 and v.4.3.5), Strelka[95] (v.1.0.6), SNPEff[96] (v.3.2 for somatic and v.4.1 for germline), ABySS[97] (v.1.3.4), TransABySS[97,98] (v.1.4.10), Chimerascan[99] (v.0.4.5), DeFuse[100] (v.0.6.2), Manta[101] (v.1.0.0), Delly[102] (v.0.7.3), MAVIS[71] (v.2.1.1), STAR[73] (v.2.5.2b), RSEM[74] (v.1.3.0), MSIsensor[76] (v.0.2), HRDtools[16] (v.0.0.0.9), BioBloomTools[78] (v.2.0.11b), EXPANDS[84] (v.2.1.1), SignIT (https://github.com/eyzhao/SignIT), samtools[93] (v.0.1.17), ClinVar[103] (v.20180905), InterVar[88], ControlFREEC[104] (v.5), CIBERSORT[89] (v.1.04), Jaguar[105] (v.2.0.3), MiXCR[90] (Java, v.2.1.2) and VDJtools[106] (v.1.1.9). Additional packages used for meta-analyses include R packages ClusteredMutations (v.1.0.1), vegan (v.2.5.3), ConsensusClusterPlus (v.1.44.0), ComplexHeatmap (v.1.18.1), survival (v.2.42.3), survminer (v.0.4.2) and Python package scikit-learn[86] (Python, v.0.20). Additional processing involved in-house scripts that are available upon request.

Pleasance, Erin, et al. "Pan-cancer analysis of advanced patient tumors reveals interactions between therapy and genomic landscapes." *Nature Cancer* (2020): 1-17.

2. Comply with strict journal requirement on data visualization (e.g. show sample size and display all data point addition to the box plot)



Pleasance, Erin, et al. "Pan-cancer analysis of advanced patient tumors reveals interactions between therapy and genomic landscapes." *Nature Cancer* (2020): 1-17.

Where to run the analysis?

LAPTOP

CLOUD

HIGH-PERFORMANCE CLUSTER

https://itservices.usc.edu/hpc/

Where to run the analysis?

LAPTOP

CLOUD

HIGH-PERFORMANCE CLUSTER

https://itservices.usc.edu/hpc/

Why Python?

Python is an extremely popular programming language for data science

Both a powerful language and easy to write code

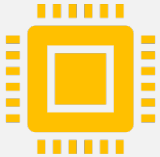
Python vs. R

Stereotypes

Python is for programmers

R is for statisticians

Ultimately, anything you can do in R you can also do in Python



Python is a full programming language and can be used across a range of computer science tasks

Interactive notebook

- A Notebook is an interactive, editable document defined by code. It’s a computer program, but one that’s designed to be easier to read and write by humans.” — Mike Bostock

Interactive notebooks

- A tool for exploratory data analysis and visualization
- Provide an interface for iteratively running code, exploring results and visualizing data
- You can run code, explore the output, tweak the code and re-evaluate the result
- Allow effectively share visualizations and code

Example of interactive notebook

4	imrep	0.54	0.64	0.64	0.62	0.61	0.64	0.64	0.61	Recall	75

In [5]:
```
df_melt = pd.melt(eval_df, id_vars = ['tool', 'Measure', 'Read Length'],
                  value_vars = ['1', '2', '4', '8', '16', '32', '64', '128'],
                  var_name = 'Coverage', value_name = 'Value')
df_melt.columns = ["Tool", "Measure", "Read length", "Coverage", "Value"]
df_melt.head()
```

Out[5]:

	Tool	Measure	Read length	Coverage	Value
0	imrep	Recall	50	1	0.13
1	mixcr	Recall	50	1	0.01
2	imseq	Recall	50	1	0.00
3	igblast	Recall	50	1	0.00
4	imrep	Recall	75	1	0.54

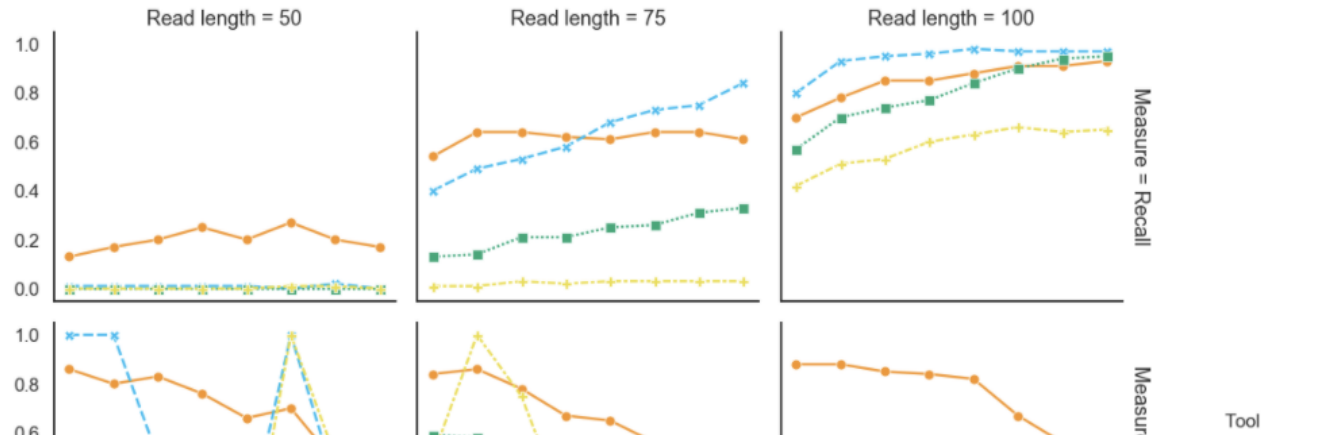
In [6]:
```
colors = ['#F7941D', '#00b9f2', '#00a875', '#ecde38']
sns.set_palette(sns.color_palette(colors))
```

Figure 2A

In [7]:
```
sns.set_style('white')
sns.set_context('talk')

g = sns.FacetGrid(df_melt, row = 'Measure', col = 'Read length', height = 4, aspect = 1.2
g = (g.map(sns.lineplot, 'Coverage', 'Value', hue = 'Tool', style = 'Tool', markers = Tru
          sort=False, err_style=None, data = df_melt).add_legend())
for ax in g.axes.flat:
    ax.set_ylabel('')

plt.savefig('../figures/Figure2_a.png')
plt.savefig('../figures/Figure2_a.pdf')
```



Tools for data analysis

Jupyter notebook is an open-source project developed by a large community of contributors. It is used by many academic and business organizations

Google Colab is s a **free cloud-based** Jupyter notebook, which executes code on Google's cloud servers. Meaning you can leverage the power of Google hardware

Google Colab

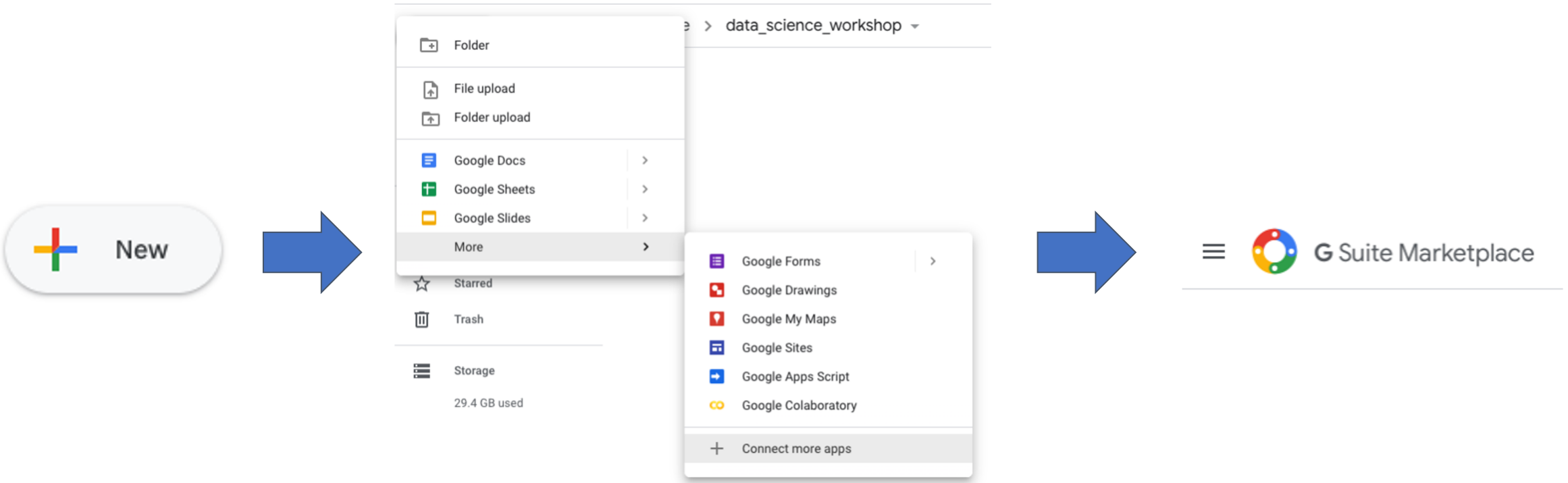
- Gets stored in your Google Drive
- Can be shared just like a Google Doc
- Separated into text blocks and code blocks

Let’s set up Google Collab

- USC Google Drive currently does not support Google Collab
- Use personal gmail account or open a new one

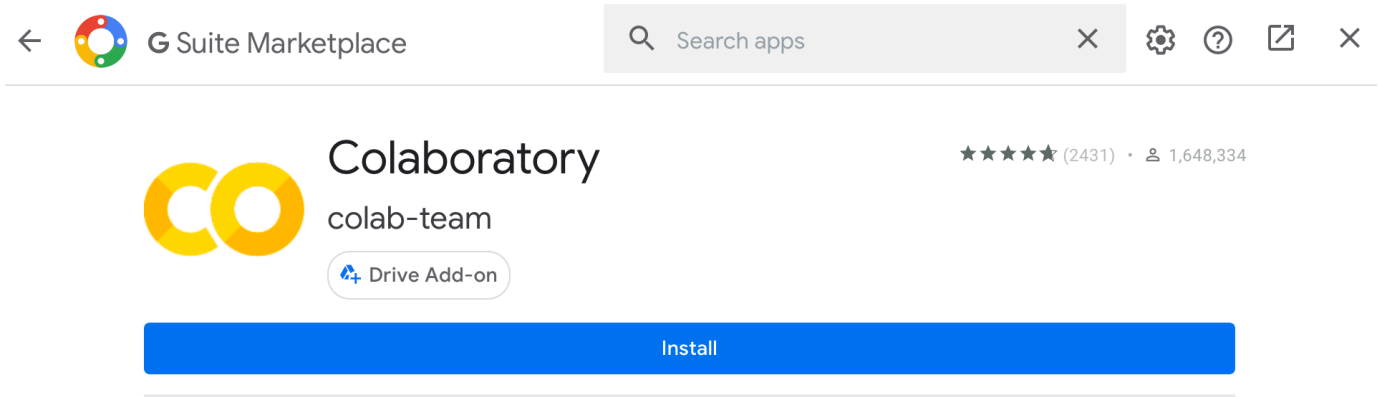
Install Google Collab

Select “New” >>> “More” >>> “Connect more Apps”



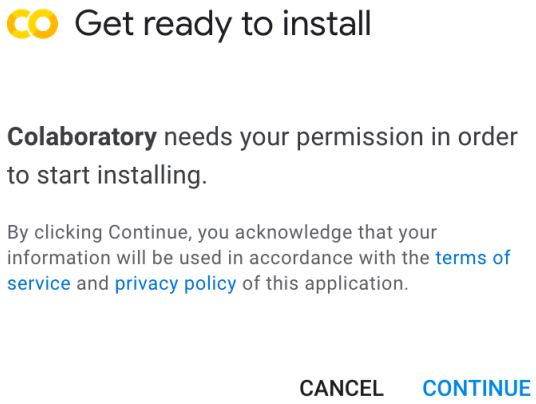
Install Google Collab

Search for Colaboratory, and install it



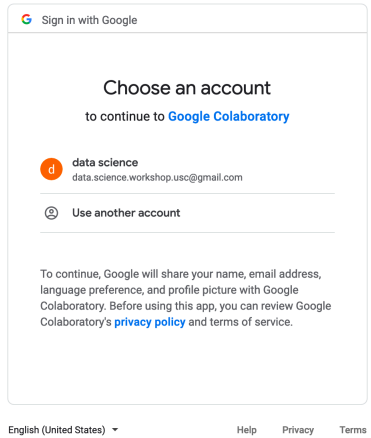
Click

1.

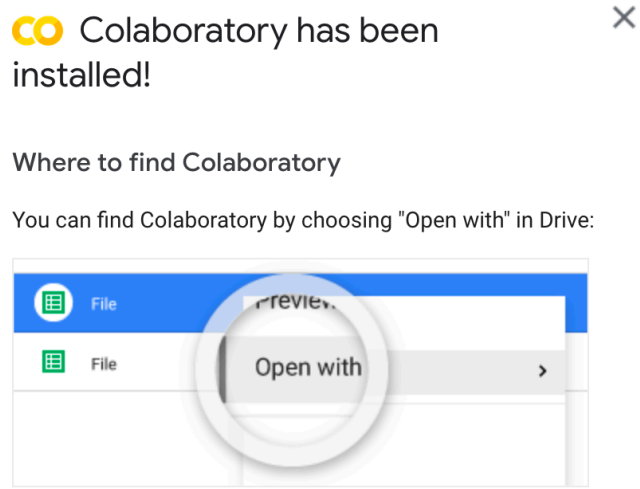


Click

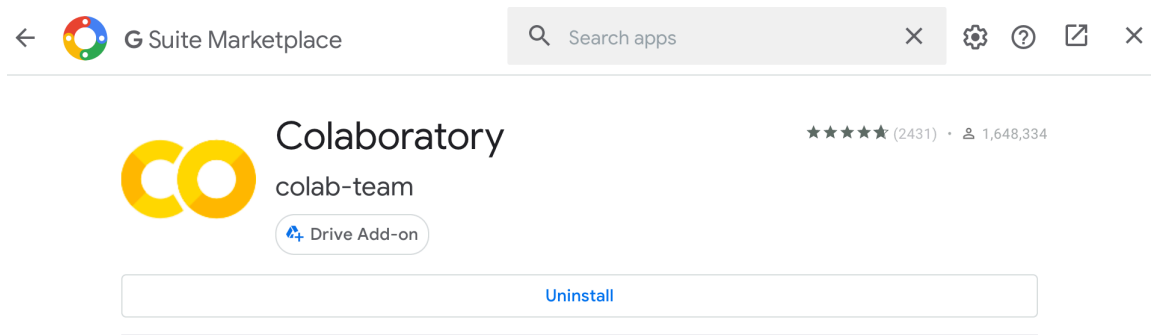
2.



3.



Click

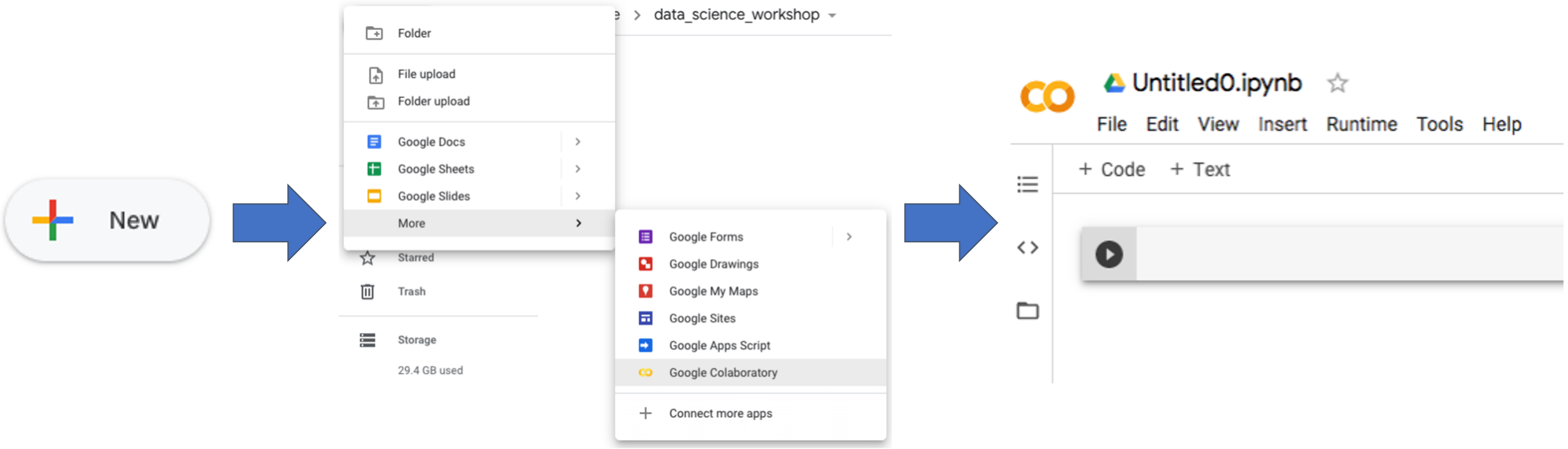


Open a directory for data analysis

1. Go to your Google Drive and create a new folder to hold a new project
    - Not necessary, but useful for keeping all things related to the colab in one place

Create a new Google Colab notebook

Go to “New” >>> “More” >>> “Google Colaboratory”



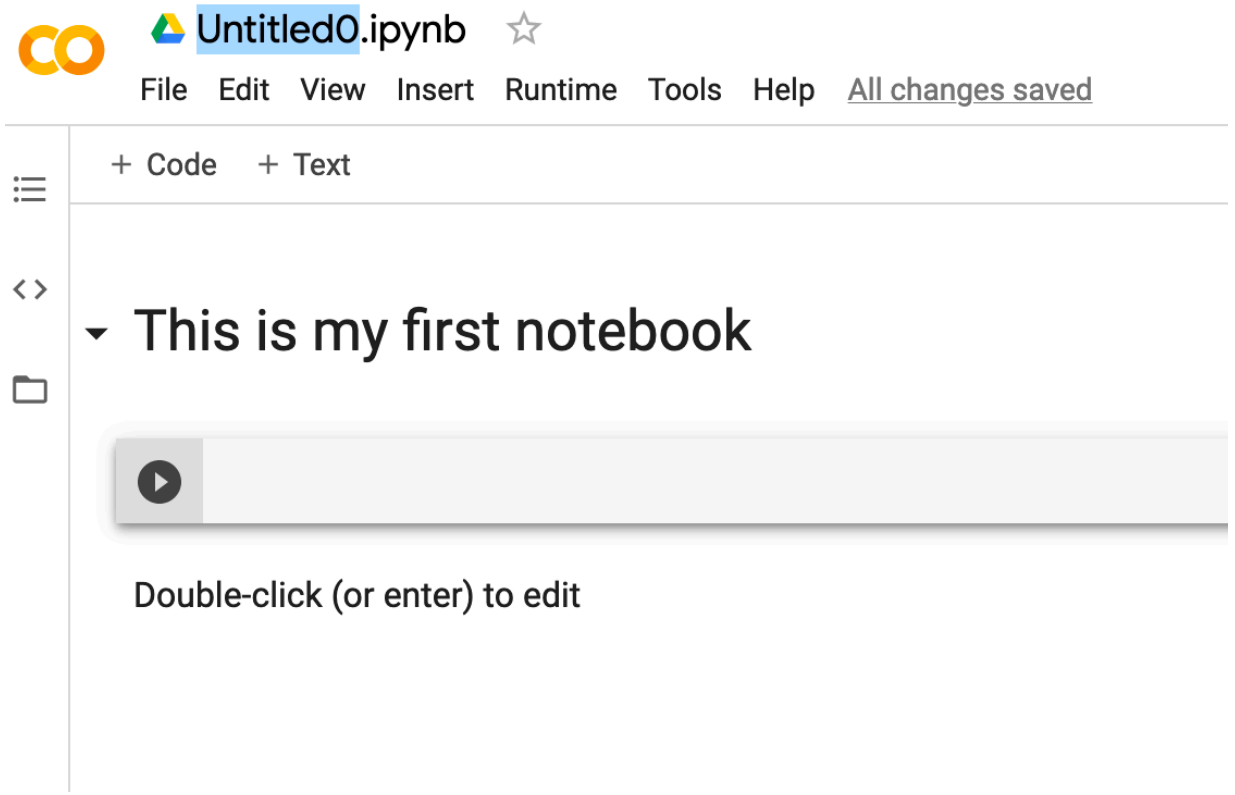
Best practice

- Keep all datasets and notebooks related to the certain project in one place
- For this workshop, all files and notebooks will be stored in

My Drive/data_science_workshop

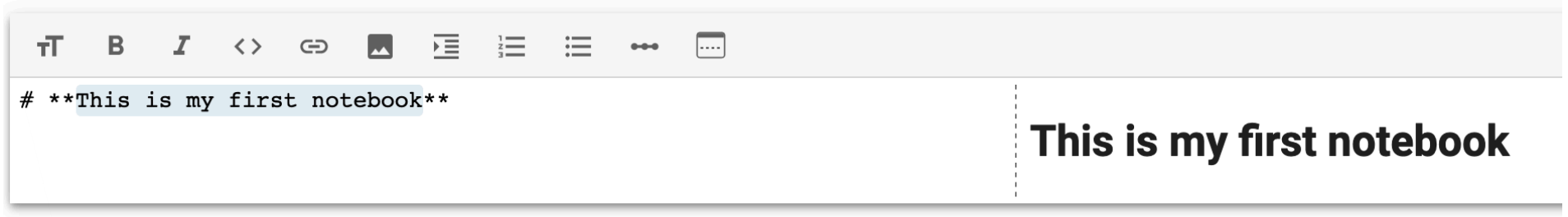
Create a block of text or code

Click



Creating blocks of text

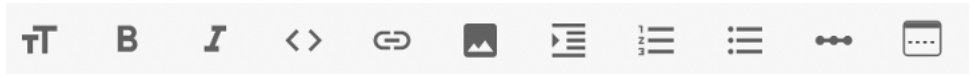
- Text blocks use markdown language



Strong emphasis, aka bold,
with **asterisks**

The right side previews what is written in the left

Header 1

Or you can click 

Creating blocks of code

Code blocks use Python

The code block can have any number of lines of code

Usually we want to limit our blocks to smaller chunks

Oftentimes one block will be the code to draw one plot

Sometimes a block will only hold 1 lines

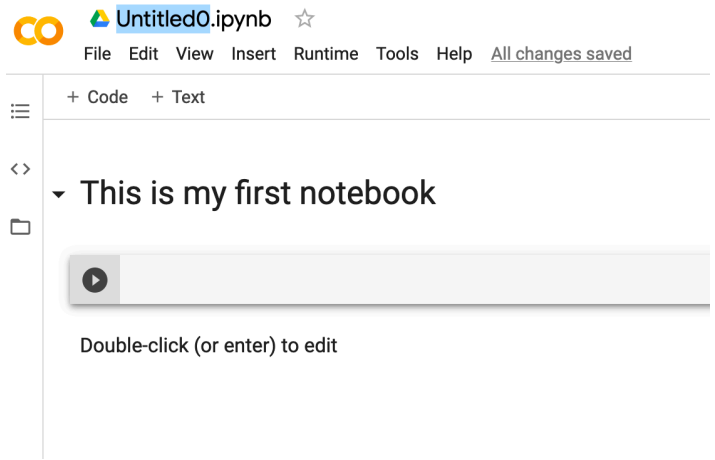
Let’s create a code block

We will write a Hello World program in our new Colab notebook

All we will use is the print() function

Let’s create a code block

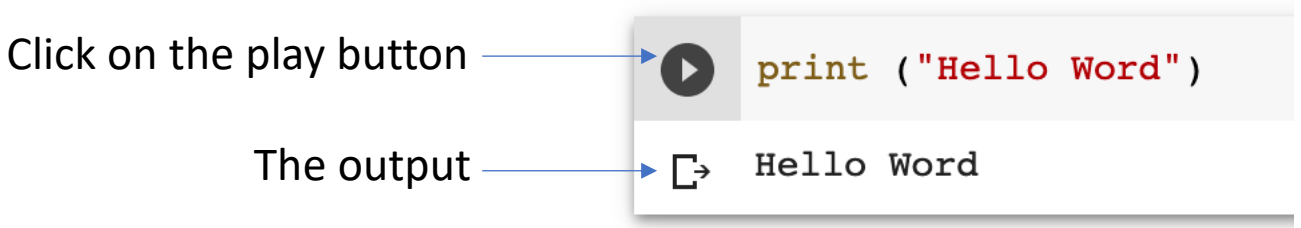
1. Click



2. Type

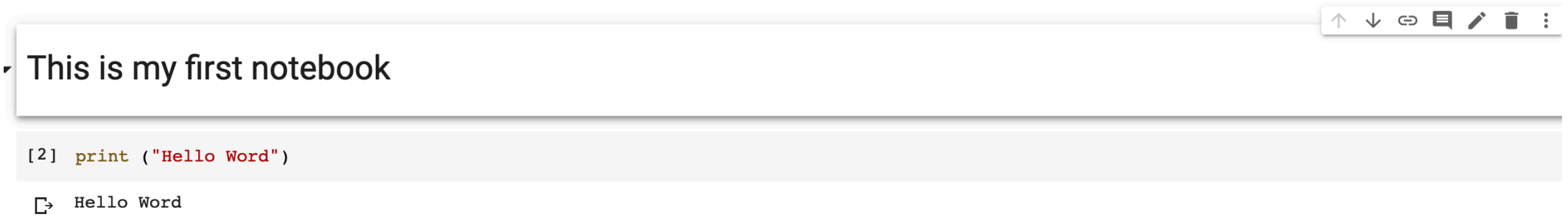
Running a code block

- After putting typing the code in the block, you have some options to run it
  - Click on the play button to the left of the code block
  - Use the shortcut command-return or control-enter
- The output will appear below the code block



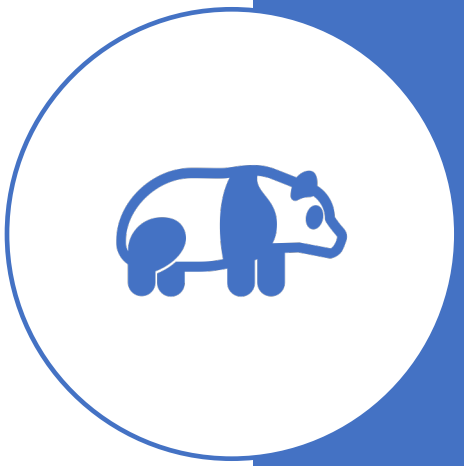
Code/text block settings

- Note that in the upper right corner of a selected block, there are additional settings allowing you to:
  - Delete a block
  - Make a comment on a block
  - Move a block up or down the notebook



Pandas

- In computer programming, pandas is a **software library** written for the Python for data manipulation and analysis
- It is centered around the **data frames** that holds data in a table format with row indexes and column names
- Also allows easy reading and writing to/from comma-separated value (CSV) files



Test data in CSV format

- For this module we will use a very simple and small CSV files
  - https://github.com/Mangul-Lab-USC/biomedical_data_science/tree/master/workshop_materials

Click

5 lines (5 sloc) | 178 Bytes — Raw | Blame | History

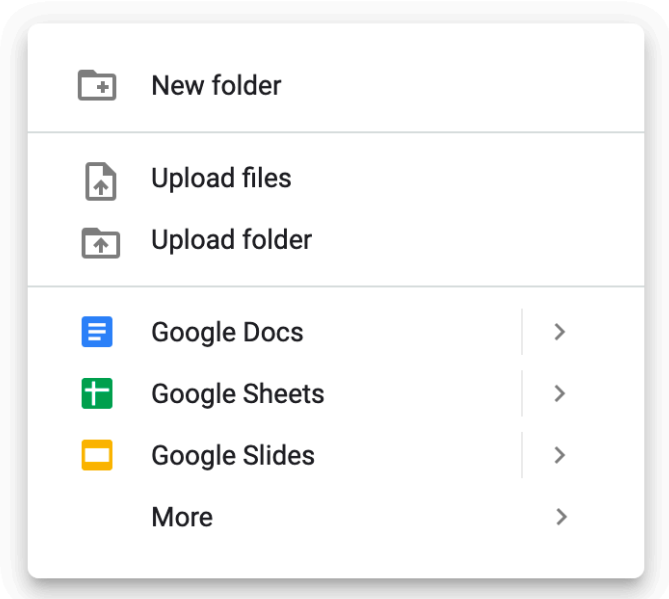
Search this file...

	column0	column1	column2	column3	column4	sample_name
2	control	yes	0.25	0.1	0	sample1
3	case	yes	0.9	0.25	0.75	sample2
4	control	yes	0.43	0.23	0.02	sample3
5	case	no	0.84	0.34	0.9	sample4

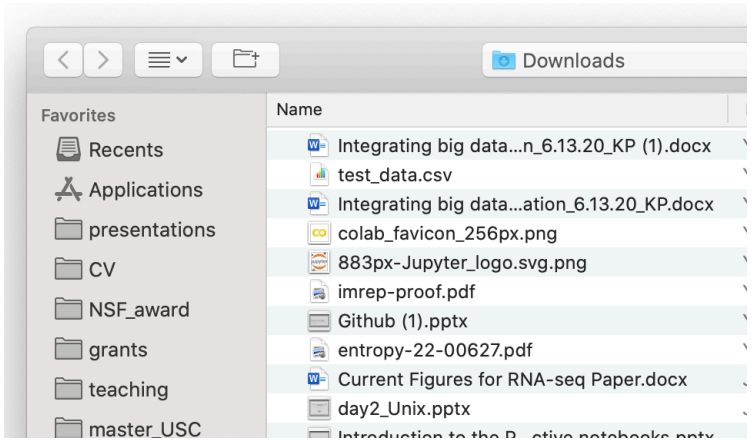
Save test CSV file to your Drive

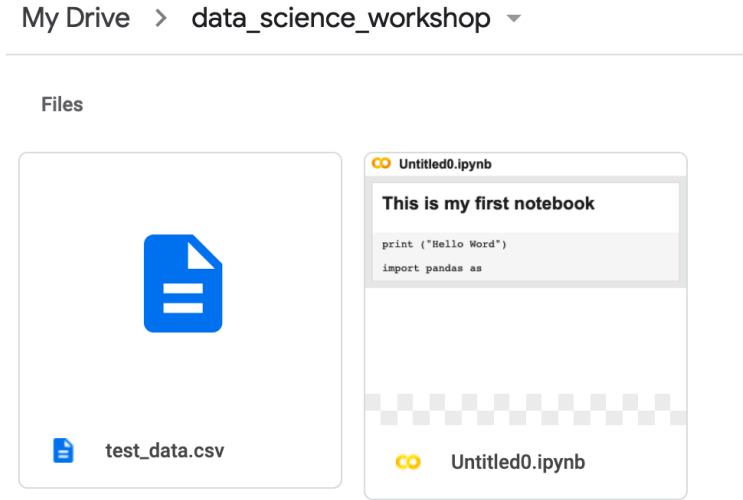
- Move all files from GitHub in a Google Drive folder named data_science_workshop
- We will use it later

Click



Click





File was successfully uploaded

CSV format

01

Often our data will be stored in CSV files

02

CSV files can be opened with MS Excel

- They also may be created from a spreadsheet
- To open with excel might need to use open with

03

Structure of a CSV file

- Essentially the same as a spreadsheet
- Every row is an entry
- Every column is a measurement (e.g. sex, age) (does not need to be numerical)
- Columns are separated by a comma (default)

More about CSV

01

Not necessarily visually friendly to a human

02

Do not use spaces between commas and values

- The computer will read this as part of the value
  - E.g. “control,_yes” turns into “control” and “_yes” in Python

03

Typically, the first column is the identifier (rowname)

- But it doesn’t need to be
- Sometimes the identifier isn’t obvious
- In our example should it be sample_name or column0?

Space

How to use a library in Python

In order to use a library it must be installed and imported (already done in Google Collab)

An import statement is just a piece of code stating what libraries will be used (e.g import pandas as pd)

“import as” means when you call functions from it later you can use a shorter name

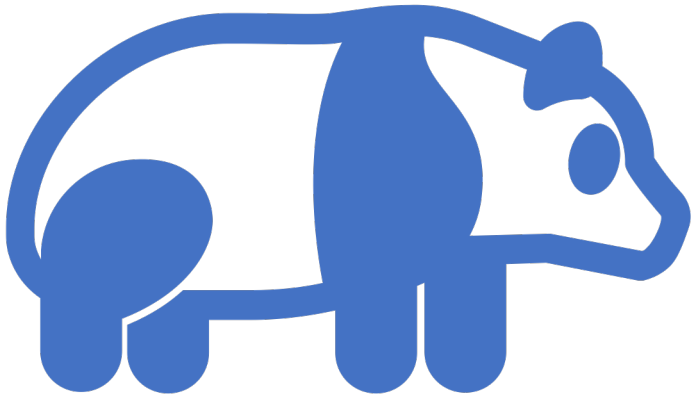
This must be in a block of code in your notebook

It must be run before any function from the library can be used

```
import pandas as pd
```

How to use pandas?

- It is already available in Google Collab
- import pandas as pd



Type

Autocomplete

Click

Where to store data for Google Colab

We will store files in Google Drive

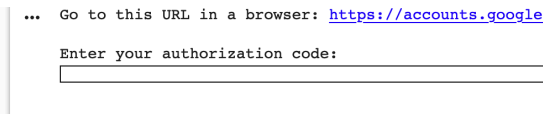
Any file saved in your “My Drive” can be accessed by using the following path: /content/drive/My Drive/<file>

Path to test.csv is “/content/drive/My Drive/data_science_workshop/test.csv

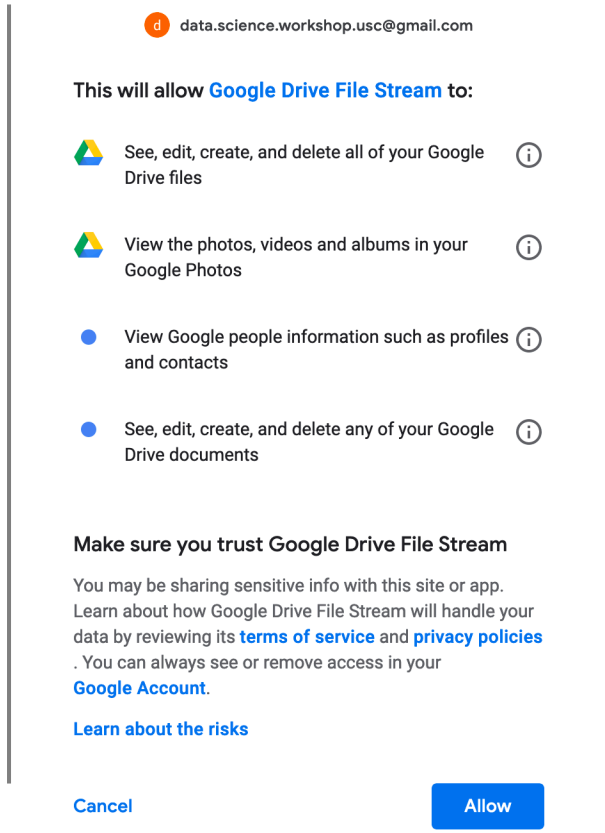
Access you Drive from notebook

Enter those two lines of code in the code block

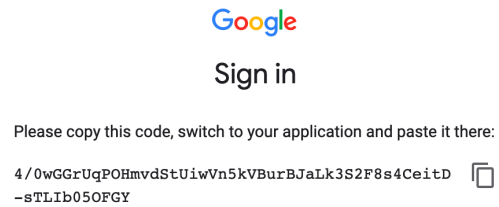
```
from google.colab import drive
drive.mount('/content/drive')
```



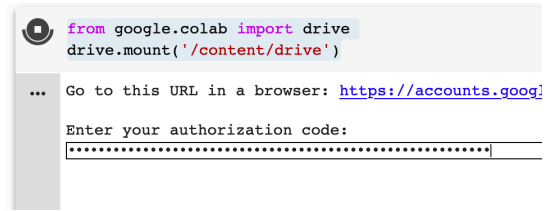
Click



Click



Click (it will copy)



Paste and press Enter/Return

Success

```
from google.colab import drive
drive.mount('/content/drive')
```

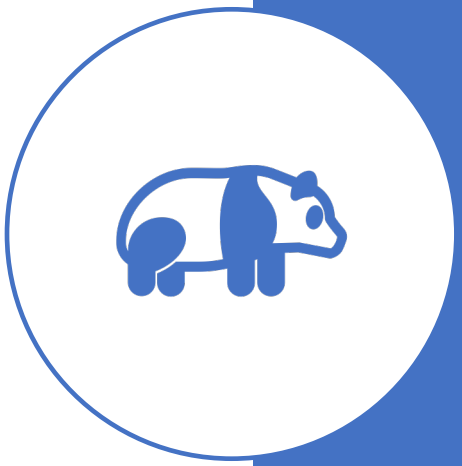
```
Go to this URL in a browser: https://accounts.goo

Enter your authorization code:
··········
Mounted at /content/drive
```

Success

Read CVS file using Pandas

pd.read_csv(<*filename*>)



Pandas library

```
import pandas as pd
```

Read test CVS file

```
df=pd.read_csv('/content/drive/My Drive/data_science_workshop/test_data.csv')
```

We create a new data frame add content from test CSV file

Was my file loaded?

Missing /

```
df=pd.read_csv('content/drive/My Drive/data_science_workshop/test_data.csv')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-12-4f7111cd9d58> in <module>()
----> 1 df=pd.read_csv('content/drive/My Drive/data_science_workshop/test_data.c

                                  4 frames
/usr/local/lib/python3.6/dist-packages/pandas/io/parsers.py in __init__(self, sr
   1889         kwds["usecols"] = self.usecols
   1890
-> 1891         self._reader = parsers.TextReader(src, **kwds)
   1892         self.unnamed_cols = self._reader.unnamed_cols
   1893

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source

FileNotFoundError: [Errno 2] File content/drive/My Drive/data_science_workshop/t
```

```
df=pd.read_csv('/content/drive/My Drive/data_science_workshop/test_data.csv')
```

No errors

Read other formats

PANDAS LIBRARY IS CAPABLE TO READ ANY FORMAT

YOU NEED TO SPECIFY HOW COLUMNS ARE DELIMITED

```
df=pd.read_csv('file_name', sep='\t')
```

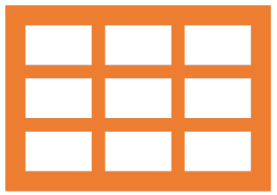
Special character for tab

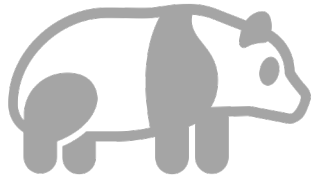
Understanding data frames

Columns

Rows

	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0





What does the dataframe look like?

Much like a spreadsheet

What makes it useful, or different?

The data structure is specifically built to make things easy to access

Pandas has many functions built to take advantage of the dataframe

Display data frame

Display the entire datadrame (for small files)

```
df
```

Display 5 (default) fist lines of

```
df.head()
```

Display a different number of lines

```
df.head(3)
```

Let's practice

```
df.head()
```

	column0	column1	column2	column3	column4	sample_name
0	control	yes	0.25	0.10	0.00	sample1
1	case	yes	0.90	0.25	0.75	sample2
2	control	yes	0.43	0.23	0.02	sample3
3	case	no	0.84	0.34	0.90	sample4

Names of rows

```
df.head()
```

	column0	column1	column2	column3	column4	sample_name
0	control	yes	0.25	0.10	0.00	sample1
1	case	yes	0.90	0.25	0.75	sample2
2	control	yes	0.43	0.23	0.02	sample3
3	case	no	0.84	0.34	0.90	sample4

An integer index starting from 0 is assigned to the data frame by default

Give names to rows

- We can explicitly define a column to be used as index by using index_col command

```
df=pd.read_csv('/content/drive/My Drive/data_science_workshop/test_data.csv',index_col='sample_name')
```

```
df.head()
```

	column0	column1	column2	column3	column4	sample_name
0	control	yes	0.25	0.10	0.00	sample1
1	case	yes	0.90	0.25	0.75	sample2
2	control	yes	0.43	0.23	0.02	sample3
3	case	no	0.84	0.34	0.90	sample4

```
df.head()
```

	column0	column1	column2	column3	column4
sample_name					
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02
sample4	case	no	0.84	0.34	0.90

use “sample_name” column as index

Access columns by name

Access column with name1

- df['name1’]

Access columns with name1 and name2

- df[['name1 ', 'name2']]

Let’s practice

- df['column2']
- df[['column2','column4']]

sample_name	column2	column4
sample1	0.25	0.00
sample2	0.90	0.75
sample3	0.43	0.02
sample4	0.84	0.90

Save selected columns to a new data frame

```
df1=df[['column2','column4']]
```

```
[43] df1=df[['column2','column4']]
     df1.head()
```

sample_name	column2	column4
sample1	0.25	0.00
sample2	0.90	0.75
sample3	0.43	0.02
sample4	0.84	0.90

Why creating a new columns

Add a descriptor column

Make a new column based on values of existing columns

Create a new column which is a sum of existing ones

```
df['column24'] = df['column2'] + df['column4']
```

```
df['column24'] = df['column2'] + df['column4']
df.head()
```

sample_name	column0	column1	column2	column3	column4	column24
sample1	control	yes	0.25	0.10	0.00	0.25
sample2	case	yes	0.90	0.25	0.75	1.65
sample3	control	yes	0.43	0.23	0.02	0.45
sample4	case	no	0.84	0.34	0.90	1.74

Concatenate columns

- Strings can be “summed” but only with other strings, this is concatenation

Let’s practice

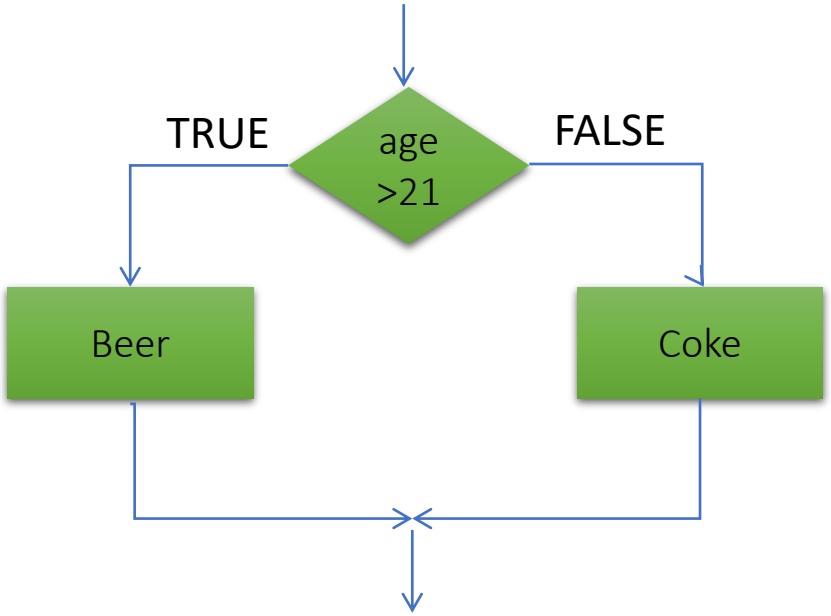
```
df['column01'] = df['column0'] + df['column1']
df.head()
```

sample_name	column0	column1	column2	column3	column4	new_column0	column24	column01
sample1	control	yes	0.25	0.10	0.00	10.0	controlyes	controlyes
sample2	case	yes	0.90	0.25	0.75	9.0	caseyes	caseyes
sample3	control	yes	0.43	0.23	0.02	10.0	controlyes	controlyes
sample4	case	no	0.84	0.34	0.90	9.0	caseno	caseno

```
df['column01'] = df['column0'] + "_"+df['column1']
df.head()
```

sample_name	column0	column1	column2	column3	column4	new_column0	column24	column01
sample1	control	yes	0.25	0.10	0.00	10.0	controlyes	control_yes
sample2	case	yes	0.90	0.25	0.75	9.0	caseyes	case_yes
sample3	control	yes	0.43	0.23	0.02	10.0	controlyes	control_yes
sample4	case	no	0.84	0.34	0.90	9.0	caseno	case_no

If statement



loc() function to add rows conditionally

- The **loc**() function is used to access a group of rows and columns

Double equal sign

```
df.loc[df['column0'] == 'control', 'new_column0'] = 10
```

Name of column

Value in column0

Name of new column

The value to be inserted in new column if the value in column0 is control

sample_name	column0	column1	column2	column3	column4	new_column0	column24	column01
sample1	control	yes	0.25	0.10	0.00	10.0	controlyes	controlyes
sample2	case	yes	0.90	0.25	0.75	9.0	caseyes	caseyes
sample3	control	yes	0.43	0.23	0.02	10.0	controlyes	controlyes
sample4	case	no	0.84	0.34	0.90	9.0	caseno	caseno

Let’s practice

```
df.loc[df['column0'] == 'control', 'new_column0'] = 10
df
```

sample_name	column0	column1	column2	column3	column4	new_column0
sample1	control	yes	0.25	0.10	0.00	10.0
sample2	case	yes	0.90	0.25	0.75	NaN
sample3	control	yes	0.43	0.23	0.02	10.0
sample4	case	no	0.84	0.34	0.90	NaN

Lack of rule when the value in column0 is “case”

Let’s practice

```
df.loc[df['column0'] == 'control', 'new_column0'] = 10
df.loc[df['column0'] == 'case', 'new_column0'] = 9

df
```

sample_name	column0	column1	column2	column3	column4	new_column0
sample1	control	yes	0.25	0.10	0.00	10.0
sample2	case	yes	0.90	0.25	0.75	9.0
sample3	control	yes	0.43	0.23	0.02	10.0
sample4	case	no	0.84	0.34	0.90	9.0

Selecting rows from data frame

Double equal sign

String

```
df.loc[df['column0'] == 'control']
```

this will select all rows with the value ‘control’ in column0

```
df.loc[df['column0'] == 'control']
```

sample_name	column0	column1	column2	column3	column4	new_column0	column24	column01
sample1	control	yes	0.25	0.10	0.00	10.0	controlyes	control_yes
sample3	control	yes	0.43	0.23	0.02	10.0	controlyes	control_yes

Selecting rows from data frame

Number

```
df.loc[df['column2'] >0.5]
```

```
df.loc[df['column2'] >0.5]
```

sample_name	column0	column1	column2	column3	column4	new_column0	column24	column01
sample2	case	yes	0.90	0.25	0.75	9.0	caseyes	case_yes
sample4	case	no	0.84	0.34	0.90	9.0	caseno	case_no

Data types

```
df.dtypes
```

```
df.dtypes

column0     object
column1     object
column2     object
column3    float64
column4    float64
dtype: object
```

Since one value is non-numerical, the whole column become non-numeric

Clean non-numerical values

Convert all non-numeric values to NaN

Remove rows with NaN in a specific column

Let’s practise

1. `df['column2'] = pd.to_numeric(df['column2'], errors='coerce')`

```
df['column2'] = pd.to_numeric(df['column2'], errors='coerce')
df
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02
sample4	case	no	NaN	0.34	0.90

Non-numerical values will be set as NaN

2. `df=df.dropna(subset=['column2'])`

```
df=df.dropna(subset=['column2'])
df
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02

More cleaning techniques

```
df=df.dropna()
```

More cleaning techniques

- Drop all rows with at least one NaN value
- Drop all rows entirely composed of NaN
- Fill NaN with a passed in value

Drop all rows entirely composed of NaN

```
df=df.dropna(how='all')
```

```
df=df.dropna(how='all')
df
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02
sample4	case	no	NaN	0.34	0.90

Drop all rows with at least one NaN value

```
df=df.dropna(how='any')
```

```
df=df.dropna(how='any')
df
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02

Replace NaN with a passed in value

Replace with 0

```
df.fillna(0)
```

```
df.fillna(0)
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02
sample4	case	no	0.00	0.34	0.90

NaN was replaced by 0

Merging data frames

Sometimes we have two data frames with related information (e.g. some other results for the same subjects)

Some subject can be only present in one of the data frames. We can choose to include them or not

Columns names with subjects should have the same name

Merge 2 data frames

	column0	column1	column2	column3	column4	sample_name
2	control	yes	0.25	0.1	0	sample1
3	case	yes	0.9	0.25	0.75	sample2
4	control	yes	0.43	0.23	0.02	sample3
5	case	no	c084	0.34	0.9	sample4

	age	sample_name
2	25	sample1
3	33	sample4
4	55	sample3

https://github.com/Mangul-Lab-USC/biomedical_data_science/blob/master/workshop_materials/

Test data in CSV format

- For this module we will use a very simple example CSV
  - https://github.com/Mangul-Lab-USC/biomedical_data_science/blob/master/module_1/test_data.csv

Click

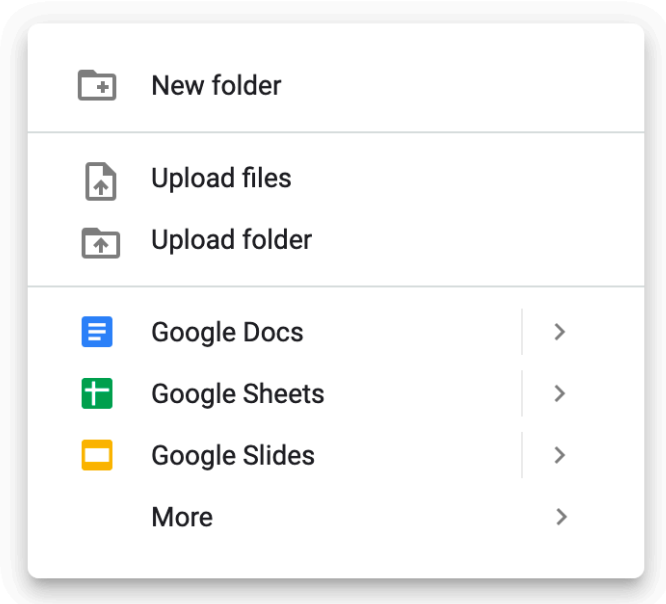
5 lines (5 sloc) | 178 Bytes

Raw | Blame | History

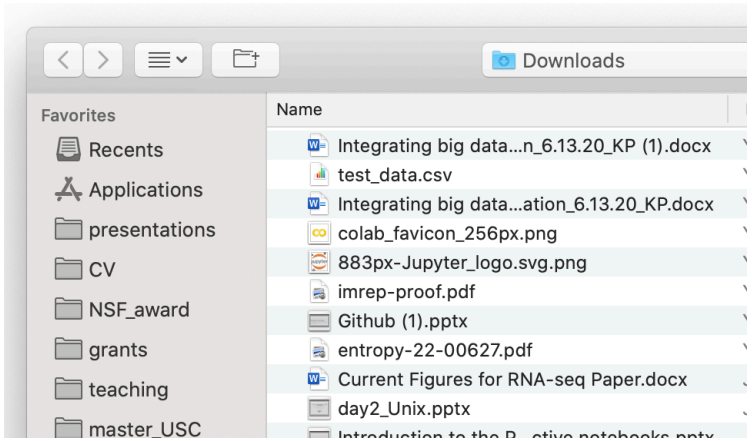
Search this file...

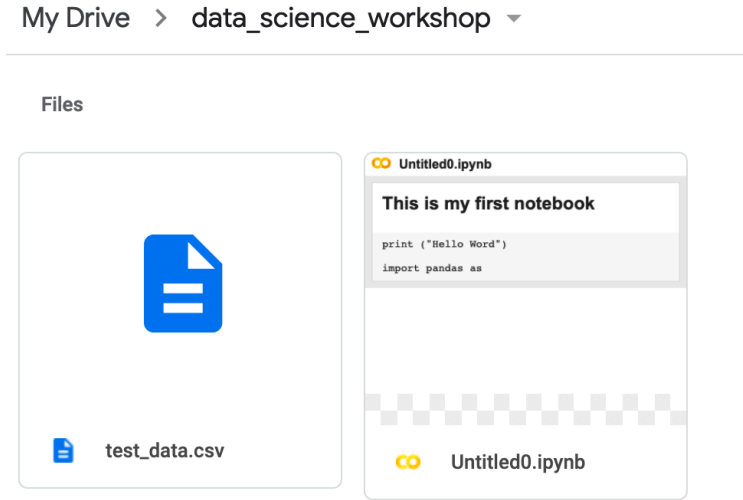
	column0	column1	column2	column3	column4	sample_name
2	control	yes	0.25	0.1	0	sample1
3	case	yes	0.9	0.25	0.75	sample2
4	control	yes	0.43	0.23	0.02	sample3
5	case	no	0.84	0.34	0.9	sample4

Click



Click





File was successfully uploaded

Read data frame #2

```
df2=pd.read_csv('/content/drive/My Drive/data_science_workshop/test_data2.csv',index_col='sample_name')
```

```
df2=pd.read_csv('/content/drive/My Drive/data_science_workshop/test_data2.csv',index_col='sample_name')
df.head()
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.90	0.25	0.75
sample3	control	yes	0.43	0.23	0.02
sample4	case	no	NaN	0.34	0.90

How to merge

- The merge function can be called on one data frame, with another as an argument to the function

New data frame to be created

Column name

```
df_full=df.merge(df2,on='sample_name')
```

```
df_full=df.merge(df2,on='sample_name')
df_full.head()
```

sample_name	column0	column1	column2	column3	column4	age
sample1	control	yes	0.25	0.10	0.00	25
sample3	control	yes	0.43	0.23	0.02	55
sample4	case	no	c084	0.34	0.90	33

Various ways of merging

inner	Take only row identifiers that appear in both data frames (the intersection)
left	Take only row identifiers that appear in the left data frame
right	Take only row identifiers that appear in the right data frame
outer	Take any row identifier that appears in either data frame (the union)

Take only row identifiers that appear in both data frames (default)

```
df_full=df.merge(df2,on='sample_name',how='inner')
```

```
df_full=df.merge(df2,on='sample_name',how='inner')
df_full.head()
```

sample_name	column0	column1	column2	column3	column4	age
sample1	control	yes	0.25	0.10	0.00	25
sample3	control	yes	0.43	0.23	0.02	55
sample4	case	no	c084	0.34	0.90	33

Take only row identifiers that appear in the left data frame

	column0	column1	column2	column3	column4	sample_name
1	control	yes	0.25	0.1	0	sample1
2	case	yes	0.9	0.25	0.75	sample2
3	control	yes	0.43	0.23	0.02	sample3
4	case	no	c084	0.34	0.9	sample4

left

	age	sample_name
1	25	sample1
2	33	sample4
3	55	sample3

right

```
df_full=df.merge(df2,on='sample_name',how='left')
```

```
df_full=df.merge(df2,on='sample_name',how='left')
df_full.head()
```

sample_name	column0	column1	column2	column3	column4	age
sample1	control	yes	0.25	0.10	0.00	25.0
sample2	case	yes	0.9	0.25	0.75	NaN
sample3	control	yes	0.43	0.23	0.02	55.0
sample4	case	no	c084	0.34	0.90	33.0

Take any row identifier that appears in either data frame (the union)

```
df_full=df.merge(df2,on='sample_name',how='outer')
```

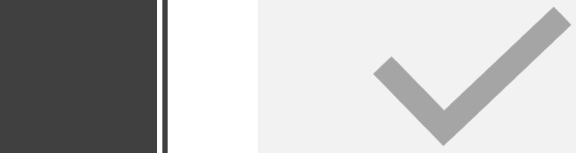
```
df_full=df.merge(df2,on='sample_name',how='outer')
df_full.head()
```

sample_name	column0	column1	column2	column3	column4	age
sample1	control	yes	0.25	0.10	0.00	25.0
sample2	case	yes	0.9	0.25	0.75	NaN
sample3	control	yes	0.43	0.23	0.02	55.0
sample4	case	no	c084	0.34	0.90	33.0

+ Co

Identical columns across data frames

We should have at least one identical column to be able to merge



What will happen if we have more than one identical column?

Let’s figure it out

	age	sample_name	column1
2	25	sample1	yes
3	33	sample4	no
4	55	sample3	yes

	column0	column1	column2	column3	column4	sample_name
2	control	yes	0.25	0.1	0	sample1
3	case	yes	0.9	0.25	0.75	sample2
4	control	yes	0.43	0.23	0.02	sample3
5	case	no	c084	0.34	0.9	sample4

```
df_full=df.merge(df3,on='sample_name)
```

Duplicated column

```
df_full=df.merge(df3,on='sample_name')
df_full.head()
```

sample_name	column0	column1_x	column2	column3	column4	age	column1_y
sample1	control	yes	0.25	0.10	0.00	25	yes
sample3	control	yes	0.43	0.23	0.02	55	yes
sample4	case	no	c084	0.34	0.90	33	no

Avoid duplicated columns

```
df_full=df.merge(df3,on=['sample_name','column1'])
```

List of all identical columns

```
df_full=df.merge(df3,on=['sample_name','column1'])
df_full.head()
```

sample_name	column0	column1	column2	column3	column4	age
sample1	control	yes	0.25	0.10	0.00	25
sample3	control	yes	0.43	0.23	0.02	55
sample4	case	no	c084	0.34	0.90	33

Resetting your index

```
df.reset_index()

df.set_index('sample_name'
```

- If you ever find that you'd like your index to be a column

```
df.head()
```

sample_name	column0	column1	column2	column3	column4
sample1	control	yes	0.25	0.10	0.00
sample2	case	yes	0.9	0.25	0.75
sample3	control	yes	0.43	0.23	0.02
sample4	case	no	c084	0.34	0.90

```
df=df.reset_index()
df.head()
```

	sample_name	column0	column1	column2	column3	column4
0	sample1	control	yes	0.25	0.10	0.00
1	sample2	case	yes	0.9	0.25	0.75
2	sample3	control	yes	0.43	0.23	0.02
3	sample4	case	no	c084	0.34	0.90

Grouping rows by categories

Often we want to group rows together for group calculations (e.g. mean for each group)

Grouping by a numeric variable requires creating a categorical column using conditionals

Creating a categorical column using conditionals

```
df_full.loc[df_full['age'] >25, 'age_cat'] = 'gt10'
```

Condition

New categorical columns

```
df_full.loc[df_full['age'] <=25, 'age_cat'] = 'lt10'
```

```
df_full.loc[df_full['age'] >25, 'age_cat'] = 'gt10'
df_full.loc[df_full['age'] <=25, 'age_cat'] = 'lt10'

df_full
```

sample_name	column0	column1	column2	column3	column4	age	age_cat
sample1	control	yes	0.25	0.10	0.00	25	lt10
sample3	control	yes	0.43	0.23	0.02	55	gt10
sample4	case	no	c084	0.34	0.90	33	gt10

Grouping rows based on a single column

```
df_full.groupby('column0').mean()
```

```
new_group=df_full.groupby('column0').mean()
new_group.head()
```

column0	column3	column4	age
case	0.340	0.90	33
control	0.165	0.01	40

Grouping rows based on a multiple columns

```
new_group=df_full.groupby(['column0','age_cat']).mean()
```

```
new_group=df_full.groupby(['column0','age_cat']).mean()
new_group.head()
```

		column3	column4	age
column0	age_cat			
case	gt10	0.34	0.90	33
control	gt10	0.23	0.02	55
	lt10	0.10	0.00	25

Writing to CSV files

- Finally, after all of our transformations, we probably want to write our new data frame to a new CSV file
- This is as simple as reading a CSV

```
df_full.to_csv('/content/drive/My Drive/data_science_workshop/test_data_full.csv')
```

Save groupby results into CSV

New data frame

```
df_group=df_full.groupby('column0').mean()
```

```
df_group.to_csv('/content/drive/My Drive/data_science_workshop/test_data_group.csv')
```