Understanding and using code notebooks

Extensive data science with Python scripts become challenging quickly

Comments in your script tend to be terse or non-existent

Intermediate results are not immediately accessible

Coworkers spend a lot of time reading your comments

You struggle to find the right plot in your output folder, because there are so many

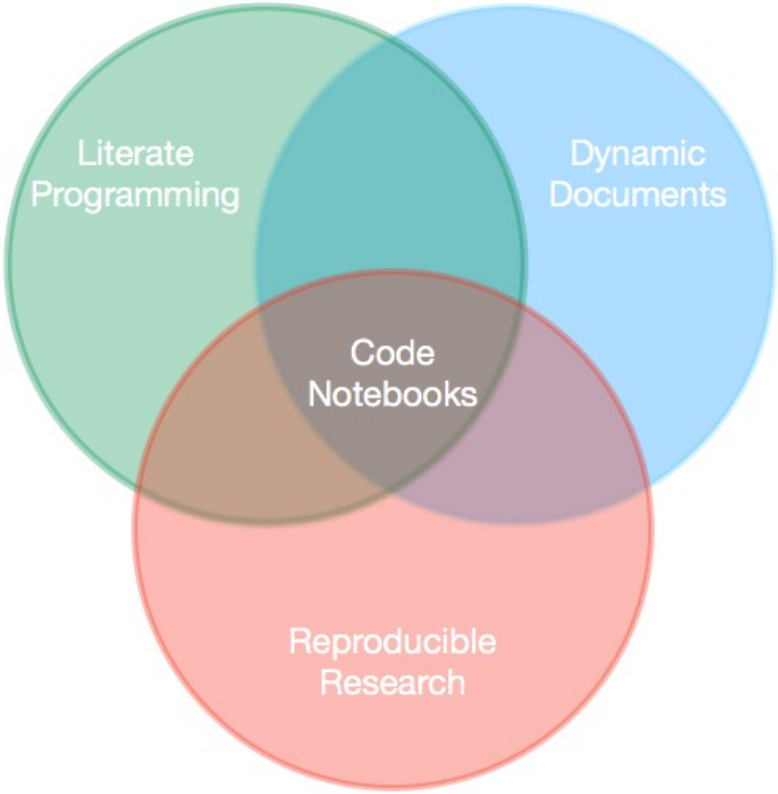
LE6_Aggregate v2.py

```
import dateutil
myData["TransDate"] = myData["TransDate"].apply(dateutil.parser.parse, dayfirst=True)

#%%
"""
#################################################################
Part 1: Aggregate on datasets
#################################################################
"""
myData.loc[myData["Customer"]==149236,]
#myData.loc[myData["Customer"]==149236,].groupby("Customer")["PurchAmount"].sum()

#%% Slide 10: Sum all purchase amounts by customer
myData.groupby("Customer", as_index=False)["PurchAmount"].sum()


#%% Slide 11: Aggregate a variable by one dimension
myData.groupby("Customer", as_index=False)["PurchAmount"].agg(["sum"]).rename(columns={"P

#%% Slide 12: Apply multiple aggregation functions to a variable by one dimension
myData.groupby("Customer", as_index=False)["PurchAmount"].agg({"AggPurch": "sum", "Purch_

#%% Slide 13: Apply an aggregating function to the whole dataset
myData["PurchAmount"].sum()

#%% Slide 15: Create new columns with the transform()-function
myData["AggPurch"]=myData.groupby("Customer")["PurchAmount"].transform(sum)
myData

#%% This will create NAs
myData["AggPurch_NAs"]=myData.groupby("Customer")["PurchAmount"].sum()
myData["AggPurch_NAs"]

#%% Slide 16: Apply an aggregating function to multiple variables
myData.groupby("Customer")["PurchAmount","Quantity"].sum()

#%% Slide 16: Apply an aggregating function to multiple variables
myData.groupby("Customer", as_index=False).agg({"PurchAmount":["max","sum"], "Quantity":"

#%% Slide 17: Aggregate multiple variables by two dimensions
myData.groupby(["Customer","TransDate"])["PurchAmount","Quantity"].sum()

#%% Slide 18: Aggregation for subset of rows
myData.iloc[0:5].groupby("Customer", as_index=False)["PurchAmount"].sum()
```

Line 1, Column 1 Tab Size: 4 Python

Scripts for data science can become messy quickly.

Code notebooks are a tool to report your code and results and make them publicly available



Code notebooks help you to:

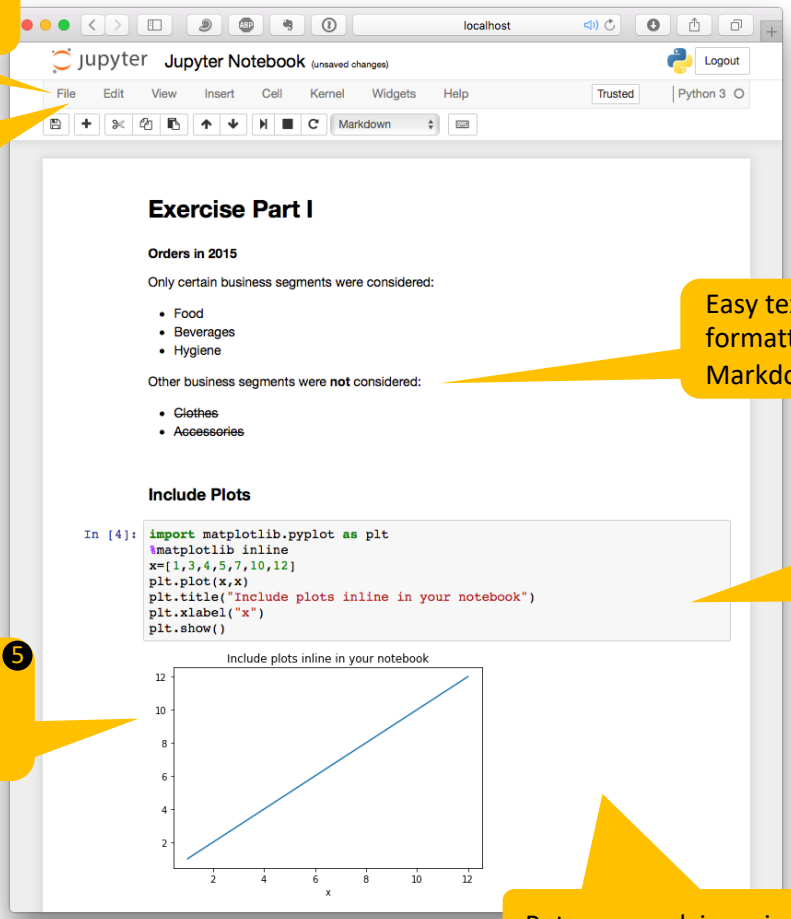
- Divide your code into manageable chunks.
  - Execute small chunks of code one by one.
  - Evaluate and store preliminary results.
- Take notes along the process in rich text format (paragraphs, equations, tables, …).
- Present and discuss your results and code chunks with colleagues.

What do you get out of code notebooks?
Example: Jupyter Notebook

This is the original look of a Jupyter Notebook- **Google Golab** uses Juypter Notebooks is a sllighty re-designed way (but the technology under the hood is the same)

1 Exporting your document will produce immediate results of your calculations

2 Different export formats, e.g. HTML or PDF



3 Easy text formatting with Markdown

4 Integration of Python code and other languages, e.g. SQL and R

5 Output of code chunks is directly displayed

6 Put your work in a nice format

Notebooks versus regular programming scripts

Activity	Scripts	Notebooks
Building a narrative	**Comments** are available to describe code.	**Rich text** is available to describe the process and conclusion
Manage output	Output is available in the console or environment.	**Output is embedded** in a single document. Code and output may be divided into separate **code chunks**.
Creating a final report	**Creating a report is a separate** and time-consuming **step**.	**Instant report possible**. Reports are publishable as HTML, PDF, …)

1 Allows very descriptive and reproducible documentation

2 Immediate update of output and report

3 Easy sharing possible

Notebooks versus regular programming scripts

Activity	Scripts	Notebooks
Building a narrative	**Comments** are available to describe code.	**Rich text** is available to describe the process and conclusion
Manage output	Output is available in the console or environment.	**Output is embedded** in a single document. Code and output may be divided into separate **code chunks**.
Creating a final report	**Creating a report is a separate** and time-consuming **step**.	**Instant report possible**. Reports are publishable as HTML, PDF, …)

1 Allows very descriptive and reproducible documentation

2 Immediate update of output and report

3 Easy sharing possible

Notebooks versus regular programming scripts

Activity	Scripts	Notebooks
Building a narrative	**Comments** are available to describe code.	**Rich text** is available to describe the process and conclusion
Manage output	Output is available in the console or environment.	**Output is embedded** in a single document. Code and output may be divided into separate **code chunks**.
Creating a final report	**Creating a report is a separate** and time-consuming **step**.	**Instant report possible**. Reports are publishable as HTML, PDF, …)

1 Allows very descriptive and reproducible documentation

2 Immediate update of output and report

3 Easy sharing possible

Notebooks versus regular programming scripts

Activity	Scripts	Notebooks
Building a narrative	**Comments** are available to describe code.	**Rich text** is available to describe the process and conclusion
Manage output	Output is available in the console or environment.	**Output is embedded** in a single document. Code and output may be divided into separate **code chunks**.
Creating a final report	**Creating a report is a separate** and time-consuming **step**.	**Instant report possible**. Reports are publishable as HTML, PDF, …)

1 Allows very descriptive and reproducible documentation

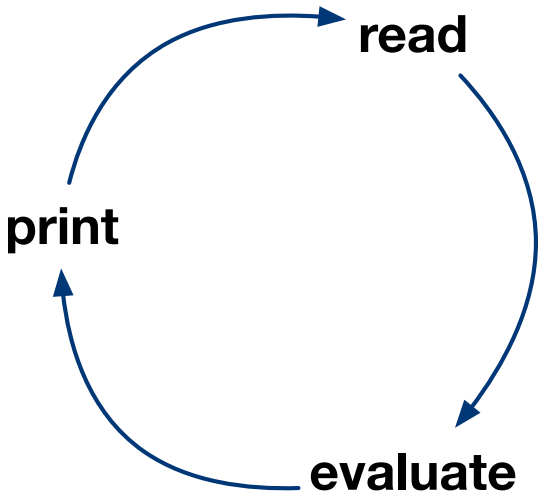
2 Immediate update of output and report

3 Easy sharing possible

Notebooks are based on the read-eval-print loop (REPL) technique

The notebook user-interface is developed on the **concept of interactive computing:**

1. User input
2. Input is evaluated
3. Result is printed and notebook is ready for next input



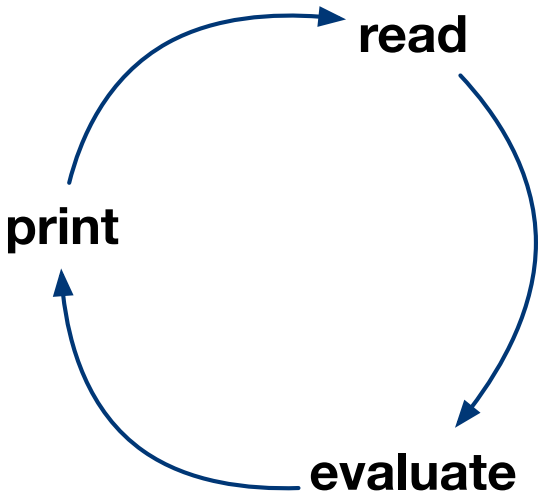
There are **two main features**:

- Store the last output for reuse
- Ability to save input and later reload and re-evaluate

Notebooks are based on the read-eval-print loop (REPL) technique

The notebook user-interface is developed on the **concept of interactive computing:**

1. User input
2. Input is evaluated
3. Result is printed and notebook is ready for next input

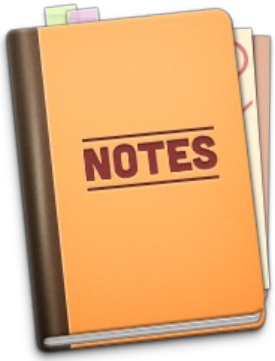


There are **two main features**:

- Store the last output for reuse
- Ability to save input and later reload and re-evaluate

There are many different notebooks, which mostly support multiple programming languages

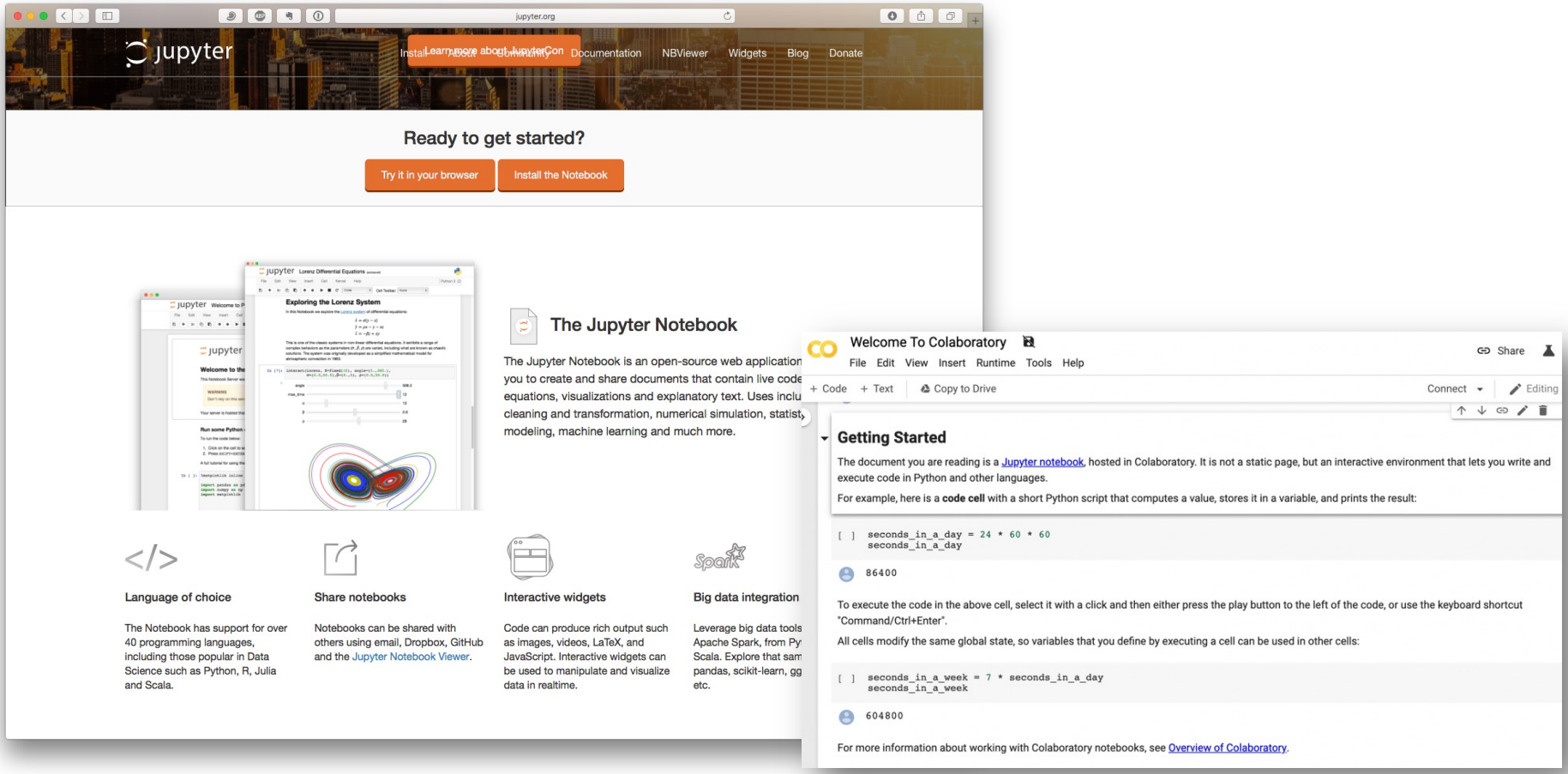
- Jupyter (e.g. used by Google Colab; R, Python; around 40 different languages)
- R Notebooks (R, Python; around 8 different languages)
- Beaker (R, Python; around 17 different languages)
- Zeppelin (R, Python; around 20 different languages)



➢ Which one to pick depends on the requirements of the programming project and your personal preference. We pick to present Jupyter notebooks due to its integration into Spyder.

http://beakernotebook.com, https://zeppelin.apache.org, http://jupyter.org, http://jupyter.org

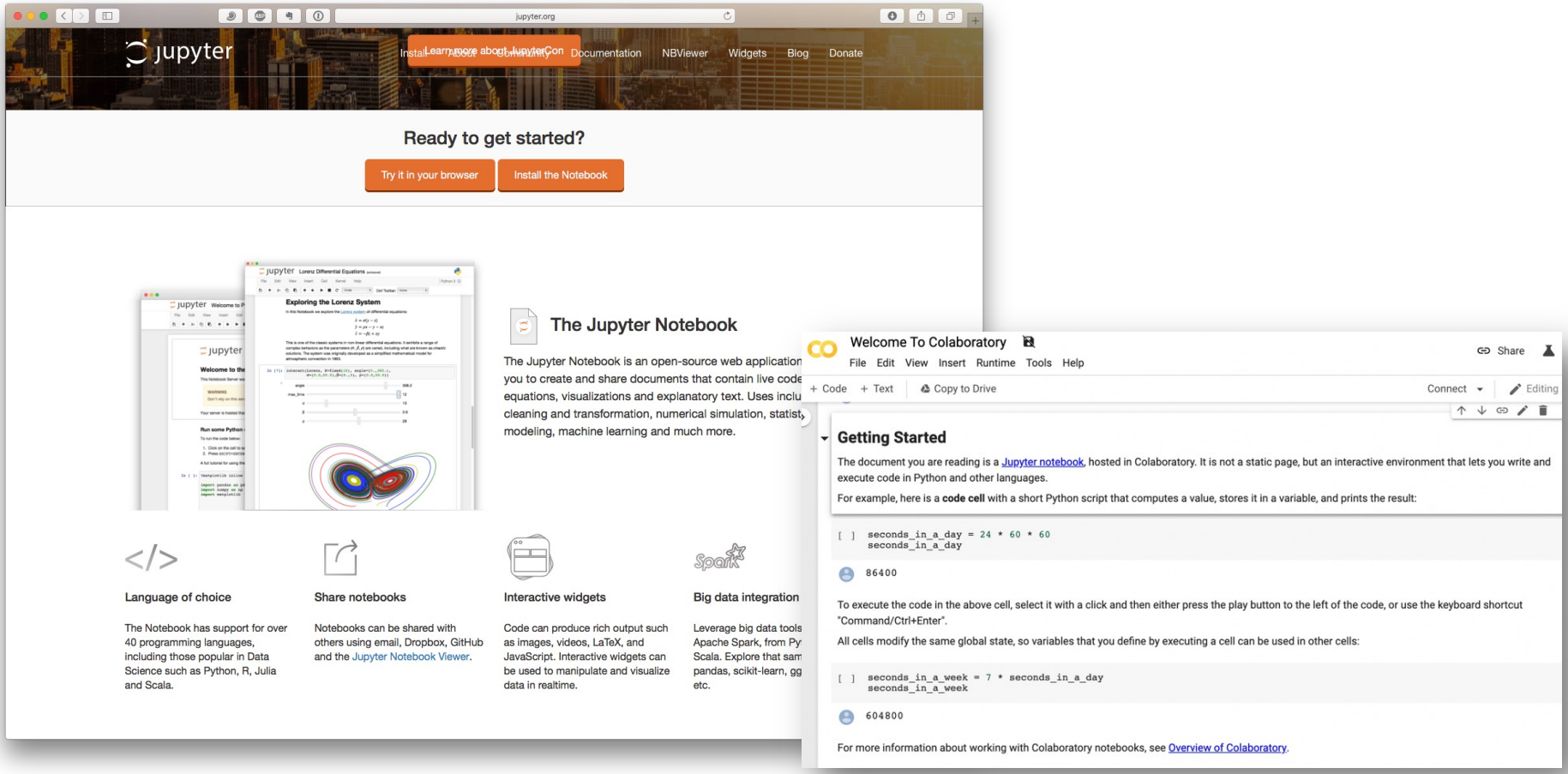
For Python, Jupyter Notebooks (which are also used by service “Google Colab”) are the most popular



http://jupyter.org

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

For Python, Jupyter Notebooks (which are also used by service “Google Colab”) are the most popular



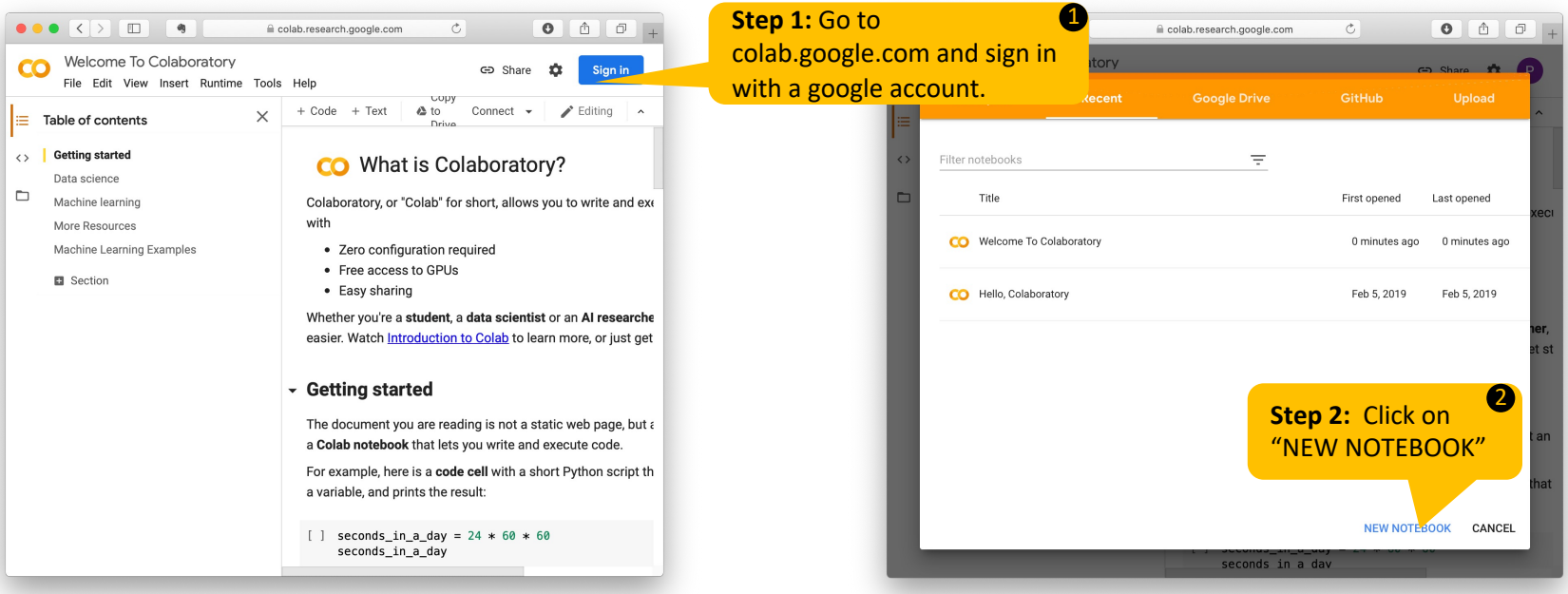
http://jupyter.org

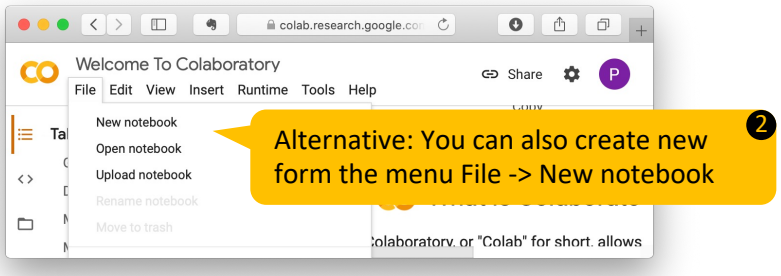
https://colab.research.google.com/

Create your first Jupyter Notebook
Steps

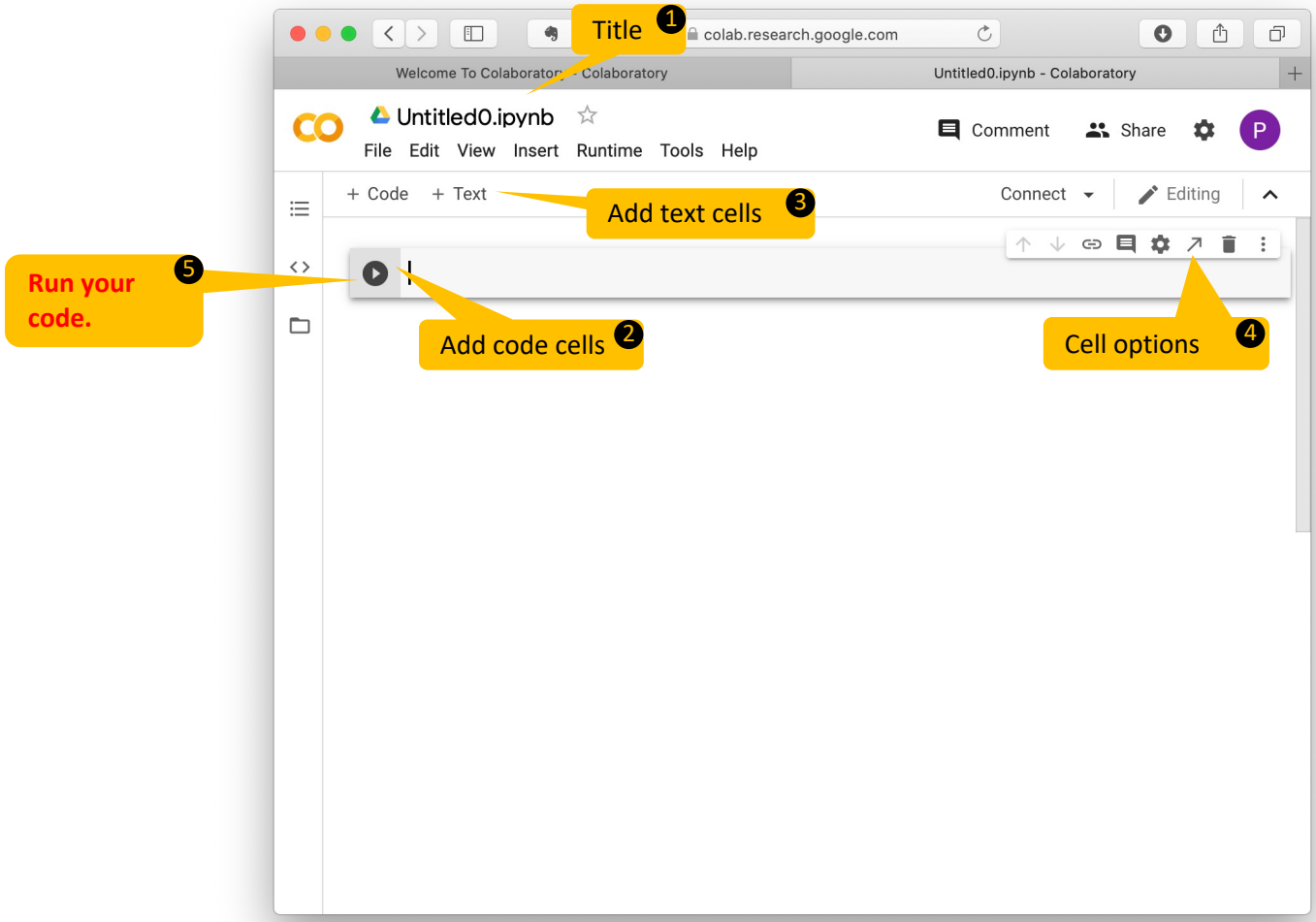
1. **Create a new Jupyter Notebook**
2. Create Content
   - Add text elements as Markdown syntax
   - Add code elements in any supported program language
   - Use LaTeX in your Jupyter Notebook

Create your first Jupyter notebook on Google Colab





Main components of a Google Colab notebook

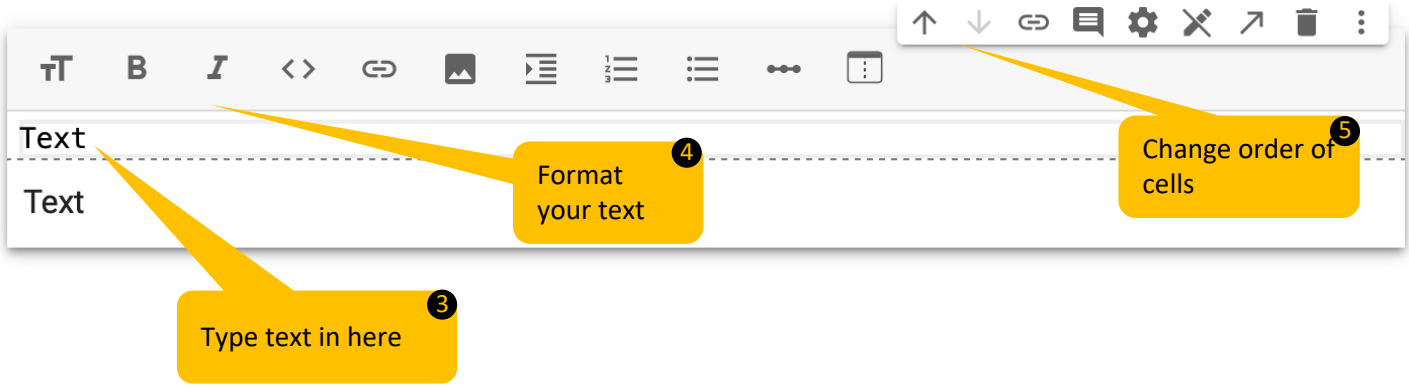


Elements of a Jupyter Notebook (which you can e.g. use through the service “Google Colab”)

Jupyter Notebooks have two major types of "cells" for content:

- **Code Cell**

- **Markdown Cells**



Understanding and using code notebooks