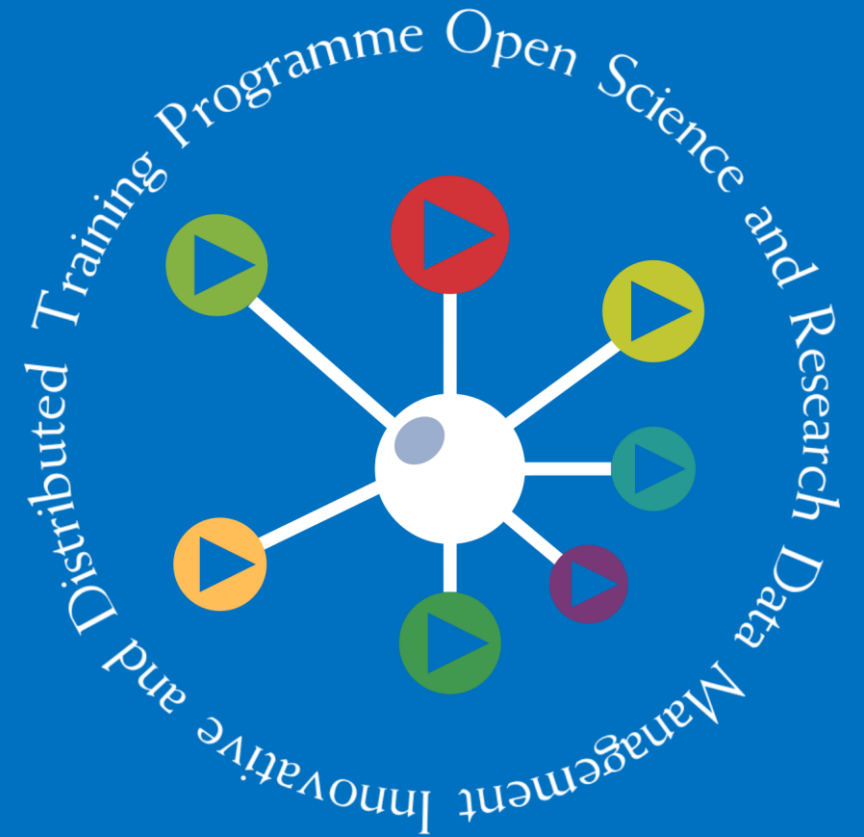


Reproducible Science: exercise

This module is part of the training session “Train for trainers” within project TrainRDM



National Institute of Research and Development in Informatics (ICI Bucharest)
University POLITEHNICA of Bucharest

This work is licensed under the Creative Commons Attribution 4.0 International License.
All images are public domain unless otherwise noted.

e-Science and the Fourth Paradigm






Thousand years ago – Experimental Science
Description of natural phenomena

Last few hundred years – Theoretical Science
Newton's Laws, Maxwell's Equations...

Last few decades – Computational Science - Simulation of complex phenomena

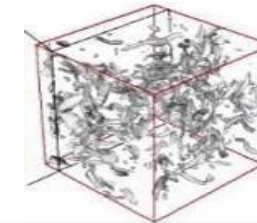
Today – Data-Intensive Science

Scientists overwhelmed with data sets
from many different sources

-  Data captured by instruments
-  Data generated by simulations
-  Data generated by sensor networks



$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi G\rho}{3} - K\frac{c^2}{a^2}$$



e-Science is the set of tools and technologies to support data federation and collaboration

- For analysis and data mining
- For data visualization and exploration
- For scholarly communication and dissemination

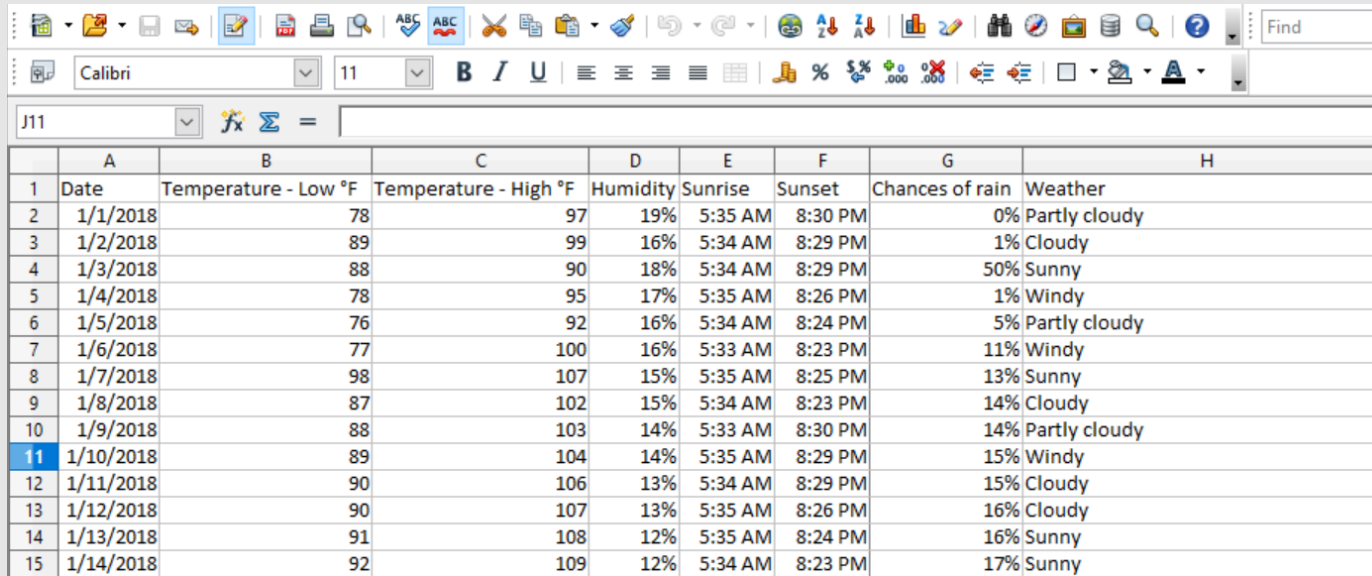
(With thanks to Jim Gray)

Data Science - introduction

Part I





Data Science terms



The screenshot shows a spreadsheet application window with a toolbar and a data table. The table has columns for Date, Temperature - Low °F, Temperature - High °F, Humidity, Sunrise, Sunset, Chances of rain, and Weather. The data covers the dates from 1/1/2018 to 1/14/2018.

	A	B	C	D	E	F	G	H
1	Date	Temperature - Low °F	Temperature - High °F	Humidity	Sunrise	Sunset	Chances of rain	Weather
2	1/1/2018	78	97	19%	5:35 AM	8:30 PM	0%	Partly cloudy
3	1/2/2018	89	99	16%	5:34 AM	8:29 PM	1%	Cloudy
4	1/3/2018	88	90	18%	5:34 AM	8:29 PM	50%	Sunny
5	1/4/2018	78	95	17%	5:35 AM	8:26 PM	1%	Windy
6	1/5/2018	76	92	16%	5:34 AM	8:24 PM	5%	Partly cloudy
7	1/6/2018	77	100	16%	5:33 AM	8:23 PM	11%	Windy
8	1/7/2018	98	107	15%	5:35 AM	8:25 PM	13%	Sunny
9	1/8/2018	87	102	15%	5:34 AM	8:23 PM	14%	Cloudy
10	1/9/2018	88	103	14%	5:33 AM	8:30 PM	14%	Partly cloudy
11	1/10/2018	89	104	14%	5:35 AM	8:29 PM	15%	Windy
12	1/11/2018	90	106	13%	5:34 AM	8:29 PM	15%	Cloudy
13	1/12/2018	90	107	13%	5:35 AM	8:26 PM	16%	Cloudy
14	1/13/2018	91	108	12%	5:35 AM	8:24 PM	16%	Sunny
15	1/14/2018	92	109	12%	5:34 AM	8:23 PM	17%	Sunny

 A **dataset** is a table/spreadsheet document with historical information

 If you want to understand details about the weather, the dataset will include the historical information about the weather in the past few years. Precipitation, Humidity, Temperature, etc.

 Sample snapshot of the dataset

Data Science terms



✿ A **variable** is like a container that can be assigned a value.

✿ Precipitation, Humidity, temperature etc are variables in the previous example.

✿ **Types of variables:**

✿ **Independent variables** are not influenced by any factors and impact the outcome variable. The outcome variable is 'chances of rain'. Temperature, Humidity, sunrise, sunset times are independent variables.

✿ **Dependent variables** are influenced or dependent on the independent variable. For example – the chance of rain depends on the temperature, humidity etc. Depending on the temperature, the chances of rain may increase or decrease.

✿ Any variable with numerical value is considered as **continuous**. Temperature, humidity, and chances of rain are considered continuous variables.

✿ Any variable with non-numerical value is considered as **categorical**. Weather is considered a categorical variable.

✿ **Descriptive data:** Summarize the data or describe the data in a meaningful way to provide insights about the dataset. Below are some of the key insights that can be provided to describe the data:

✿ Highest and lowest temperature

✿ Did increase in the temperature lower the chances of rain?

✿ Effects of humidity on rain

✿ Weather Vs. chances of rain

Data Science Project



1. Importing the Dataset

🌟 We will learn how to import the data set and load it onto the Jupyter notebook

2. Reviewing the Dataset

🌟 We can review the dataset by looking at the first few rows of data. You can do that by using the head function in python. It helps provide an idea about the data structure.

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

```
import pandas as pd
df=pd.read_csv (r'C:\Users\CA34789\Desktop\Personal\Blog\Data Science Learning content\module 2.csv', encoding='latin-1')
df.head()
```

Jupyter Notebooks

Part II



*This tutorial is based on <https://www.dataquest.io/blog/jupyter-notebook-tutorial/>

Jupyter Notebook



🌟 A **notebook** integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media

🌟 It is a single document where you can run code, display the output, and also add explanations, formulas, charts, and make your work more transparent, understandable, repeatable, and shareable.

🌟 Using Notebooks is now a major part of the **data science workflow** at companies across the globe

🌟 A Notebook will speed up your workflow and make it easier to communicate and share your results.

🌟 Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects

🌟 As part of the open source Project Jupyter, Jupyter Notebooks are completely free.

🌟 You can download the software on its own, or as part of the Anaconda data science toolkit.

Installing Jupyter Notebook

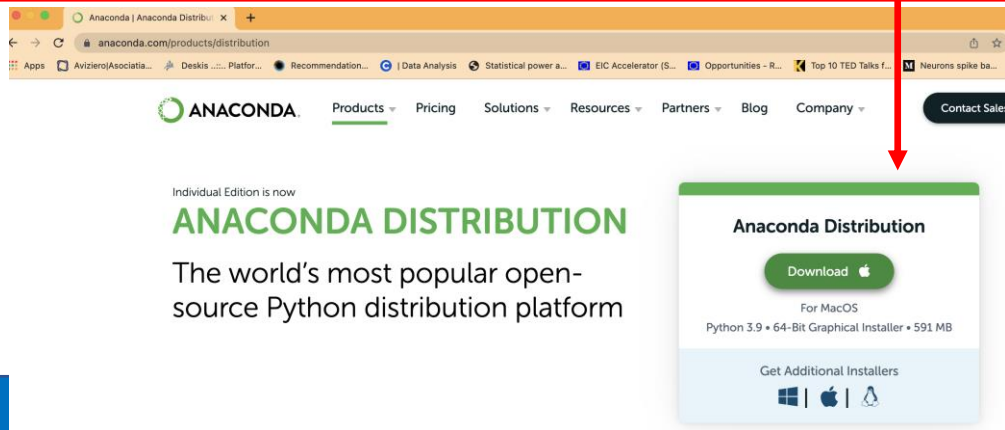
- ✿ The easiest way for a beginner to get started with Jupyter Notebooks is by installing [Anaconda](#).
 - ✿ Anaconda is the most widely used Python distribution for data science and comes pre-loaded with all the most popular libraries and tools.
 - ✿ Some of the biggest Python libraries included in Anaconda include [NumPy](#), [pandas](#), and [Matplotlib](#), though the [full 1000+ list](#) is exhaustive.

To get Anaconda, simply:

- [Download](#) the latest version of Anaconda for Python 3.8.
- Install Anaconda by following the instructions on the download page and/or in the executable.

If you are a more advanced user with Python already installed and prefer to manage your packages manually, you can just [use pip](#):

pip3 install jupyter



```
Last login: Sat May 7 08:20:31 on console

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Ciprians-MBP:~ cipriandobres$ pip3 install jupyter
Requirement already satisfied: jupyter in ./opt/anaconda3/lib/python3.9/site-packages (1.0.0)
Requirement already satisfied: ipykernel in ./opt/anaconda3/lib/python3.9/site-packages (from jupyter) (6.4.1)
Requirement already satisfied: nbconvert in ./opt/anaconda3/lib/python3.9/site-packages (from jupyter) (6.1.0)
Requirement already satisfied: notebook in ./opt/anaconda3/lib/python3.9/site-packages (from jupyter) (6.4.5)
Requirement already satisfied: jupyter-console in ./opt/anaconda3/lib/python3.9/site-packages (from jupyter) (6.4.0)
Requirement already satisfied: ipywidgets in ./opt/anaconda3/lib/python3.9/site-packages (from jupyter) (7.6.5)
```

Exercise...

Running Jupyter



- One can run Jupyter via the shortcut Anaconda adds to your start menu, or via `'jupyter notebook'`, which will open a new tab in your default web browser:

<input type="checkbox"/>	0		Name ↓	Last Modified
<input type="checkbox"/>		3D Objects		11 days ago
<input type="checkbox"/>		Contacts		11 days ago
<input type="checkbox"/>		Desktop		11 days ago
<input type="checkbox"/>		Documents		5 days ago
<input type="checkbox"/>		Downloads		2 days ago
<input type="checkbox"/>		Favorites		11 days ago

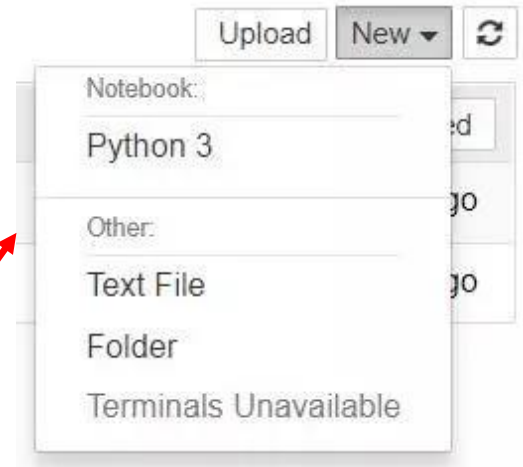
- This isn't a notebook just yet, but don't panic! There's not much to it. This is the **Notebook Dashboard**, specifically designed for managing your Jupyter Notebooks. Think of it as the launchpad for exploring, editing and creating your notebooks.

Jupyter Notebook

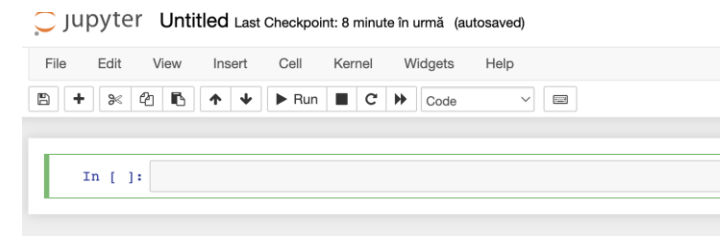


Jupyter's Notebooks and dashboard are web apps, and Jupyter starts up a local Python server to serve these apps to your web browser, making it essentially platform-independent and opening the door to easier sharing on the web.

Browse to the folder in which you would like to create your first notebook, click the “New” drop-down button in the top-right and select “Python 3”:



Your first Jupyter Notebook will open in new tab — each notebook uses its own tab because you can open multiple notebooks simultaneously.



If you switch back to the dashboard, you will see the new file Untitled.ipynb and you should see some green text that tells you your notebook is running.



Exercise...

ipynb File

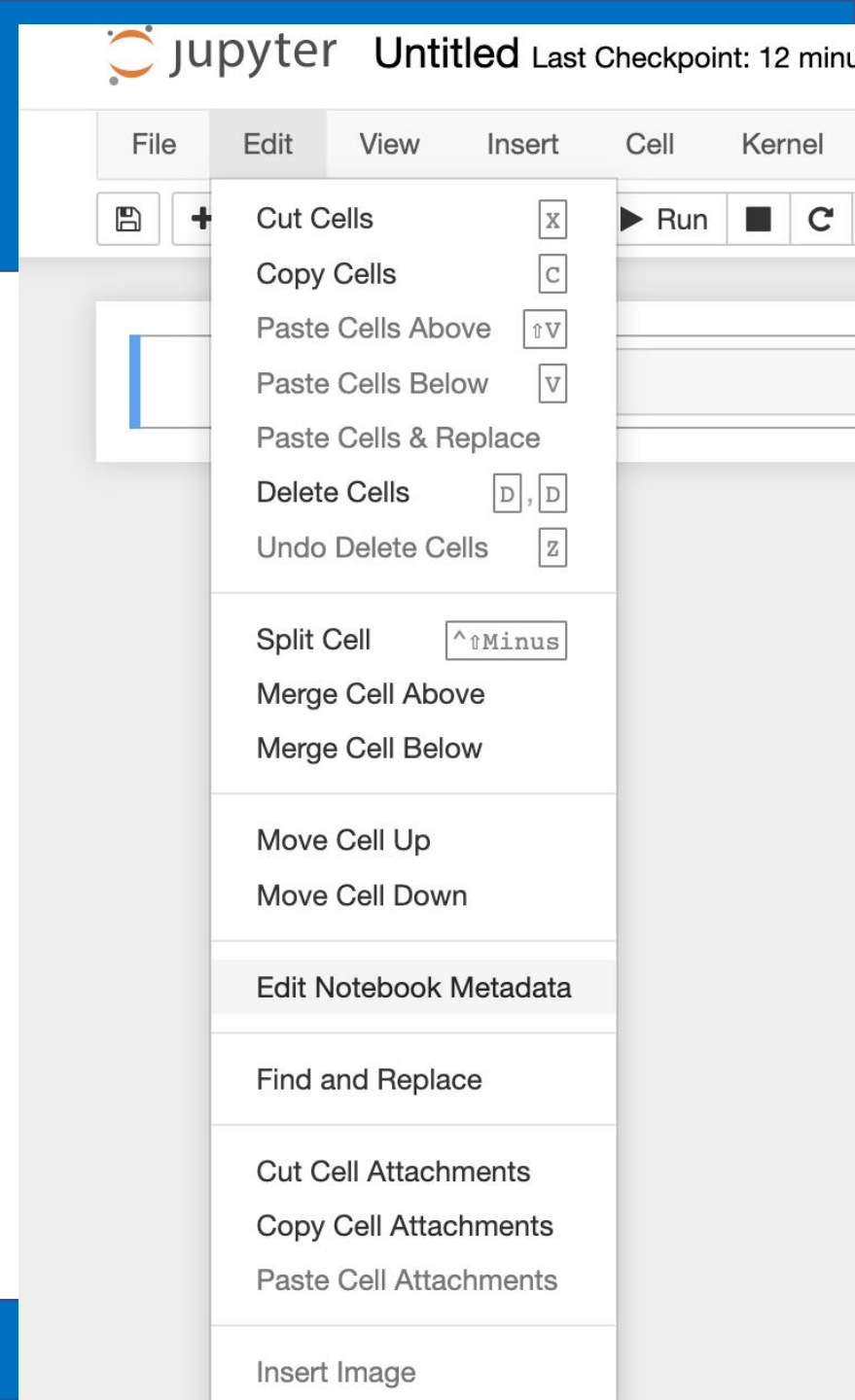
Each .ipynb file is one notebook

Each time you create a new notebook, a new .ipynb file will be created.

An .ipynb file is a text file that describes the contents of your notebook in a format called JSON. Each cell and its contents, including image attachments that have been converted into strings of text, is listed therein along with metadata.

You can edit this yourself — if you know what you are doing! — by selecting “Edit > Edit Notebook Metadata” from the menu bar in the notebook. You can also view the contents of your notebook files by selecting “Edit” from the controls on the dashboard

However, the key word there is *can*. In most cases, there’s no reason you should ever need to edit your notebook metadata manually.



Notebook terms

There are two Notebook terms that we work with: *cells* and *kernels*

A **kernel** is a “computational engine” that executes the code contained in a notebook document.

A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook’s kernel.

Cells



- ❖ Cells form the body of a notebook.
- ❖ There are two main cell types :
- ❖ A **code cell** contains code to be executed in the kernel.
 - ❖ When the code is run, the notebook displays the output below the code cell that generated it.
 - ❖ The first cell in a new notebook is always a code cell.
- ❖ A **Markdown cell** contains text formatted using Markdown and displays its output in-place when the Markdown cell is run.
- ❖ Let's test it out with a classic hello world example:
- ❖ Type `print('Hello World!')` into the cell and click the run button Notebook Run Button in the toolbar above or press Ctrl (or CMD) + Enter.

```
In [1]: print('Hello World!')  
Hello World!
```

Cells



Jupyter Untitled Last Checkpoint: 8 minute în urmă (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In []:

```
In [1]: print('Hello World!')
```

```
Hello World!
```

🌟 Notice the difference? When we run the cell, its output is displayed below and the label to its left will have changed from In [] to In [1]

🌟 The “In” part of the label is simply short for “Input,” while the label number indicates *when* the cell was executed on the kernel — in this case the cell was executed first.

🌟 Run the cell again and the label will change to In [2] because now the cell was the second to be run on the kernel.

🌟 The output of a code cell also forms part of the document

🌟 You can always tell the difference between code and Markdown cells because code cells have that label on the left and Markdown cells do not.



Cells




- From the menu bar, click *Insert* and select *Insert Cell Below* to create a new code cell underneath your first and try out the following code to see what happens. Do you notice anything different?

```
In [1]: print('Hello World!')
Hello World!

In [3]: import time
time.sleep(3)

In [ ]:
```

- This cell doesn't produce any output, but it does take three seconds to execute. Notice how Jupyter signifies when the cell is currently running by changing its label to `In [*]`.

In general, the output of a cell comes from any text data specifically printed during the cell's execution, as well as the value of the last line in the cell, be it a lone variable, a function call, or something else. 

```
In [1]: print('Hello World!')
Hello World!

In [3]: import time
time.sleep(3)

In [4]: def say_hello(recipient):
         return 'Hello, {}'.format(recipient)
say_hello('Tim')

Out[4]: 'Hello, Tim!'

In [ ]:
```


Keyboard Shortcuts



- ✿ One thing you may have observed when running your cells is that their border turns blue, whereas it was green while you were editing.
 - ✿ In a Jupyter Notebook, there is always one “active” cell highlighted with a border whose color denotes its current mode:
 - ✿ **Green outline** — cell is in “edit mode”
 - ✿ **Blue outline** — cell is in “command mode”
- ✿ Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions you can carry out on the active cell when it’s in command mode.
 - ✿ Scroll up and down your cells with your Up and Down keys.
 - ✿ Press A or B to insert a new cell above or below the active cell.
 - ✿ M will transform the active cell to a Markdown cell.
 - ✿ Y will set the active cell to a code cell.
 - ✿ D + D (D twice) will delete the active cell.
 - ✿ Z will undo cell deletion.
 - ✿ Hold Shift and press Up or Down to select multiple cells at once. With multiple cells selected, Shift + M will merge your selection.
- ✿ Ctrl + Shift + -, in edit mode, will split the active cell at the cursor.
- ✿ You can also click and Shift + Click in the margin to the left of your cells to select them.

Markdown



- ✿ Markdown is a lightweight, easy to learn markup language for formatting plain text
- ✿ Its syntax has a one-to-one correspondence with HTML tags, so some prior knowledge here would be helpful but is definitely not a prerequisite
- ✿ Let's cover the basics with a quick example:

```
# This is a level 1 heading
## This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via bold and bold, or italic and italic.

Paragraphs must be separated by an empty line.

* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: foo(), and code blocks use triple backticks:
```
bar()
```

Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

This is a level 1 heading

This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via **bold** and bold, or *italic* and italic.

Paragraphs must be separated by an empty line.

- Sometimes we want to include lists.
- Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: foo(), and code blocks use triple backticks:

```
bar()
```

Or can be indented by 4 spaces:

```
    foo()
```

And finally, adding images is easy:

 Alt text

Kernels



- ✿ Behind every notebook runs a kernel
 - ✿ When you run a code cell, that code is executed within the kernel.
 - ✿ Any output is returned back to the cell to be displayed.
 - ✿ The kernel's state **persists** over time and between cells — it pertains to the document as a whole and not individual cells.
- ✿ For example, if you import libraries or declare variables in one cell, they will be available in another. Let's try this out to get a feel for it:

```
In [8]: import numpy as np
def square(x):
    return x * x
```

```
In [9]: x = np.random.randint(1, 10)
y = square(x)
print('%d squared is %d' % (x, y))

6 squared is 36
```

Choosing a Kernel



- 🌟 You may have noticed that Jupyter gives you the option to change kernel, and in fact there are many different options to choose from.
 - 🌟 Back when you created a new notebook from the dashboard by selecting a Python version, you were actually choosing which kernel to use.
- 🌟 There kernels for different versions of Python, and also for over 100 languages including Java, C, and even Fortran.
 - 🌟 Data scientists may be particularly interested in the kernels for R and Julia, as well as both imatlab and the Calysto MATLAB Kernel for Matlab.
- 🌟 The SoS kernel provides multi-language support within a single notebook.

Let's dissect an example

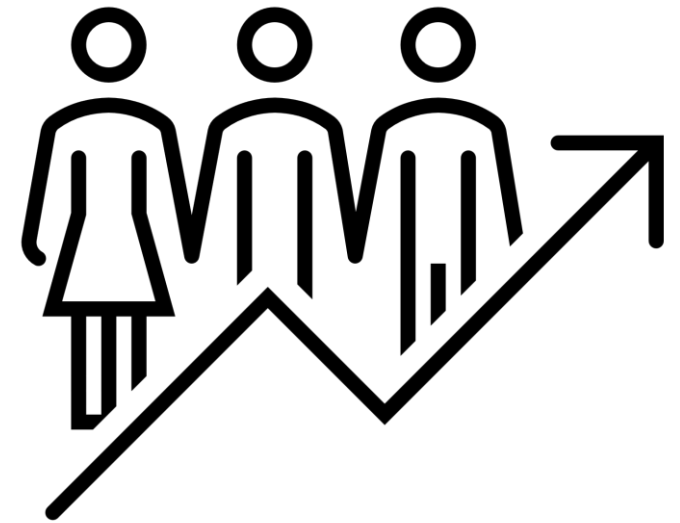
Part III



*This tutorial is based on <https://www.dataquest.io/blog/jupyter-notebook-tutorial/>

A little example

- 🌟 It is time to look at *how* Jupyter Notebooks are used in practice
- 🌟 We start with the Fortune 500 dataset that you received. Our goal is to find out **how the profits of the largest companies in the US changed historically.**



Setup



It's common to start off with a code cell specifically for imports and setup, so that if you choose to add or change anything, you can simply edit and re-run the cell without causing any side-effects.

```
In [ ]: %matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(style="darkgrid")
```

work with data

plot charts

visual charts

```
In [ ]: df = pd.read_csv('fortune500.csv')
```

read data

Work with data



🌟 We've loaded our data set *df* into the most-used pandas data structure, which is called a DataFrame and basically looks like a table:

```
In [4]: %matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="darkgrid")
```

```
Matplotlib is building the font cache; this may take a moment.
Fontconfig warning: ignoring UTF-8: not a valid region tag
```

```
In [5]: df = pd.read_csv('fortune500.csv')
```

```
In [6]: df.head()
```

Out[6]:

	Year	Rank	Company	Revenue (in millions)	Profit (in millions)
0	1955	1	General Motors	9823.5	806
1	1955	2	Exxon Mobil	5661.4	584.8
2	1955	3	U.S. Steel	3250.4	195.4
3	1955	4	General Electric	2959.1	212.6
4	1955	5	Esmark	2510.8	19.1

```
In [7]: df.tail()
```

Out[7]:

	Year	Rank	Company	Revenue (in millions)	Profit (in millions)
25495	2005	496	Wm. Wrigley Jr.	3648.6	493
25496	2005	497	Peabody Energy	3631.6	175.4
25497	2005	498	Wendy's International	3630.4	57.8
25498	2005	499	Kindred Healthcare	3616.6	70.6
25499	2005	500	Cincinnati Financial	3614.0	584

```
In [8]: df.columns = ['year', 'rank', 'company', 'revenue', 'profit']
df.tail()
```

Out[8]:

	year	rank	company	revenue	profit
25495	2005	496	Wm. Wrigley Jr.	3648.6	493
25496	2005	497	Peabody Energy	3631.6	175.4
25497	2005	498	Wendy's International	3630.4	57.8
25498	2005	499	Kindred Healthcare	3616.6	70.6
25499	2005	500	Cincinnati Financial	3614.0	584

Further exploring the data



```
In [9]: len(df)
```

```
Out[9]: 25500
```

```
In [10]: df.dtypes
```

```
Out[10]: year          int64  
rank            int64  
company         object  
revenue         float64  
profit          object  
dtype: object
```

← Hmm, not float64?

```
In [11]: non_numeric_profits = df.profit.str.contains('[^0-9.-]')  
df.loc[non_numeric_profits].head()
```

```
Out[11]:
```

	year	rank	company	revenue	profit
228	1955	229	Norton	135.0	N.A.
290	1955	291	Schlitz Brewing	100.0	N.A.
294	1955	295	Pacific Vegetable Oil	97.9	N.A.
296	1955	297	Liebmann Breweries	96.0	N.A.
352	1955	353	Minneapolis-Moline	77.4	N.A.

Just as we suspected! Some of the values are strings, which have been used to indicate missing data. Are there any other values that have crept in?

Further exploring the data



```
In [12]: set(df.profit[non_numeric_profits])
```

```
Out[12]: {'N.A.'}
```

That makes it easy to interpret, but what should we do?

```
In [13]: len(df.profit[non_numeric_profits])
```

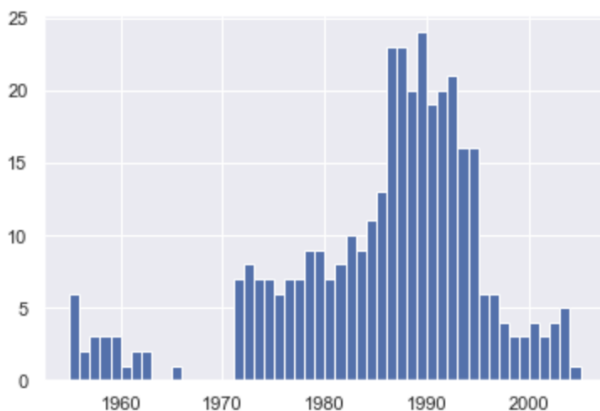
```
Out[13]: 369
```

click to scroll output; double click

It's a small fraction of our data set, though not completely inconsequential as it is still around 1.5%.

If rows containing N.A. are, roughly, uniformly distributed over the years, the easiest solution would just be to remove them. So let's have a quick look at the distribution.

```
In [14]: bin_sizes, _, _ = plt.hist(df.year[non_numeric_profits], bins=range(1955, 2006))
```



click to scroll output; double click to hide

At a glance, we can see that the most invalid values in a single year is fewer than 25, and as there are 500 data points per year, removing these values would account for less than 4% of the data for the worst years. Indeed, other than a surge around the 90s, most years have fewer than half the missing values of the peak.

```
In [15]: df = df.loc[~non_numeric_profits]
df.profit = df.profit.apply(pd.to_numeric)
```

let's say this is acceptable and go ahead and remove these rows

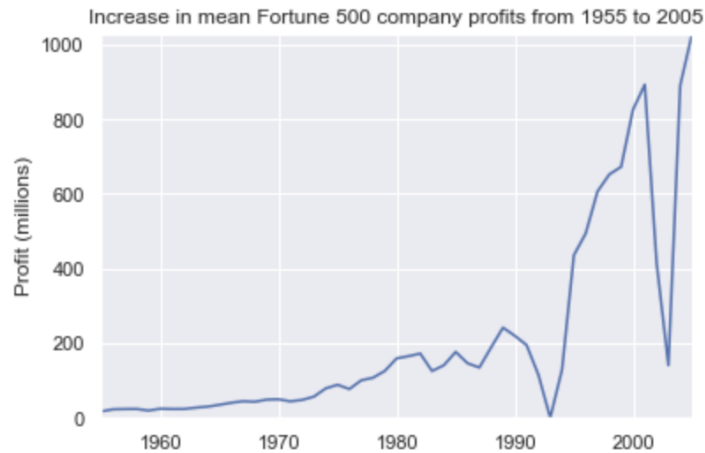
```
In [16]: len(df)
```

```
Out[16]: 25131
```

```
In [17]: group_by_year = df.loc[:, ['year', 'revenue', 'profit']].groupby('year')
avgs = group_by_year.mean()
x = avgs.index
y1 = avgs.profit
def plot(x, y, ax, title, y_label):
    ax.set_title(title)
    ax.set_ylabel(y_label)
    ax.plot(x, y)
    ax.margins(x=0, y=0)
```

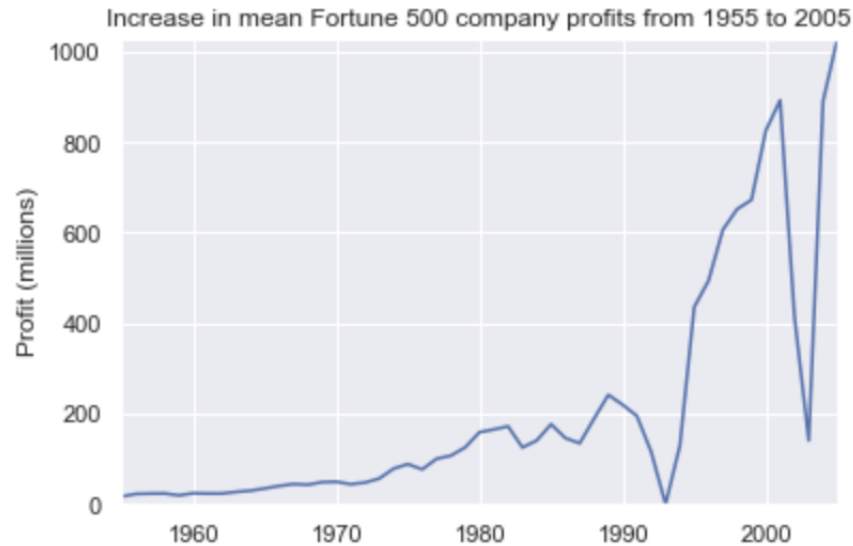
Next, we address our goal and plot average profit by year. We might as well plot the revenue as well, so first we can define some variables and a method to reduce our code.

```
In [18]: fig, ax = plt.subplots()
plot(x, y1, ax, 'Increase in mean Fortune 500 company profits from 1955 to 2005', 'Profit (millions)')
```



let's plot!

Looking at the result



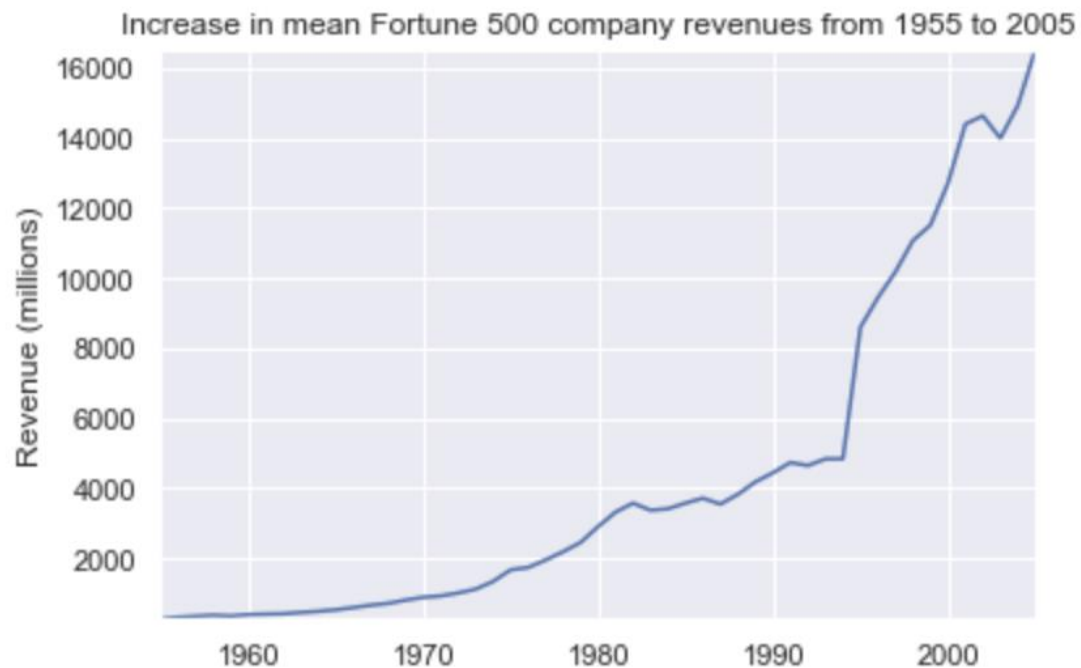
Wow, that looks like an exponential, but it's got some huge dips. They must correspond to the early 1990s recession and the dot-com bubble. It's pretty interesting to see that in the data. But how come profits recovered to even higher levels post each recession?

Maybe the revenues can tell us more...

Further analysis



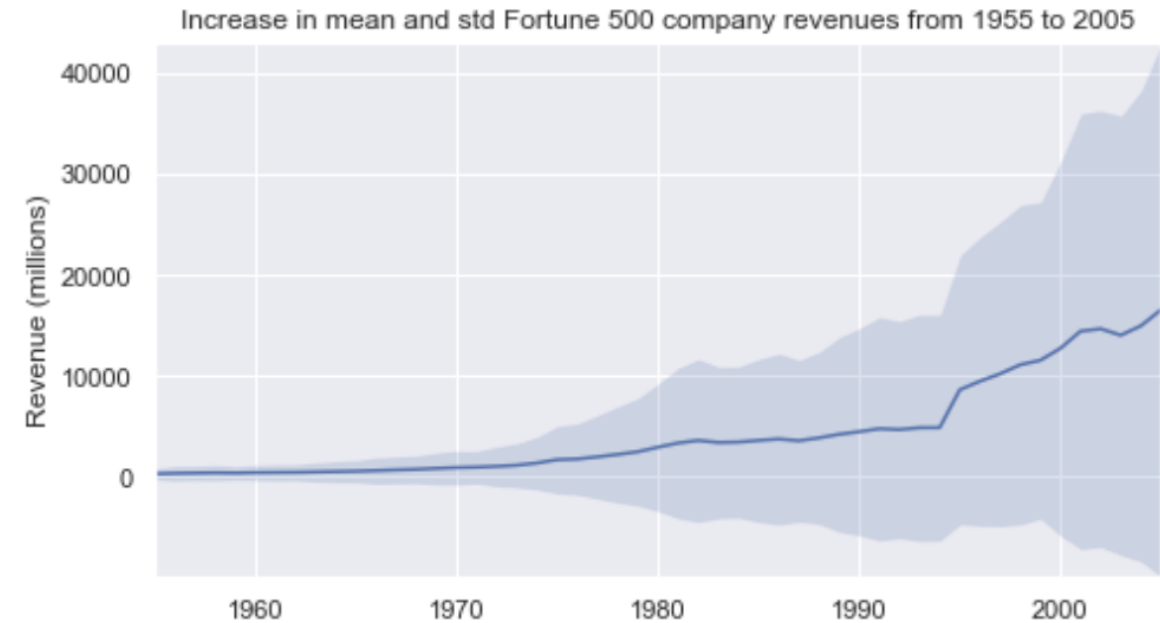
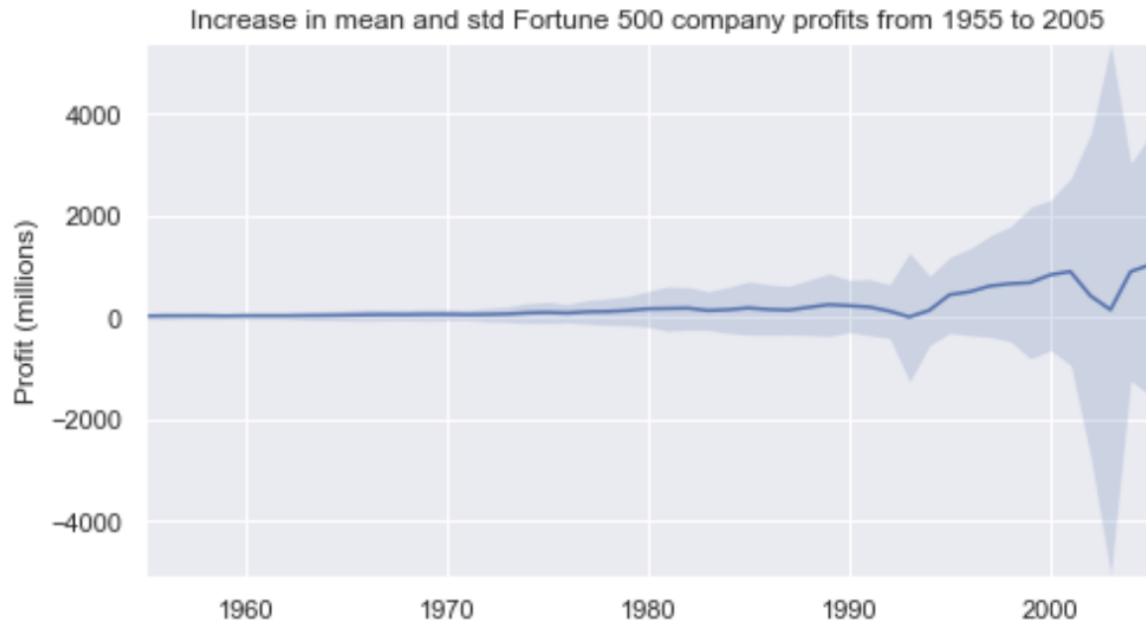
```
In [19]: y2 = avgs.revenue  
fig, ax = plt.subplots()  
plot(x, y2, ax, 'Increase in mean Fortune 500 company revenues from 1955 to 2005', 'Revenue (millions)')
```



That adds another side to the story. Revenues were not as badly hit — that's some great accounting work from the finance departments.

Let's superimpose these plots with +/- their standard deviations...

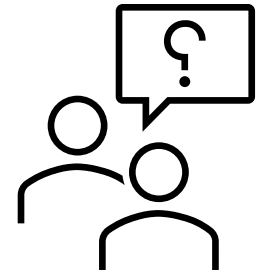
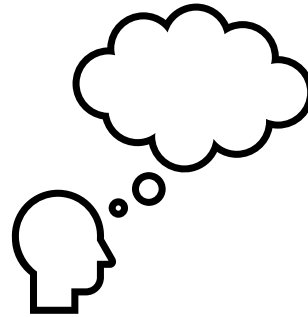
```
In [20]: def plot_with_std(x, y, stds, ax, title, y_label):
    ax.fill_between(x, y - stds, y + stds, alpha=0.2)
    plot(x, y, ax, title, y_label)
fig, (ax1, ax2) = plt.subplots(ncols=2)
title = 'Increase in mean and std Fortune 500 company %s from 1955 to 2005'
stds1 = group_by_year.std().profit.values
stds2 = group_by_year.std().revenue.values
plot_with_std(x, y1.values, stds1, ax1, title % 'profits', 'Profit (millions)')
plot_with_std(x, y2.values, stds2, ax2, title % 'revenues', 'Revenue (millions)')
fig.set_size_inches(14, 4)
fig.tight_layout()
```



Conclusion



- ✿ The standard deviations are huge! Some Fortune 500 companies make billions while others lose billions, and the risk has increased along with rising profits over the years.
- ✿ Perhaps some companies perform better than others; are the profits of the top 10% more or less volatile than the bottom 10%?
- ✿ There are plenty of questions that we could look into next, and it's easy to see how the flow of working in a notebook can match one's own thought process.



Jupyter - more

Part IV



Enabling Jupyter Notebook extensions



- There are [multiple ways](#) to install contributed extensions. For this example, we will use pip.

```
sudo -E pip install jupyter_contrib_nbextensions
```

- Next, add the notebook extension style files to the Jupyter configuration files.

```
sudo -E jupyter contrib nbextension install --sys-prefix
```

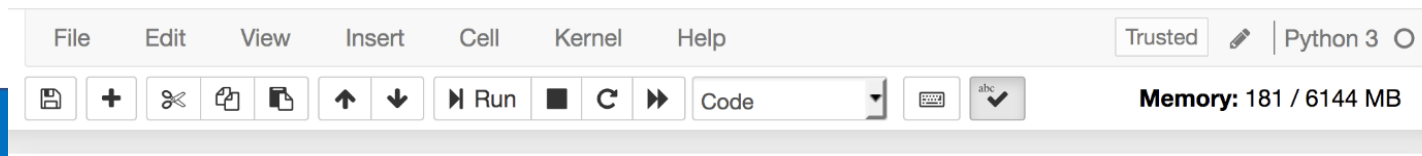
- Then, you will enable the extensions you would like to use. The syntax for this is `jupyter nbextension enable` followed by the path to the desired extension's main file. For example, to enable [scratchpad](#), you would type the following:

```
sudo -E jupyter nbextension enable scratchpad/main --sys-prefix
```

- When this is completed, the enabled extension should be visible in the extension list:

```
jupyter nbextension list
```

- One can also verify the availability of the extension via its user interface in the notebook. For example, spellchecker adds an ABC checkmark icon to the interface.



Getting your class going with Jupyter



✿ Local installation on students' or lab computers

✿ The benefits of installation on student-owned computers include:

- ✿ Once students have the software on their computers, they always have access to it; they can work anywhere, and they can use it for internships, jobs, and other non-school activities.
- ✿ It is easy for them to install additional packages later.
- ✿ Students learn to install and set up Jupyter, and software in general, which is a skill they are likely to need.
- ✿ The total computing power for the class scales with the number of students, as long as each student has enough CPU power and memory to support the intended applications.
- ✿ You can adopt Jupyter without support or resources from your institution.
- ✿ Students learn to use Jupyter on their preferred OS, e.g. Linux, Mac, or Windows, which means they are already familiar with the basic idioms of their OS.

✿ Drawbacks include:

- ✿ This approach is only possible if every student owns a computer with enough capacity.
- ✿ Students with less powerful computers might be at an unfair disadvantage.
- ✿ Although installation is generally easy, it still takes time. The time one spends at the beginning of a class can be worthwhile for a semester-long course that uses Jupyter throughout, but it is a barrier to using Jupyter for a single module or one-off assignment in a course about something else.
- ✿ Also, the amount of time spent debugging esoteric problems scales with the number of students: a class of 25 students is bound to have a few people with 32-bit processors, incompatible libraries, out-of-date operating systems, over-zealous virus checkers, etc., and a class with 100 students will have four times as many. One work-around is to have students work in pairs: the probability that more than half of the students cannot get it working is reduced.
- ✿ Discrepancies in installed library versions can cause issues for students and may lead to different behaviors when students run code.

Getting your class going with Jupyter



🌟 Jupyter on lab computers

🌟 Using lab computers instead of student-owned computers has the benefits of uniformity and improved equity. Each student will have exactly the same setup, and the instructions will work the same for everyone. This reduces the amount of individual tech support required and guarantees that all students have access to enough computational power.


🌟 However, this deployment has some disadvantages:

- 🌟 Depending on how much control you have of the computer lab, you might need institutional permission and support.
- 🌟 Students might be limited to working on assignments only when they are on campus and when computer labs are open, which might be an unfair disadvantage for non-resident students or those with full time jobs.
- 🌟 It might be difficult to install additional packages as the need arises, and students might not be allowed to install packages they need for projects.

Getting your class going with Jupyter



Jupyter on remote servers

 Even when Jupyter runs locally, it runs as a web application; that is, it runs in a browser connected to a server. In a local installation, the browser and the server run on the same machine. But it is also possible to run the server remotely.

 In that case, students don't have to install anything; they only have to run a browser and load a URL.

 There are several ways to run Jupyter on a remote server:

 You can run Jupyter on a server owned by you or your institution.

 You can run Jupyter in a temporary environment running in the cloud.

 You can run Jupyter in a persistent environment running in the cloud.

Getting your class going with Jupyter



🌟 Running in a temporary environment in the cloud

- 🌟 The easiest option for running Jupyter in the cloud is to use a cloud service that provides temporary environments. Some of these services are free of cost, and you can use them without installing anything.
- 🌟 These environments are well-suited for short examples in classes that do not use Jupyter extensively. Students can open a notebook and start running with the push of a button.
- 🌟 However, there are some limitations to these services:
 - 🌟 If your notebooks depend on particular packages, or particular versions of packages, it can be difficult to satisfy these requirements.
 - 🌟 These services run notebooks in a temporary environment that disappears if it is left idle. So they might not be suitable for managing student work.
 - 🌟 Some of these services do not guarantee a level of service and may not be as reliable as you need for a class or workshop.




Binder mybinder.org

Binder is an open-source service provided by Project Jupyter. It allows the owner of a set of notebooks residing in a public repository to pre-build an image in the Binder service, and get a shareable link that any visitor can use to obtain a working instance of JupyterHub, pre-loaded with the notebooks in the repository. The session is temporary (any changes the user makes will be deleted when closing the tab or window), but it's fully interactive. Binder is currently one of the favorite services for running one-off workshops or tutorials.

Getting your class going with Jupyter



Running on servers you control

-  If you have access to a server or cluster with enough computing power to support your class—including CPU and especially memory—you can provide a Jupyter as a service using **JupyterHub**.
-  JupyterHub is open-source software that provides a cloud-based Jupyter application for each user in a group. Each user has their own account and home directory on the server. The Hub, JupyterHub's central system, allows authenticating users and starting individual Jupyter notebook servers. Programs that start notebook servers can use a variety of technical solutions.
-  Once the Hub starts a user's notebook server, the Jupyter Notebook running in the cloud behaves just like Jupyter does when installed on an individual's computer, but JupyterHub will be running notebooks and storing files on a remote cloud computer. Students can download notebooks stored in the cloud to their local computer if they wish to work with a local installation as well. Additionally, students can upload notebooks (and other files) from their local computer to the cloud.

Getting your class going with Jupyter



- ✿ Providing a JupyterHub service offers several benefits. First, students get up and running immediately—they spend no time installing software. They navigate to a web URL, log in to JupyterHub, and begin using Jupyter. This ability to quickly log in and begin computing is a powerful way to get students to engage with the lesson, builds confidence, and avoids the sometimes-stressful experience of installing software on the student's computer.
- ✿ However, running JupyterHub on your own server has drawbacks:
 - ✿ Getting started is not easy; most instructors would require (or at least benefit from) institutional support that may not be available.
 - ✿ It can be difficult to scale: if the number of students increases, you might need more computing power. And the load students generate can be uneven; for example, if everyone runs a computationally-intensive example at the same time, your server might not be able to handle it.
 - ✿ This option can be expensive, unless you already have servers with sufficient power.

Getting your class going with Jupyter



🌟 Running Jupyter in the cloud

- 🌟 If you or your institution don't own computing hardware with the power to support your class, you can run JupyterHub on virtual servers provided by cloud services like AWS and Microsoft Azure.
- 🌟 Commercial offerings also exist to use Jupyter in the cloud, some of which provide free trials or a “freemium” pricing model. They include:
 - 🌟 CoCalc (<https://cocalc.com>) is an online computing environment that allows multiple users to edit a Jupyter notebook simultaneously. It also allows the notebook user to cycle through the revision history of a notebook and provides a number of popular kernels by default. The service includes the ability to share files with project collaborators
 - 🌟 Gryd (<https://gryd.us>) is another subscription service with a free tier. It includes course-management features, like a way to create a course, invite students, and deploy auto-graded assignments.
 - 🌟 Kaggle Kernels (<https://kaggle.com/kernels>)
 - 🌟 Microsoft Azure notebooks (<https://notebooks.azure.com/>)
 - 🌟 Amazon Sagemaker (<https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-prepare.html>)
 - 🌟 Google Colaboratory (<https://colab.research.google.com/>)

Learning management systems



✿ Many instructors use a Learning Management System (LMS) to communicate with students. These tools offer private file sharing and assignments that connect to the students' institutional computing accounts and they can be used to distribute and collect notebooks as text files. However, most LMS tools are not yet notebook-aware, so they don't render notebooks or make it easy for instructors to comment on or grade them.

✿ Web hosting

✿ Notebooks can be publicly hosted on any website, so students can download the files by clicking on a link. Most web-hosting software is not notebook-aware, but one can use *nbviewer* to share public notebooks, rendered as a static web page.

Sharing Notebooks



- ✿ When people talk about sharing their notebooks, there are generally two paradigms they may be considering.
- ✿ Most often, individuals share the end-result of their work, which means sharing non-interactive, pre-rendered versions of their notebooks. However, it is also possible to collaborate on notebooks with the aid of version control systems such as Git or online platforms like Google Colab.
- ✿ Before You Share: A shared notebook will appear exactly in the state it was in when you export or save it, including the output of any code cells. Therefore, to ensure that your notebook is share-ready, so to speak, there are a few steps you should take before sharing:
 - ✿ Click “Cell > All Output > Clear”
 - ✿ Click “Kernel > Restart & Run All”
 - ✿ Wait for your code cells to finish executing and check ran as expected
- ✿ This will ensure your notebooks don’t contain intermediary output, have a stale state, and execute in order at the time of sharing.

🌟 GitHub has integrated support for rendering .ipynb files directly both in repositories and gists on its website.

🌟 Once you have a GitHub account, the easiest way to share a notebook on GitHub doesn't actually require Git at all. Since 2008, GitHub has provided its Gist service for hosting and sharing code snippets, which each get their own repository. To share a notebook using Gists:

- 🌟 Sign in and navigate to gist.github.com.
- 🌟 Open your .ipynb file in a text editor, select all and copy the JSON inside.
- 🌟 Paste the notebook JSON into the gist.
- 🌟 Give your Gist a filename, remembering to add .ipynb or this will not work.
- 🌟 Click either “Create secret gist” or “Create public gist.”

🌟 If you created a public Gist, you will now be able to share its URL with anyone, and others will be able to fork and clone your work.

🌟 Creating your own Git repository and sharing this on GitHub is beyond the scope of this tutorial, but GitHub provides plenty of guides for you to get started on your own.



```
685     "codemirror_mode": {
686       "name": "ipython",
687       "version": 3
688     },
689     "file_extension": ".py",
690     "mimetype": "text/x-python",
691     "name": "python",
692     "nbconvert_exporter": "python",
693     "pygments_lexer": "ipython3",
694     "version": "3.9.7"
695   }
696 },
697 "nbformat": 4,
698 "nbformat_minor": 5
699 }
700 }
```

JupyterHub



- ❁ If your students are using JupyterHub, you can place notebooks and any related files directly into the students' directories manually or via a script.
- ❁ If *nbgrader* is available on your JupyterHub instance you can use it to collect and distribute notebooks (whether or not you choose to use nbgrader's assessment features).
 - ❁ This allows you to develop the notebooks and incrementally make them visible to the students for them to “fetch”.
 - ❁ They can then edit the notebooks or create new ones in the directory created in their storage space, and then publish their notebooks back to you for downloading, viewing, or assessing with the nbgrader tools (see the next section for details on this tool).

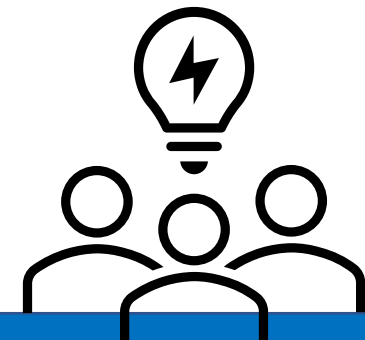
nbgrader is a tool for creating, handling, and automatically grading assignments based on Jupyter notebooks. It works as a Jupyter extension that the course creator installs on their computer. nbgrader is a flexible project in the Jupyter ecosystem that allows the distribution and collection of materials. As its name implies, it also can grade assignments; it can be used in a distributed manner where each student is running Jupyter on their own computers, or in a centralized manner, for example, if the students each have an account on a JupyterHub installation.

Using an LMS and nbgrader together



🔗 The following is a strategy that works with current tools:

1. The instructor creates an assignment notebook using nbgrader, then distributes the assignment to students via an LMS.
2. Students complete the assignment and upload the solution to the LMS.
3. The instructor downloads the completed assignments as a zip file and extracts the students' solutions in a Jupyter environment.
4. Instructors and graders use nbgrader to grade the assignment and save the grades to a CSV file.
5. The CSV file is then uploaded to the LMS.



Using an LMS and nbgrader together



- ✿ Many educators develop course-assessment activities as Jupyter notebooks. This includes exams, in-class activities, homework assignments, and projects.
 - ✿ Simple ways to handle the assessment of a notebook-based submission: have students either print them out, email them, submit them as a standard electronic document (say, into the LMS), or drop them into a shared folder. At that point, the instructor can mark and grade them in a traditional manner, for example by writing comments on a printout or adding annotations to a PDF.
- ✿ *nbgrader* allows code cells in a notebook to be marked to be auto-graded or manually graded. An instructor can then create an assignment that can be completely auto-graded, requiring little work after the notebook has been created.
 - ✿ This makes grading much easier and scales well with large class sizes. However, creating such an auto-graded notebook in nbgrader can be quite time-consuming.
- ✿ Pedagogically, a completely auto-graded notebook may have serious downsides. For example, students learn better when they can actively connect a topic to their own interests. One method of encouraging this is to have a “reflection” question on each submission. Such a reflection question can encourage students to comment on the material in a personal way, but it cannot be auto-graded.
 - ✿ To address this, you can create manually graded cells for a portion of an assignment and provide written feedback to the student.
- ✿ Some third-party notebook-based assessment solutions do exist. For example [CoCalc](#), [Vocareum](#), and [Gryd](#) provide a cloud notebook platform that can also grade assessments similar to or using nbgrader. Berkeley uses DataHub for their large Data8 course.

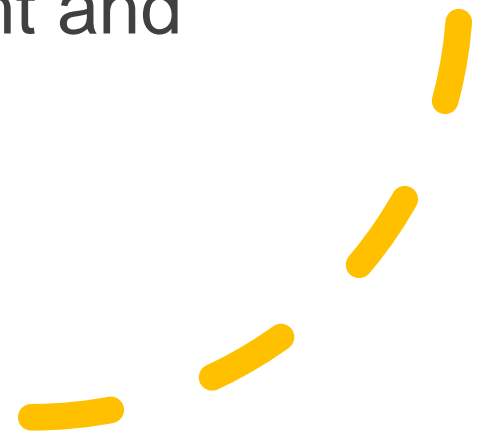
Data Science - more

Part V



Exploratory data analysis

- 🌟 Before building any model, it is essential to understand the data set.
- 🌟 **EDA** (*exploratory data analysis*) is a crucial step in building any successful model.
- 🌟 The dataset used for this model is straightforward. We are trying to predict the weight of the individual based on their height. So let us first see if there is any correlation between height and weight of the individual....



Descriptive statistics



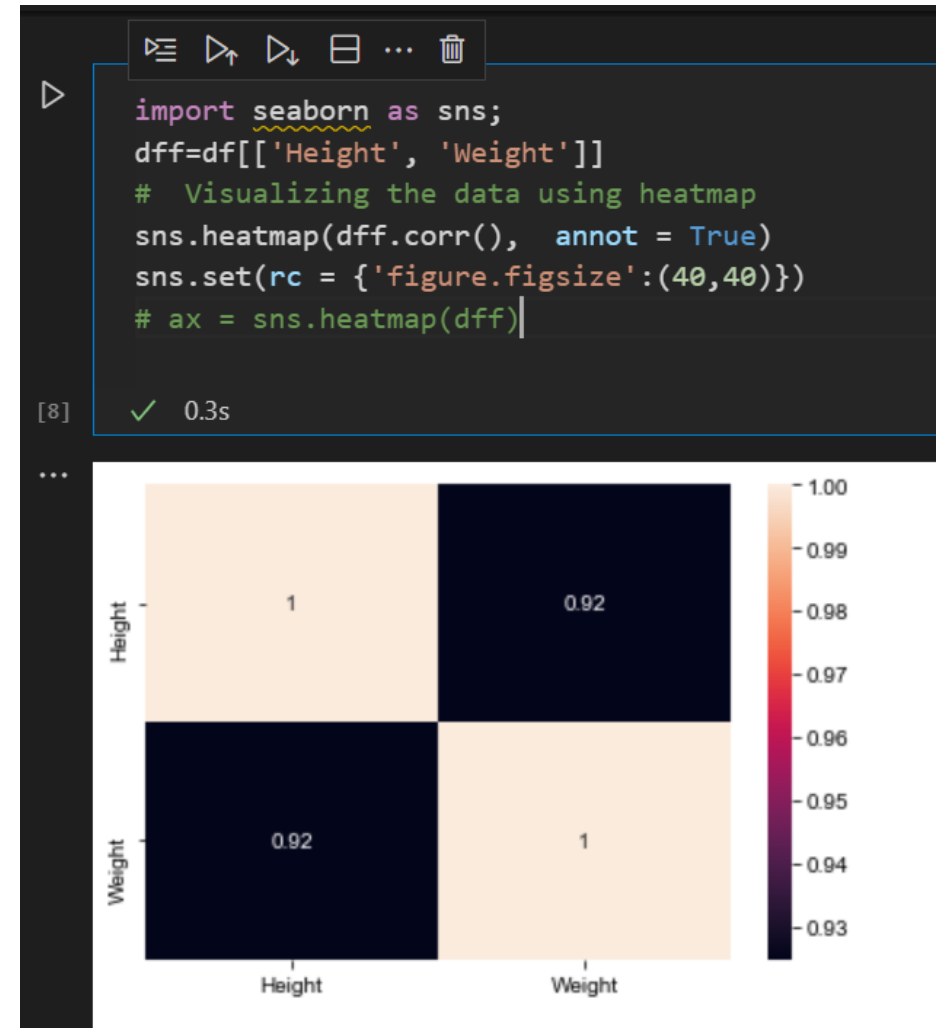
- Run heatmap plot to check for correlation between dependent and independent variables.
 - Heatmap is a two-dimensional visual representation of the variables. It is considered one of the best visual graphical representations when you want to show complex data. It is comprised of square boxes, and the x-axis and y-axis represent the variables. For correlation, each square box represents the level of correlation between the variable on the X-axis Vs. variable on the Y-axis. The annotations shown in each square box represent the correlation value. If you look across the matrix, the correlation value for weight and height is 0.92, which indicates that these variables are highly correlated. Please note that if you look diagonally, the values will always be one as you are just comparing the variable with itself.

Dependent variable: Weight →

Independent variable: Height

Regarding correlation:

- Values closer to 1 show a high correlation, and values closer to 0 show the lowest correlation.
- A positive correlation value shows that the variables are positively correlated. For example, taller individuals usually weigh more.
- A negative correlation value shows that the variables are negatively correlated.



This is our example



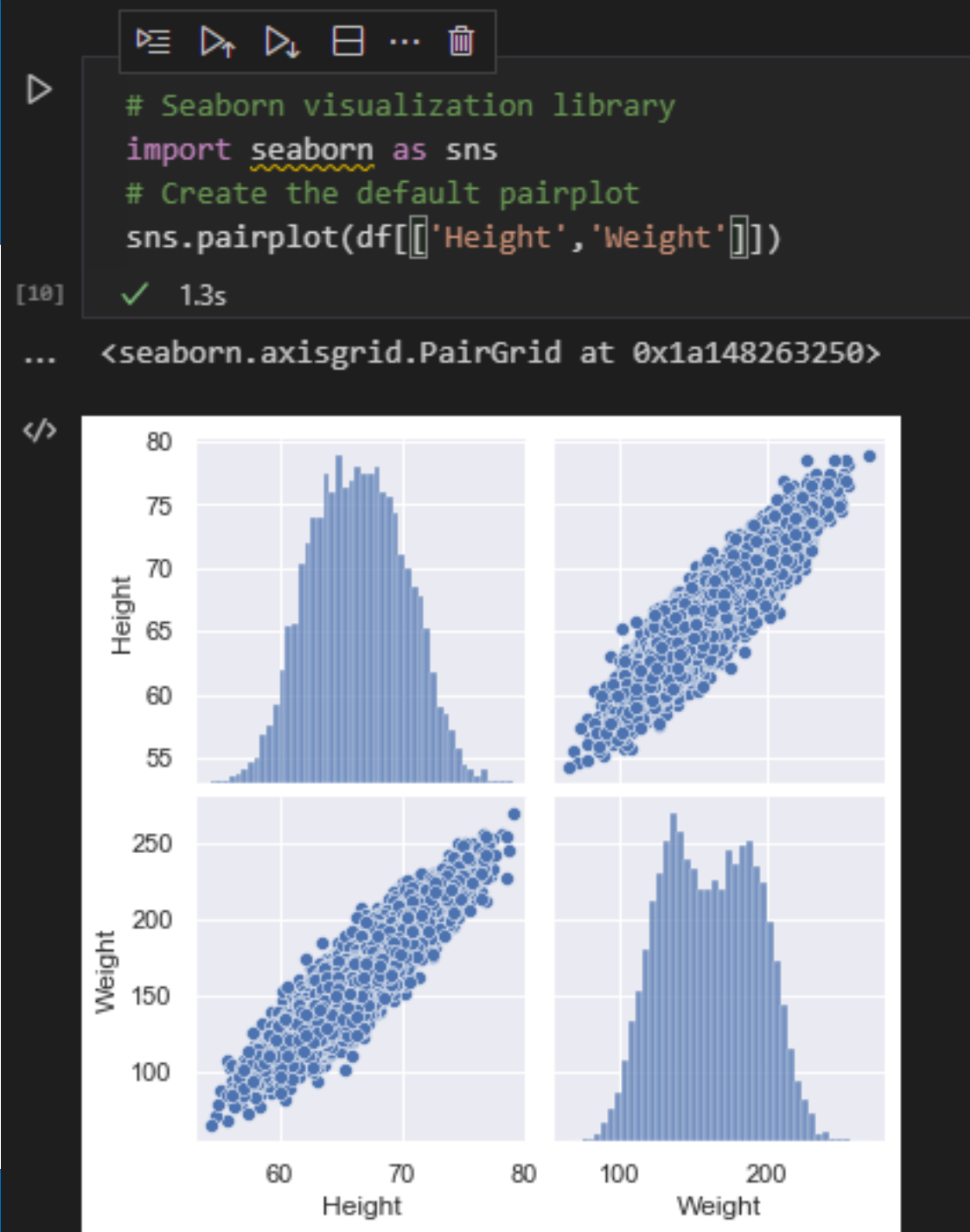
	A	B	C	D	E	F	G	H
1	Date	Temperature - Low °F	Temperature - High °F	Humidity	Sunrise	Sunset	Chances of rain	Weather
2	1/1/2018	78	97	19%	5:35 AM	8:30 PM	0%	Partly cloudy
3	1/2/2018	89	99	16%	5:34 AM	8:29 PM	1%	Cloudy
4	1/3/2018	88	90	18%	5:34 AM	8:29 PM	50%	Sunny
5	1/4/2018	78	95	17%	5:35 AM	8:26 PM	1%	Windy
6	1/5/2018	76	92	16%	5:34 AM	8:24 PM	5%	Partly cloudy
7	1/6/2018	77	100	16%	5:33 AM	8:23 PM	11%	Windy
8	1/7/2018	98	107	15%	5:35 AM	8:25 PM	13%	Sunny
9	1/8/2018	87	102	15%	5:34 AM	8:23 PM	14%	Cloudy
10	1/9/2018	88	103	14%	5:33 AM	8:30 PM	14%	Partly cloudy
11	1/10/2018	89	104	14%	5:35 AM	8:29 PM	15%	Windy
12	1/11/2018	90	106	13%	5:34 AM	8:29 PM	15%	Cloudy
13	1/12/2018	90	107	13%	5:35 AM	8:26 PM	16%	Cloudy
14	1/13/2018	91	108	12%	5:35 AM	8:24 PM	16%	Sunny
15	1/14/2018	92	109	12%	5:34 AM	8:23 PM	17%	Sunny

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

```
import pandas as pd
df=pd.read_csv(r'C:\Users\CA34789\Desktop\Personal\Blog\Data Science Learning content\module 2.csv', encoding='latin-1')
df.head()
```

Descriptive statistics

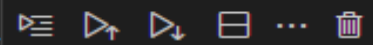
- Plot a pair plot to visually represent the positive/negative correlation between the variables.
- Based on the pair plot, the Height is positively correlated to Weight of the individual.
- Now, let's build a model that will predict the individual's weight based on height....



Create, Train and Test datasets



- ✿ We are first splitting the dataset into train and test data sets. We then build the model based on the train data set and test it on the test dataset...

```
▶ 
# Create array for independent and dependent variables
X = df.iloc[:, 1:2].values
Y = df.iloc[:, 2].values

# Splitting the variables as training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size = 0.7,
                                                    test_size = 0.3, random_state = 100)

print('Row count of X_train table' + ' - ' + str(f"{len(X_train):,}"))
print('Row count of y_train table' + ' - ' + str(f"{len(y_train):,}"))

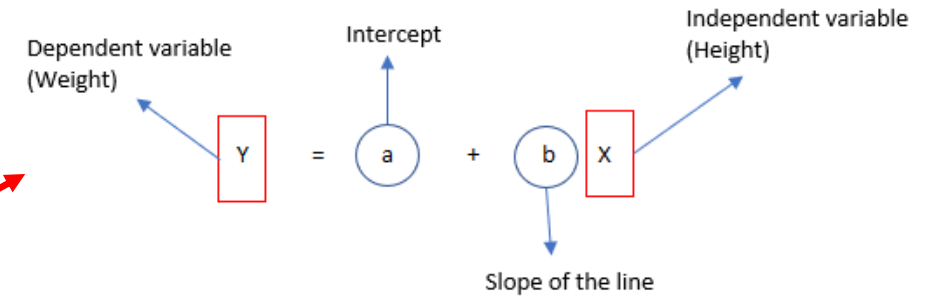
print('Row count of X_test table' + ' - ' + str(f"{len(X_test):,}"))
print('Row count of y_test table' + ' - ' + str(f"{len(y_test):,}"))

[19] ✓ 0.4s
... Row count of X_train table - 7,000
Row count of y_train table - 7,000
Row count of X_test table - 3,000
Row count of y_test table - 3,000
```

Build a simple Linear Regression Model



- 🌐 We are ready to build a simple linear regression model.
- 🌐 We will be using existing python packages to build the model.
 - 🌐 Sklearn
 - 🌐 Statsmodel
- 🌐 The equation for simple linear regression:
- 🌐 The Statsmodel library that we will be using for building the linear regression model will fit a line for our dataset. Let's input the intercept value using the Statsmodel library.



```
[29] In [29]: # Importing Statsmodels.api library from Statsmodel package
import statsmodels.api as sm

# Adding a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

print('Intercept added to the linear regression model - '+str(X_train_sm))

... Intercept added to the linear regression model - [[ 1.          70.06468847]
 [ 1.          72.09890444]
 [ 1.          73.62154397]
 ...
 [ 1.          62.71958949]
 [ 1.          58.12602402]
 [ 1.          64.12314824]]
```

Fit the regression line



- ❁ R-squared value is 0.855. This shows that 85.5% of the variance in the weight can be explained by height.
- ❁ F statistic is pretty low, which shows that the model fit is statistically significant.
- ❁ The coefficient for height is 7.7 and P-value is close to 0. This shows that the coefficient is statistically significant.
- ❁ Based on the summary, we can now derive the linear regression equation using intercept and slope.

$$\text{Weight} = -351.4621 + 7.7275 * \text{height}$$

```
# Fitting the regression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()

# Printing the parameters
lr.params

# Linear regression summary
lr.summary()
```

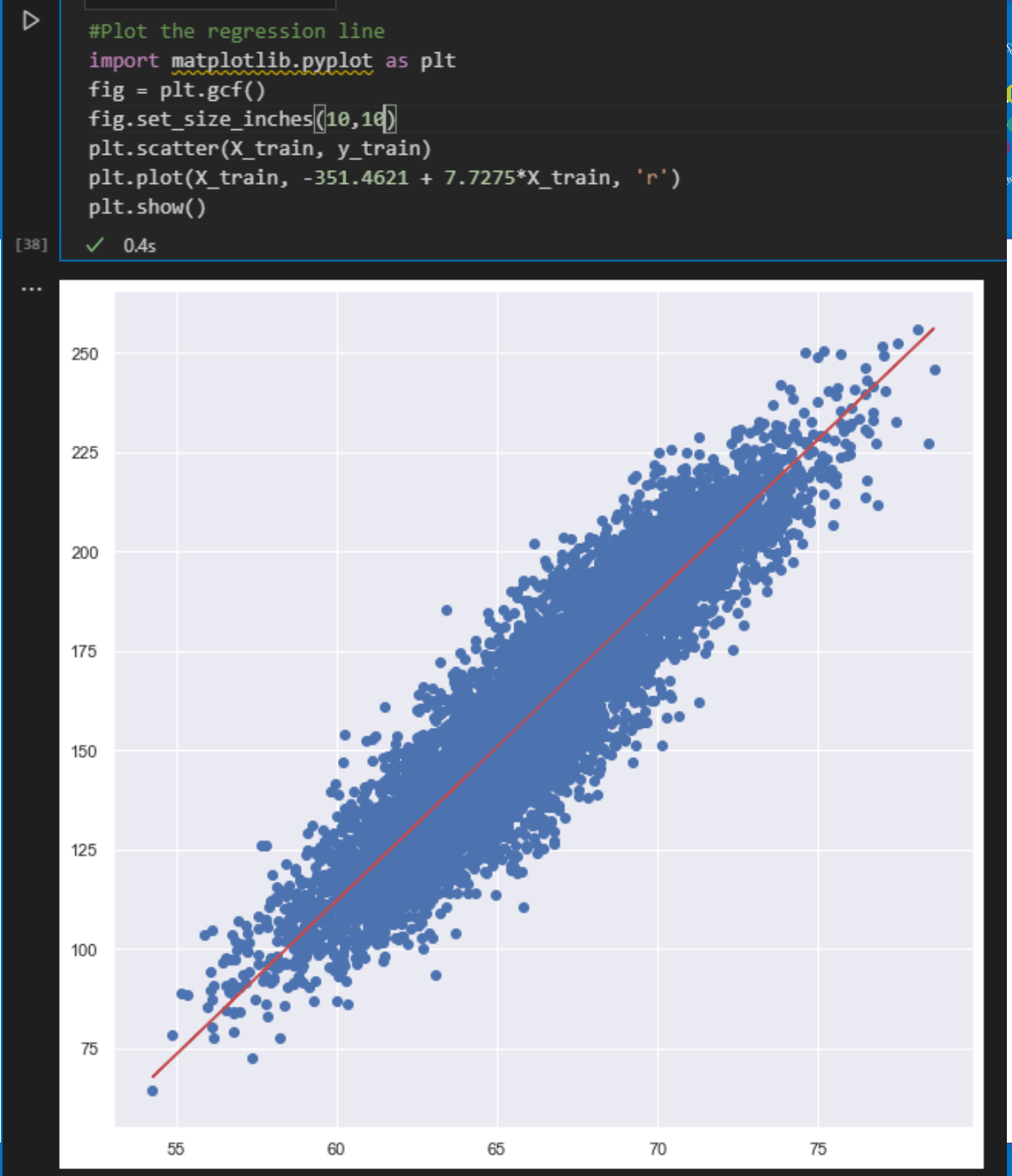
[31] ✓ 0.9s

... OLS Regression Results

Dep. Variable:	y	R-squared:	0.855			
Model:	OLS	Adj. R-squared:	0.855			
Method:	Least Squares	F-statistic:	4.124e+04			
Date:	Mon, 06 Sep 2021	Prob (F-statistic):	0.00			
Time:	08:07:49	Log-Likelihood:	-27455.			
No. Observations:	7000	AIC:	5.491e+04			
Df Residuals:	6998	BIC:	5.493e+04			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-351.4621	2.531	-138.840	0.000	-356.424	-346.500
x1	7.7275	0.038	203.073	0.000	7.653	7.802
Omnibus:	1.675	Durbin-Watson:	1.998			
Prob(Omnibus):	0.433	Jarque-Bera (JB):	1.708			
Skew:	0.035	Prob(JB):	0.426			
Kurtosis:	2.970	Cond. No.	1.15e+03			

Plotting

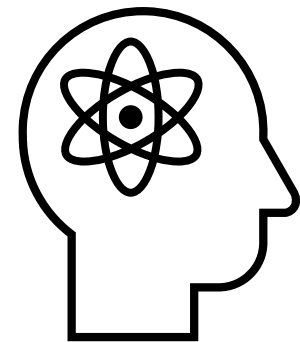
Plot the regression line using the equation: $y = -351.4621 + 7.7275x$



Exercise #1



- 🌐 Go to https://opendata.swiss/en/dataset?keywords_en=precipitation and download any open dataset
- 🌐 Analyse your data, try to do a data analysis
- 🌐 Produce a documented Jupyter Notebook to illustrate the thinking process for your scientific work methodology



2nd example



Building a Recommendation System with Beer Data

 2nd day...

* From <https://www.r-bloggers.com/2019/08/building-a-recommendation-system-with-beer-data/>

A way to share educational notebooks, gain feedback on them and receive credit for your work is to publish with the Journal of Open Source Education. This is a peer-reviewed journal aimed at educators developing Open Education Resources that use code to teach. In addition to receiving a publication advertising your work, the peer-review process will result in higher quality software, code, and educational material.

THANK YOU!



Follow us

