

# Foundations of Big Data Analysis in Python

1112BDA03

MBA, IM, NTPU (M6031) (Spring 2023)

Tue 2, 3, 4 (9:10-12:00) (B8F40)



<https://meet.google.com/paj-zhji-mya>



Min-Yuh Day, Ph.D,  
Associate Professor

Institute of Information Management, National Taipei University

<https://web.ntpu.edu.tw/~myday>



# Syllabus

Week	Date	Subject/Topics
1	2023/02/21	Introduction to Big Data Analysis
2	2023/02/28	(Day Off)
3	2023/03/07	AI, Data Science, and Big Data Analysis
4	2023/03/14	Foundations of Big Data Analysis in Python
5	2023/03/21	Case Study on Big Data Analysis I
6	2023/03/28	Machine Learning: SAS Viya, Data Preparation and Algorithm Selection

# Syllabus

**Week Date Subject/Topics**

7 2023/04/04 (Children's Day) (Day off)

**8 2023/04/11 Midterm Project Report**

9 2023/04/18 Machine Learning: Decision Trees and Ensembles of Trees

10 2023/04/25 Machine Learning: Neural Networks (NN) and  
Support Vector Machines (SVM)

11 2023/05/02 Case Study on Big Data Analysis II

12 2023/05/09 Machine Learning: Model Assessment and Deployment

# Syllabus

Week	Date	Subject/Topics
13	2023/05/16	ChatGPT and Large Language Models (LLM) for Big Data Analysis
14	2023/05/23	Deep Learning for Finance Big Data Analysis
15	2023/05/30	Final Project Report I
16	2023/06/06	Final Project Report II
17	2023/06/13	Self-learning
18	2023/06/20	Self-learning

# **Foundations of Big Data Analysis in Python**

# Outline

- **Foundations of Big Data Analysis in Python**
  - **Python Ecosystem for Data Science**
  - **Python**
    - Programming language
  - **Numpy**
    - Scientific computing
  - **Pandas**
    - Data structures and data analysis tools

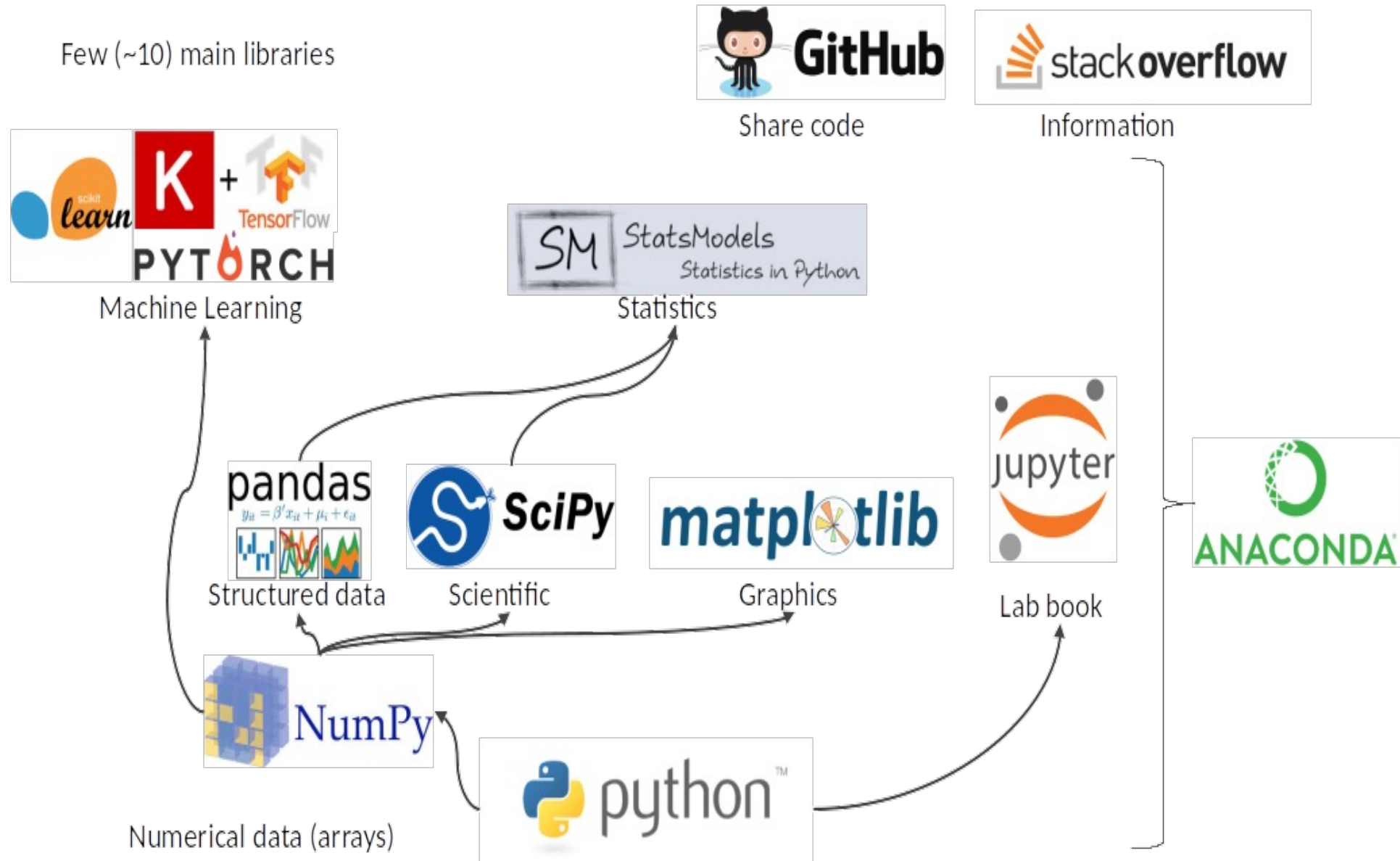


**Python**

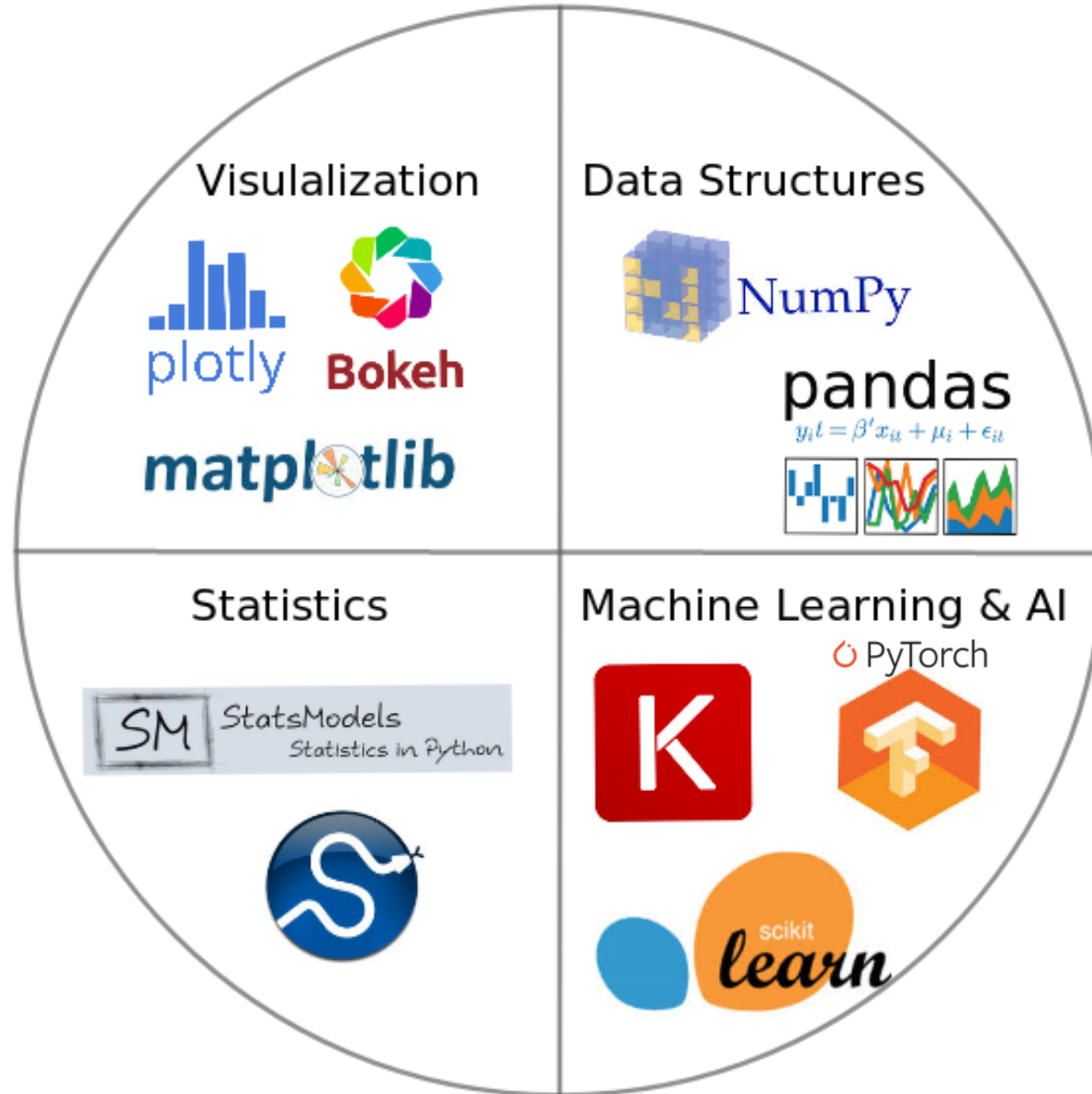
**Python** is an  
interpreted,  
object-oriented,  
high-level  
programming language  
with  
dynamic semantics.



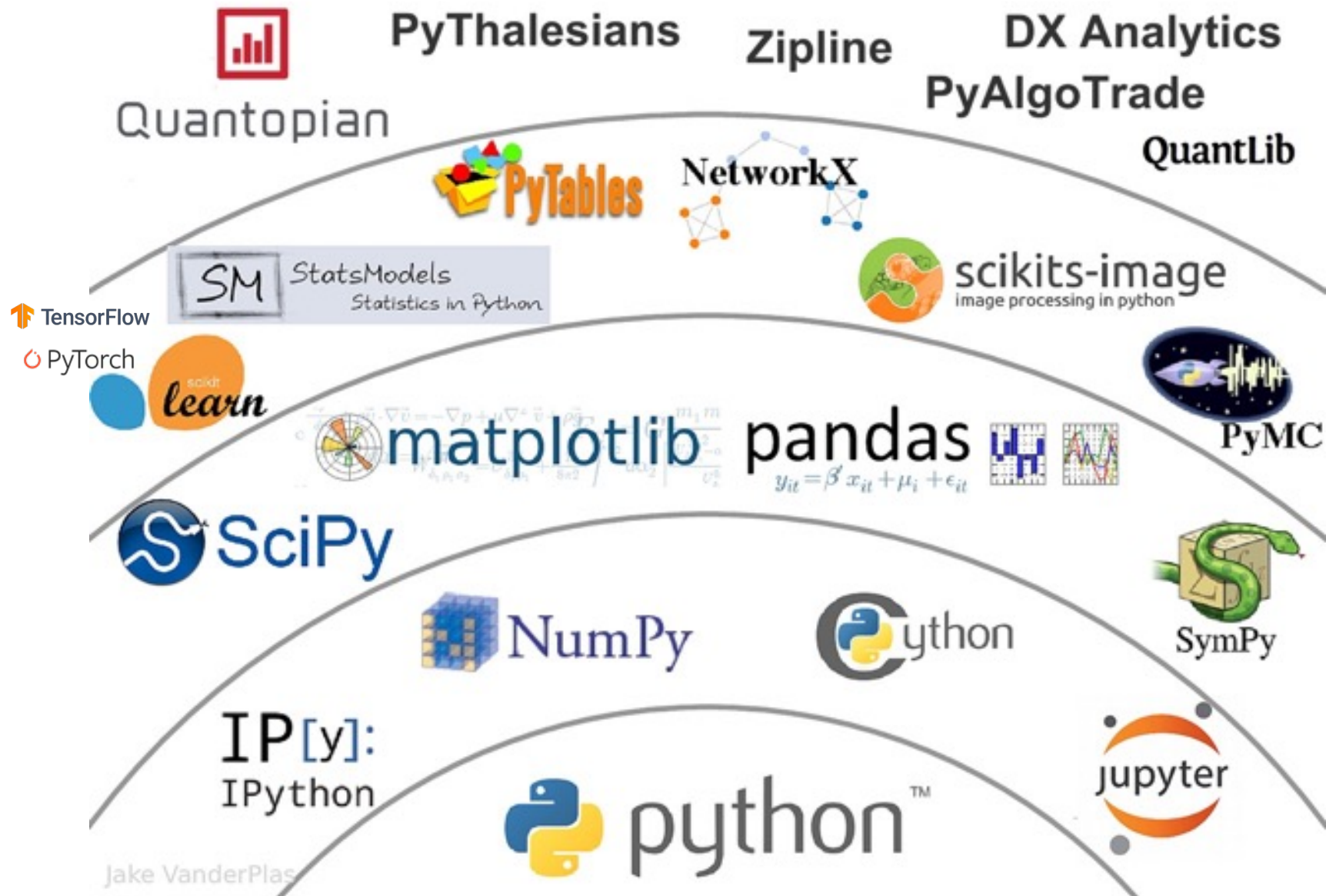
# Python Ecosystem for Data Science



# Python Ecosystem for Data Science



# The Quant Finance PyData Stack



# Google Colab

The screenshot shows the Google Colaboratory interface in a web browser. The browser's address bar displays the URL `https://colab.research.google.com/notebooks/welcome.ipynb`. The page header includes the Colab logo and the text "Hello, Colaboratory". Below the header is a menu with options: File, Edit, View, Insert, Runtime, Tools, and Help. A secondary menu contains "CODE", "TEXT", "CELL", "COPY TO DRIVE", "CONNECT", and "EDITING".

On the left side, there is a "Table of contents" sidebar with the following items: "Getting Started", "Highlighted Features" (with sub-items: TensorFlow execution, GitHub, Visualization, Forms, Examples), and "Local runtime support".

The main content area features a "Welcome to Colaboratory!" message with the Colab logo and a brief description: "Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. See our [FAQ](#) for more info." Below this is a "Getting Started" section with a list of links:

- [Overview of Colaboratory](#)
- [Loading and saving data: Local files, Drive, Sheets, Google Cloud Storage](#)
- [Importing libraries and installing dependencies](#)
- [Using Google Cloud BigQuery](#)
- [Forms, Charts, Markdown, & Widgets](#)
- [TensorFlow with GPU](#)
- [Machine Learning Crash Course: Intro to Pandas & First Steps with TensorFlow](#)

Below the list is a "Highlighted Features" section with a sub-section "Seedbank" containing the text: "Looking for Colab notebooks to learn from? Check out [Seedbank](#), a place to discover interactive machine learning examples."

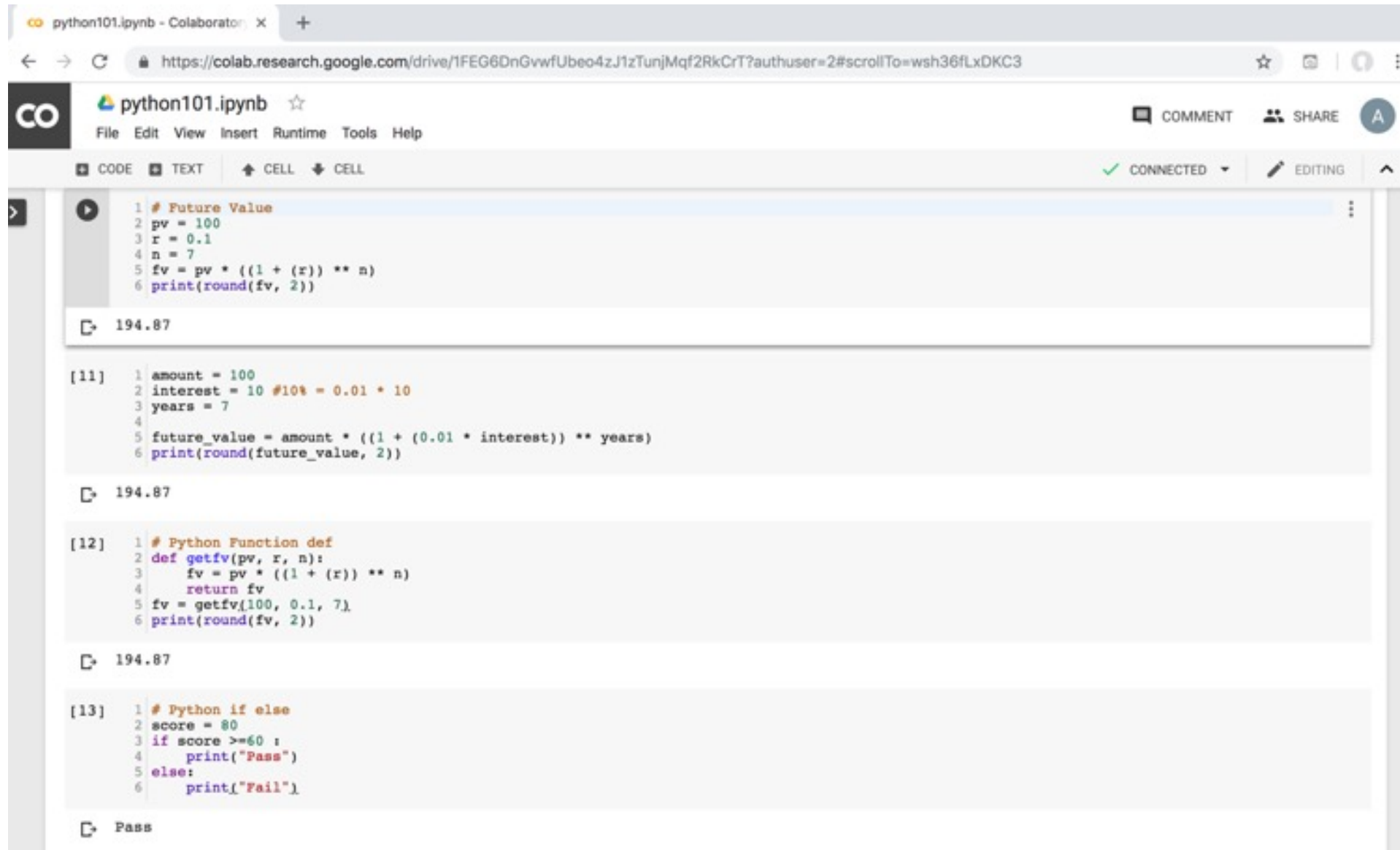
The "TensorFlow execution" section is partially visible, starting with the text: "Colaboratory allows you to execute TensorFlow code in your browser with a single click. The example below adds two matrices."

At the bottom of the visible content, there is a mathematical equation:

$$\begin{bmatrix} 1. & 1. & 1. \end{bmatrix} + \begin{bmatrix} 1. & 2. & 3. \end{bmatrix} = \begin{bmatrix} 2. & 3. & 4. \end{bmatrix}$$

# Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>



The screenshot shows a Google Colab notebook titled "python101.ipynb". The interface includes a browser address bar, a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), and a toolbar with options for CODE, TEXT, CELL, and a status indicator showing "CONNECTED" and "EDITING".

The notebook contains four code cells:

- Cell 1:** A code cell with the following Python code:

```
1 # Future Value
2 pv = 100
3 r = 0.1
4 n = 7
5 fv = pv * ((1 + (r)) ** n)
6 print(round(fv, 2))
```

The output of this cell is "194.87".
- Cell 2:** A code cell with the following Python code:

```
[11] 1 amount = 100
2 interest = 10 #10% = 0.01 * 10
3 years = 7
4
5 future_value = amount * ((1 + (0.01 * interest)) ** years)
6 print(round(future_value, 2))
```

The output of this cell is "194.87".
- Cell 3:** A code cell with the following Python code:

```
[12] 1 # Python Function def
2 def getfv(pv, r, n):
3     fv = pv * ((1 + (r)) ** n)
4     return fv
5 fv = getfv(100, 0.1, 7)
6 print(round(fv, 2))
```

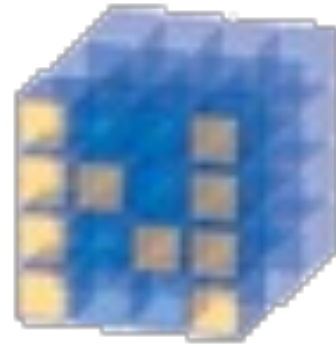
The output of this cell is "194.87".
- Cell 4:** A code cell with the following Python code:

```
[13] 1 # Python if else
2 score = 80
3 if score >=60 :
4     print("Pass")
5 else:
6     print("Fail")
```

The output of this cell is "Pass".

<https://tinyurl.com/aintpupython101>

# NumPy



NumPy

Base

**N-dimensional array**

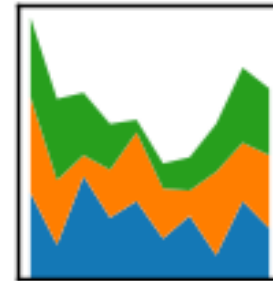
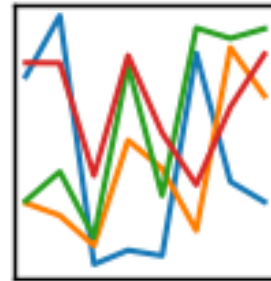
package

**Python**  
**matplotlib**  
**matplotlib**

# Python Pandas

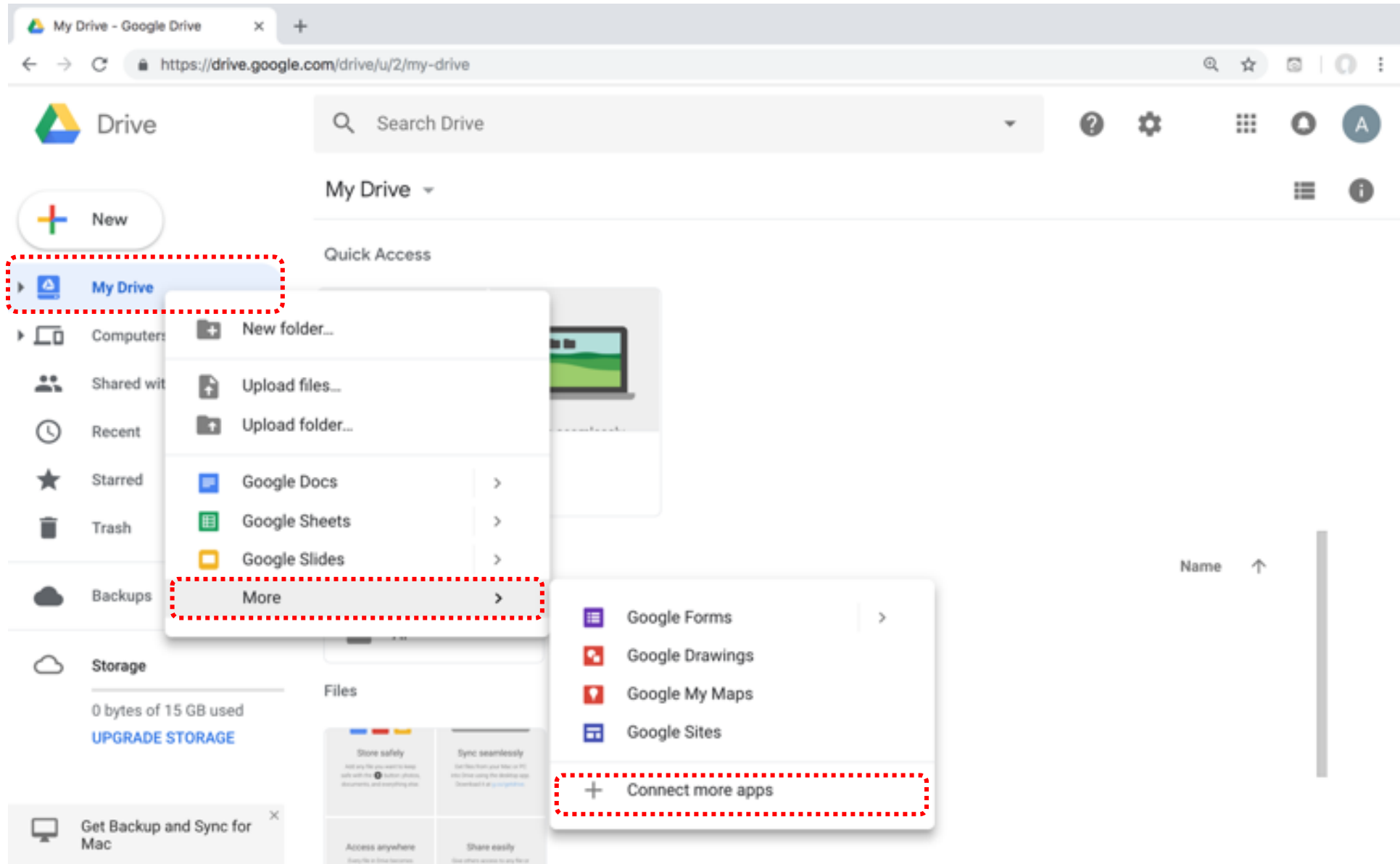
pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$





# Connect Google Colab in Google Drive

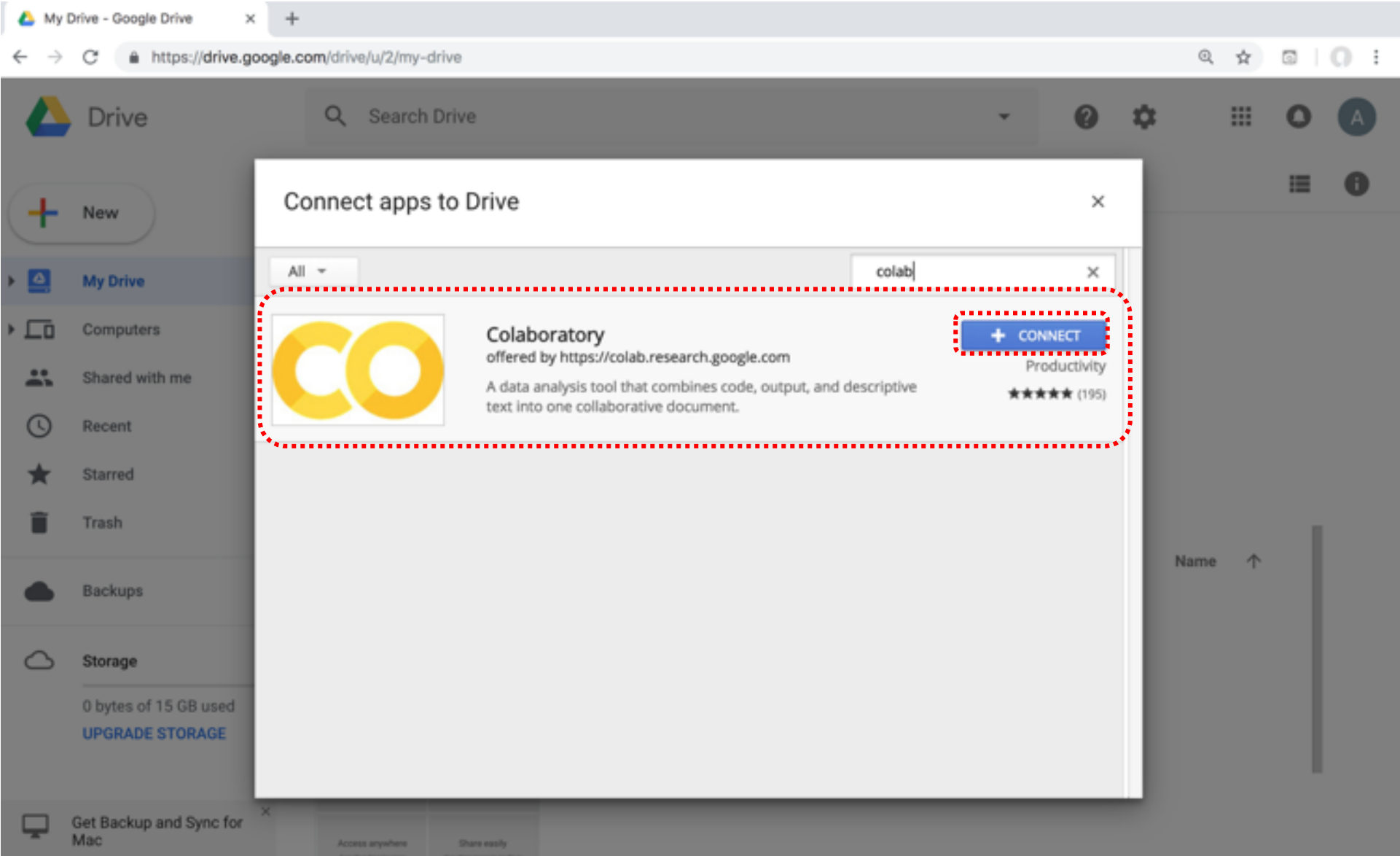


# Google Colab

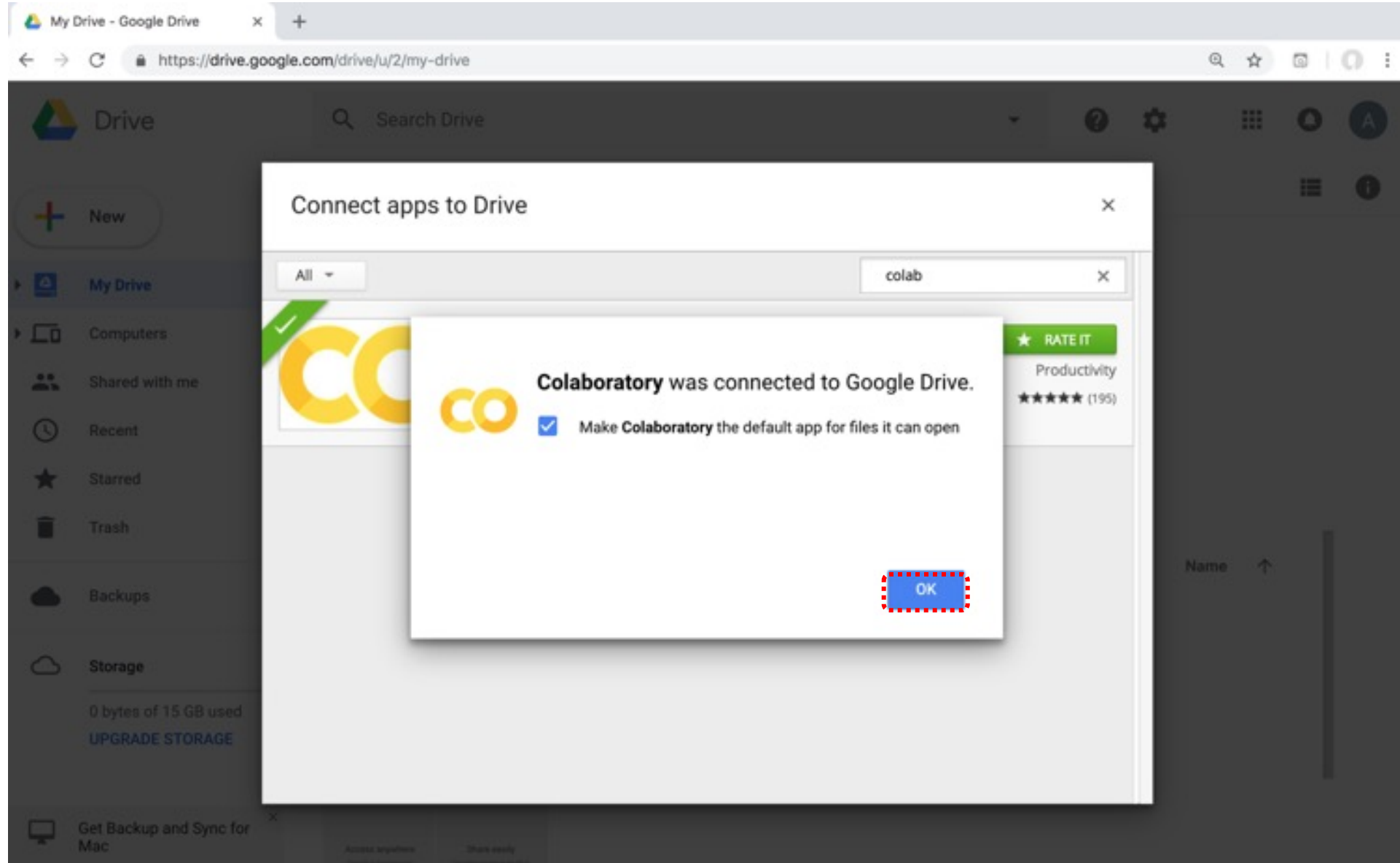
The screenshot shows the Google Drive web interface. A dialog box titled "Connect apps to Drive" is open, displaying a grid of application cards. The "colab" app card is highlighted with a red dashed border. The dialog box includes a search bar, a filter dropdown set to "All", and a close button. The background shows the Drive sidebar with options like "New", "My Drive", "Computers", "Shared with me", "Recent", "Starred", "Trash", "Backups", and "Storage".

App Name	Users
ZIP Extractor	307,585 users
Lumin PDF - Beautiful PDF Editor	289,310 users
CloudConvert	373,161 users
Sejda	1106 users (5 stars)
DocHub - Edit and Sign PDF Documents	2,131,600 users
Google Forms	4,803,614 users

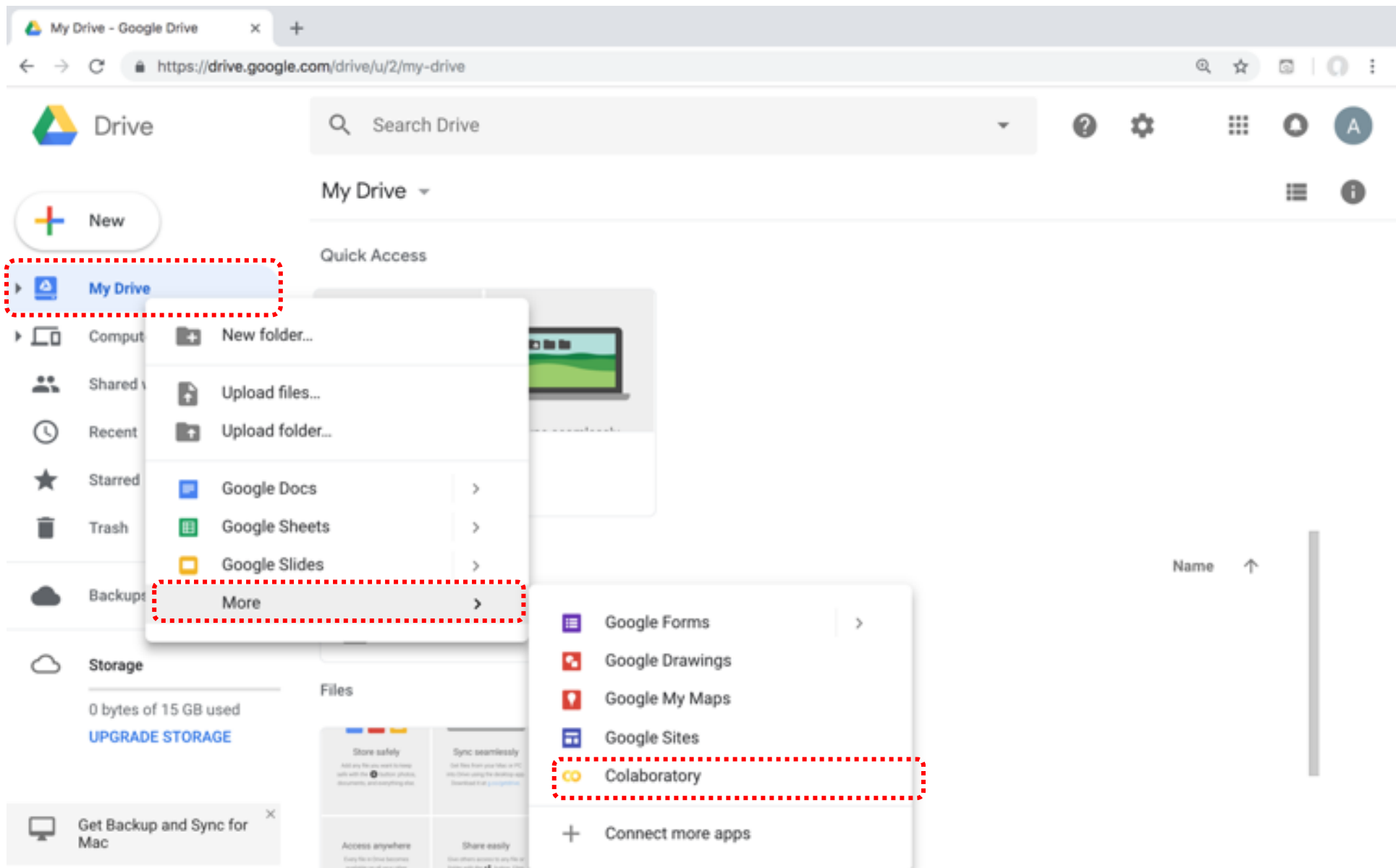
# Google Colab



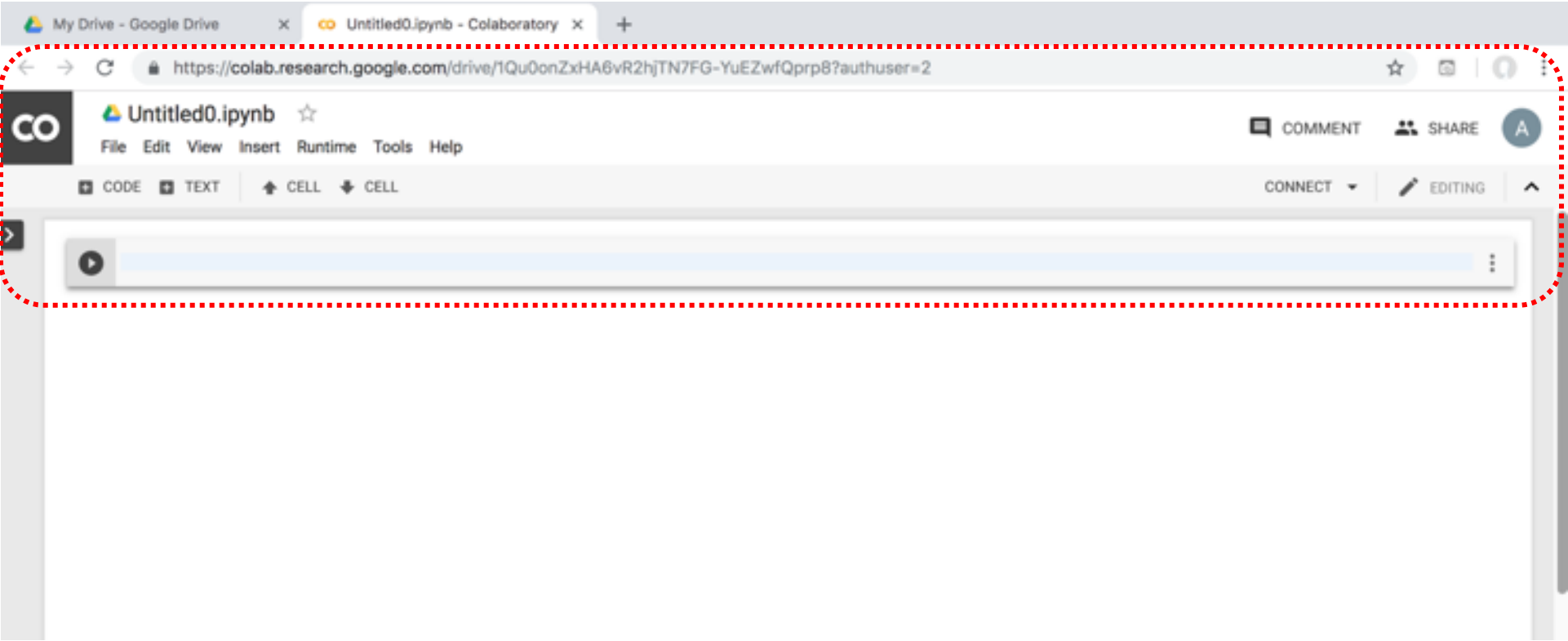
# Connect Colaboratory to Google Drive



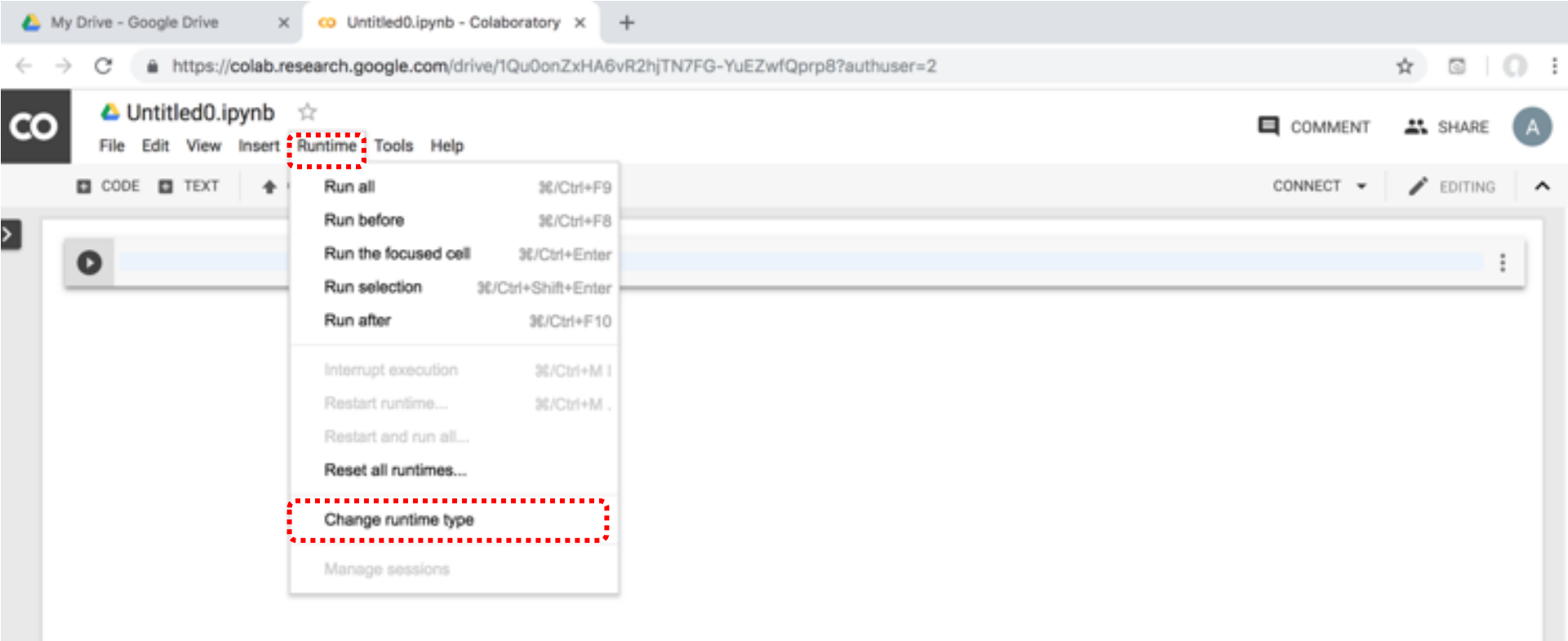
# Google Colab



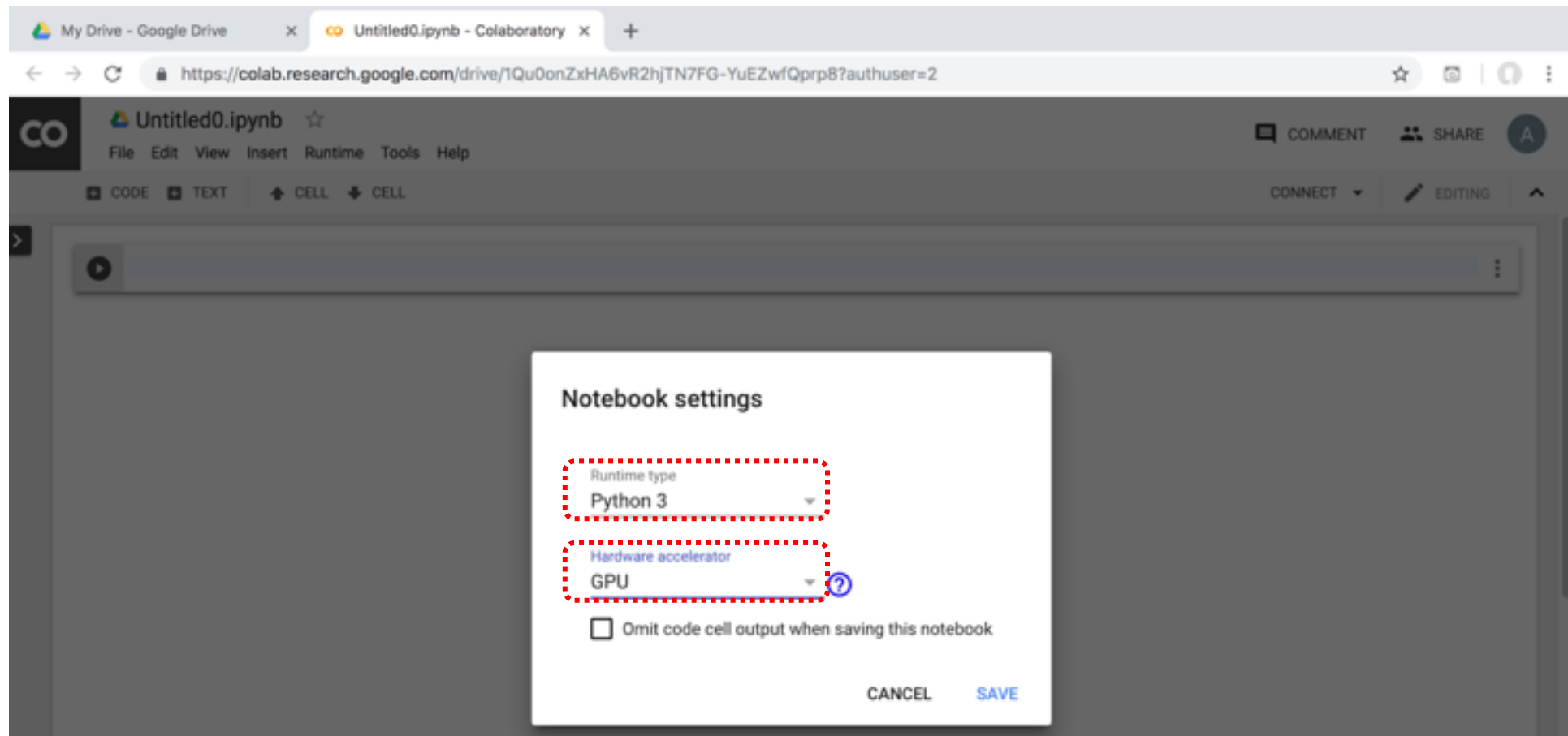
# Google Colab



# Google Colab



# Run Jupyter Notebook Python3 GPU Google Colab





# Google Colab Python Hello World

```
print('Hello World')
```





**Anaconda**  
**The Most Popular**  
**Python**  
**Data Science Platform**

# Download Anaconda



Products ▾

Pricing

Solutions ▾

Resources ▾

Partners ▾

Blog

Company ▾

Contact Sales

## Data science technology for a better world.

Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. Start working with thousands of open-source packages and libraries today.

Download 

Get Additional Installers



<https://www.anaconda.com/download>

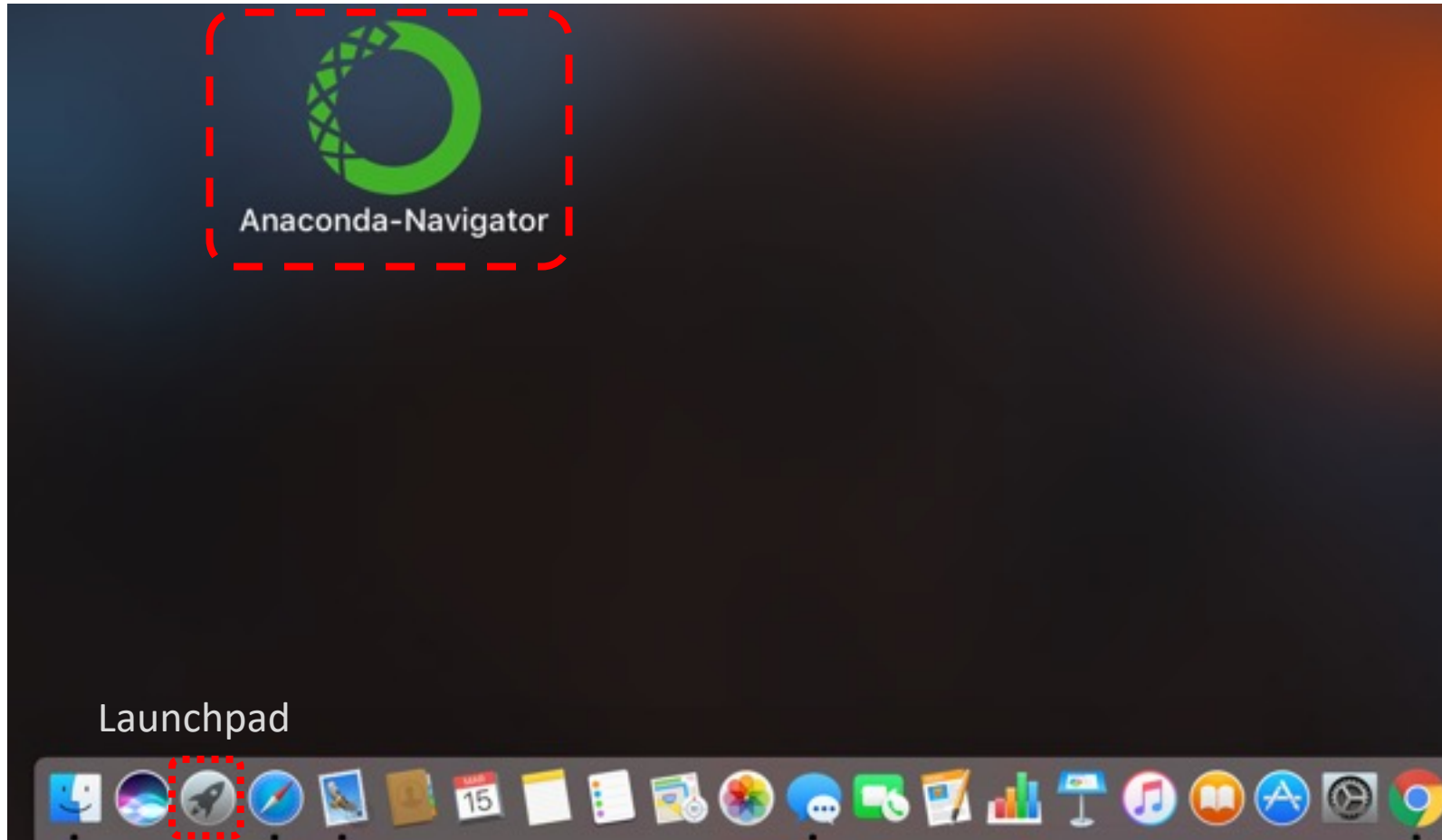




# Python

# HelloWorld

# Anaconda-Navigator



# Anaconda Navigator

The screenshot displays the Anaconda Navigator desktop application. At the top, the title bar reads "Anaconda Navigator". Below it, the application logo and name "ANACONDA NAVIGATOR" are on the left, and a "Sign in to Anaconda Cloud" button is on the right. A left-hand sidebar contains navigation options: Home, Environments, Learning, and Community. At the bottom of the sidebar are links for Documentation, Developer Blog, and Feedback, along with social media icons for Twitter, YouTube, and GitHub. The main content area is titled "Applications on" followed by a dropdown menu set to "base (root)" and a "Channels" button. A "Refresh" button is in the top right of this area. A grid of six application cards is shown. The "jupyter notebook" card is highlighted with a red dashed border and its "Launch" button is circled in red. The cards are: jupyterlab (0.31.5), jupyter notebook (5.4.0), qtconsole (4.3.1), spyder (3.2.6), vscode (1.22.2), and glueviz (0.12.4). Each card includes a description and a button to either "Launch" or "Install".

Applications on base (root) Channels Refresh

- jupyterlab** (0.31.5)  
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.  
Launch
- jupyter notebook** (5.4.0)  
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.  
Launch
- qtconsole** (4.3.1)  
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.  
Launch
- spyder** (3.2.6)  
Scientific PYTHON Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features  
Launch
- vscode** (1.22.2)  
Streamlined code editor with support for development operations like debugging, task running and version control.  
Launch
- glueviz** (0.12.4)  
Multidimensional data visualization across files. Explore relationships within and among related datasets.  
Install

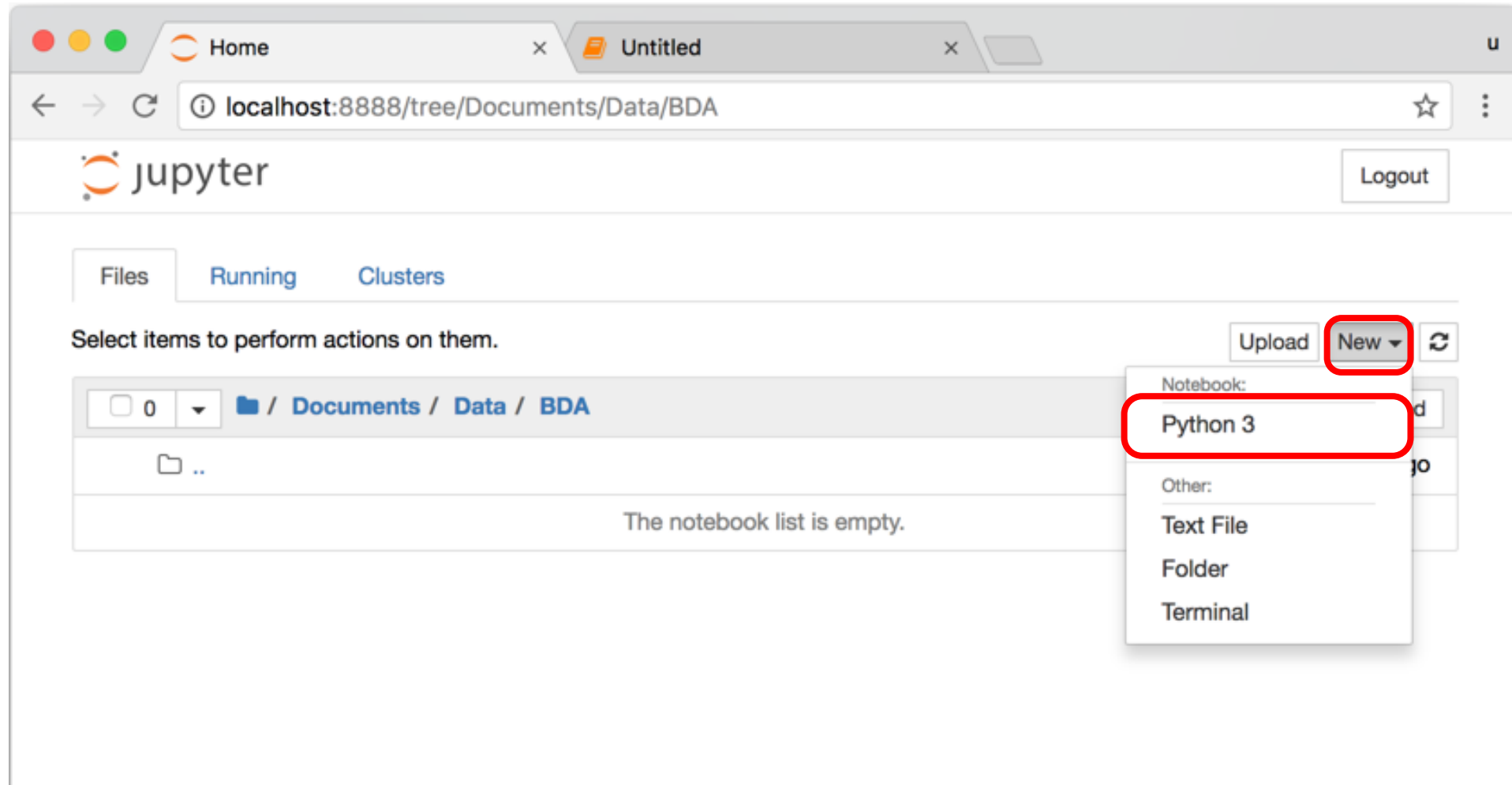
# Jupyter Notebook

The screenshot shows a web browser window with the URL `localhost:8888/tree/Documents/Data/BDA`. The Jupyter logo and a "Logout" button are visible at the top. Below the navigation tabs ("Files", "Running", "Clusters"), there are buttons for "Upload", "New", and a refresh icon. A message "Select items to perform actions on them." is displayed above a table. The table has a header row with "Name" and "Last Modified" columns. The first row shows a folder icon and the text `..` under the "Name" column, and "seconds ago" under the "Last Modified" column. Below the table, a message states "The notebook list is empty." A red dashed box highlights the table and the message below it.

Name	Last Modified
..	seconds ago

The notebook list is empty.

# Jupyter Notebook New Python 3





```
print("hello, world")
```

The screenshot shows a web browser window displaying a Jupyter Notebook. The browser's address bar shows the URL `localhost:8888/notebooks/Documents/Data/BDA/HelloWorld.ipynb`. The notebook interface includes a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar below the menu contains icons for file operations and a prominent **Run** button, which is circled in red. The notebook content area shows a code cell with the input `In [1]: print("hello, world")` and the output `hello, world`. The code cell itself is also circled in red. Below the first cell is an empty input field labeled `In [ ]:`. The notebook title is "HelloWorld (autosaved)" and the kernel is identified as "Python 3".



# Python

# Programming



# Python Fiddle

The screenshot shows the Python Fiddle web application in a browser. The browser's address bar displays 'pythonfiddle.com'. The application's header includes navigation buttons for 'Run', 'Reset', 'Share', 'Import', and 'Login', along with a 'Language' dropdown menu. The Python Fiddle logo and 'Python Cloud IDE' text are positioned on the right side of the header. A Google+ badge with '+1' and '2.6k' is located below the navigation buttons. The main workspace is divided into three sections: a left sidebar with a list of 'Examples' (including 'Chaining comparison operators', 'Decorators', 'Creating generators objects', 'Enumerate', 'Function closure', 'Lex tokenizer', 'Step argument in slice operators', 'For Else', 'Verbose regular expressions', 'In-place value swapping', and 'Function argument unpacking'), a central code editor with the text `1 print("Hello Python Fiddle")` and `2`, and a right sidebar with form fields for 'Title', 'Description', and 'Tags', and a 'Save' button.

Hello Python Fiddle

# Text input and output

```
print("Hello World")
```

```
print("Hello World\nThis is a message")
```

```
x = 3  
print(x)
```

```
x = 2  
y = 3  
print(x, ' ', y)
```

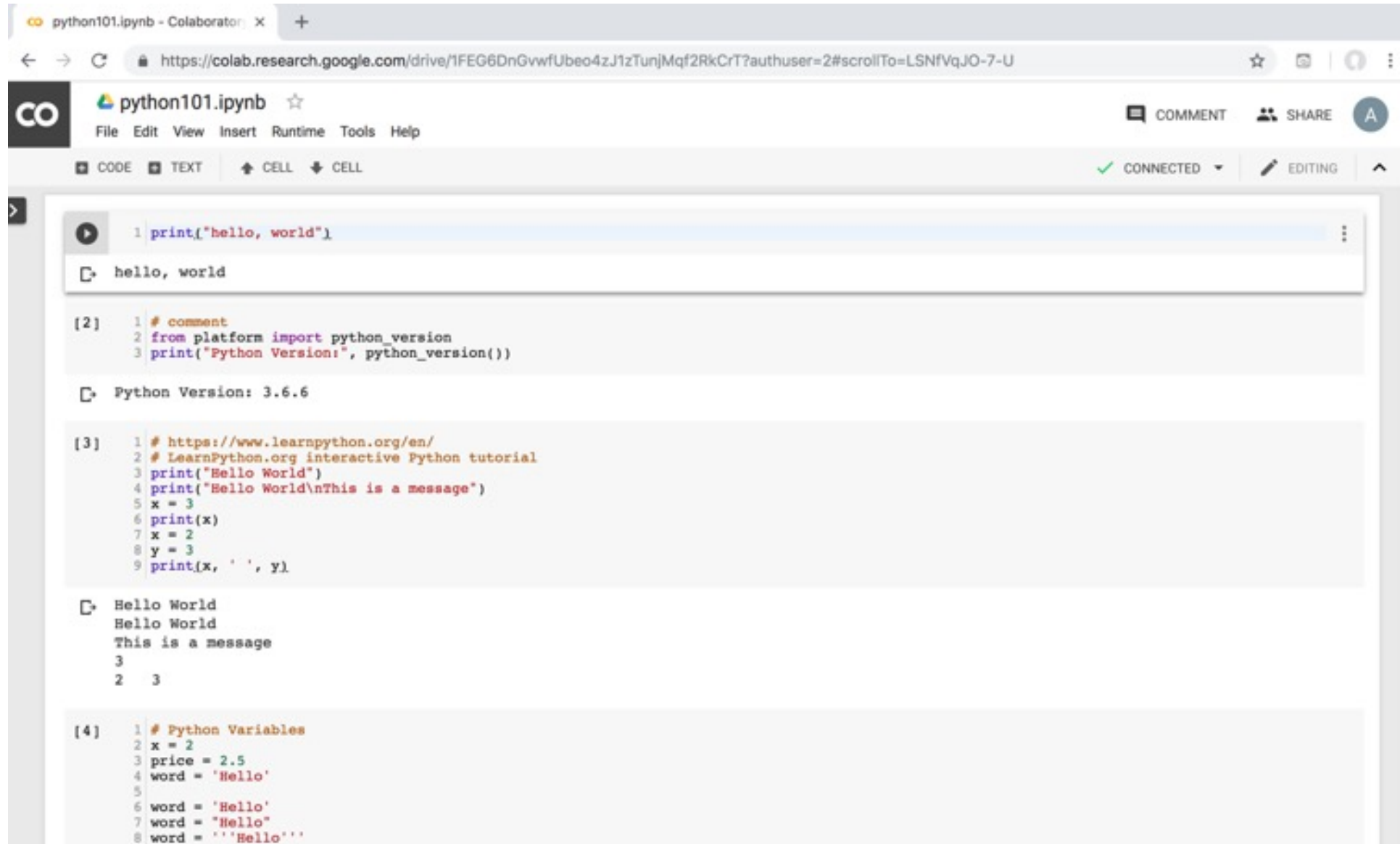
```
name = input("Enter a name: ")
```

```
x = int(input("What is x? "))
```

```
x = float(input("Write a number "))
```

# Python in Google Colab

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>



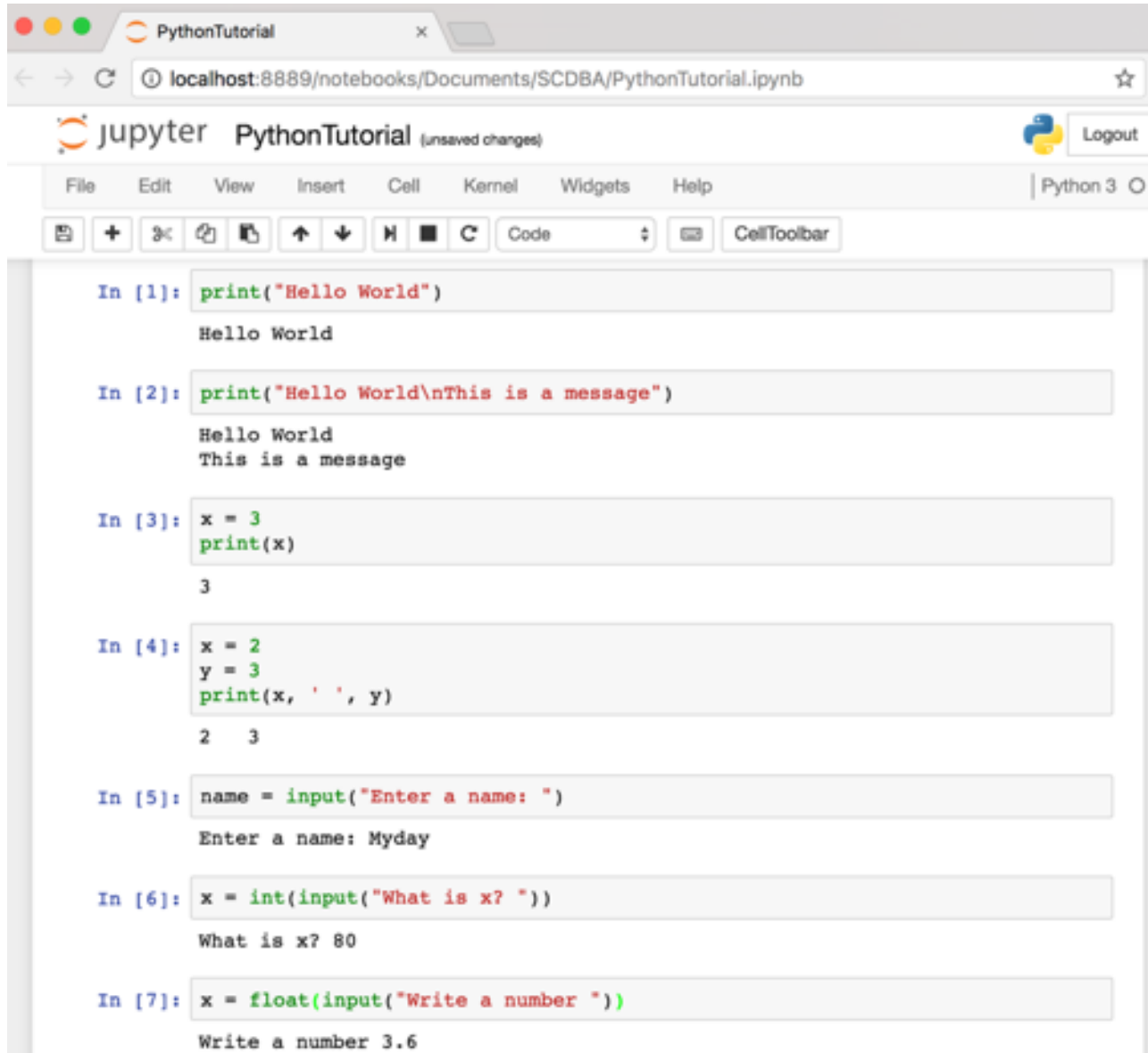
The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: <https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT?authuser=2#scrollTo=LSNfVqJO-7-U>. The notebook title is "python101.ipynb". The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with options for CODE, TEXT, CELL, and a status indicator showing "CONNECTED" and "EDITING".

The notebook contains four code cells:

- Cell 1:** `print("hello, world")`. Output: `hello, world`.
- Cell 2:** `# comment`, `from platform import python_version`, `print("Python Version:", python_version())`. Output: `Python Version: 3.6.6`.
- Cell 3:** `# https://www.learnpython.org/en/`, `# LearnPython.org interactive Python tutorial`, `print("Hello World")`, `print("Hello World\nThis is a message")`, `x = 3`, `print(x)`, `x = 2`, `y = 3`, `print(x, ' ', y)`. Output: `Hello World`, `Hello World`, `This is a message`, `3`, `2 3`.
- Cell 4:** `# Python Variables`, `x = 2`, `price = 2.5`, `word = 'Hello'`, `word = 'Hello'`, `word = "Hello"`, `word = '''Hello'''`.

<https://tinyurl.com/aintpupython101>

# Text input and output



The screenshot shows a Jupyter Notebook window titled "PythonTutorial" with the URL `localhost:8889/notebooks/Documents/SCDBA/PythonTutorial.ipynb`. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for adding, deleting, and running code. The notebook contains seven code cells, each with its input and output:

```
In [1]: print("Hello World")
Hello World

In [2]: print("Hello World\nThis is a message")
Hello World
This is a message

In [3]: x = 3
        print(x)
3

In [4]: x = 2
        y = 3
        print(x, ' ', y)
2 3

In [5]: name = input("Enter a name: ")
Enter a name: Myday

In [6]: x = int(input("What is x? "))
What is x? 80

In [7]: x = float(input("Write a number "))
Write a number 3.6
```

# Variables

```
x = 2  
price = 2.5  
word = 'Hello'
```

```
word = 'Hello'  
word = "Hello"  
word = '''Hello'''
```

```
x = 2  
x = x + 1  
x = 5
```



# Python Basic Operators

```
print('7 + 2 =', 7 + 2)
print('7 - 2 =', 7 - 2)
print('7 * 2 =', 7 * 2)
print('7 / 2 =', 7 / 2)
print('7 // 2 =', 7 // 2)
print('7 % 2 =', 7 % 2)
print('7 ** 2 =', 7 ** 2)
```

```
print('7 + 2 =', 7 + 2)
print('7 - 2 =', 7 - 2)
print('7 * 2 =', 7 * 2)
print('7 / 2 =', 7 / 2)
print('7 // 2 =', 7 // 2)
print('7 % 2 =', 7 % 2)
print('7 ** 2 =', 7 ** 2)
```

```
7 + 2 = 9
7 - 2 = 5
7 * 2 = 14
7 / 2 = 3.5
7 // 2 = 3
7 % 2 = 1
7 ** 2 = 49
```

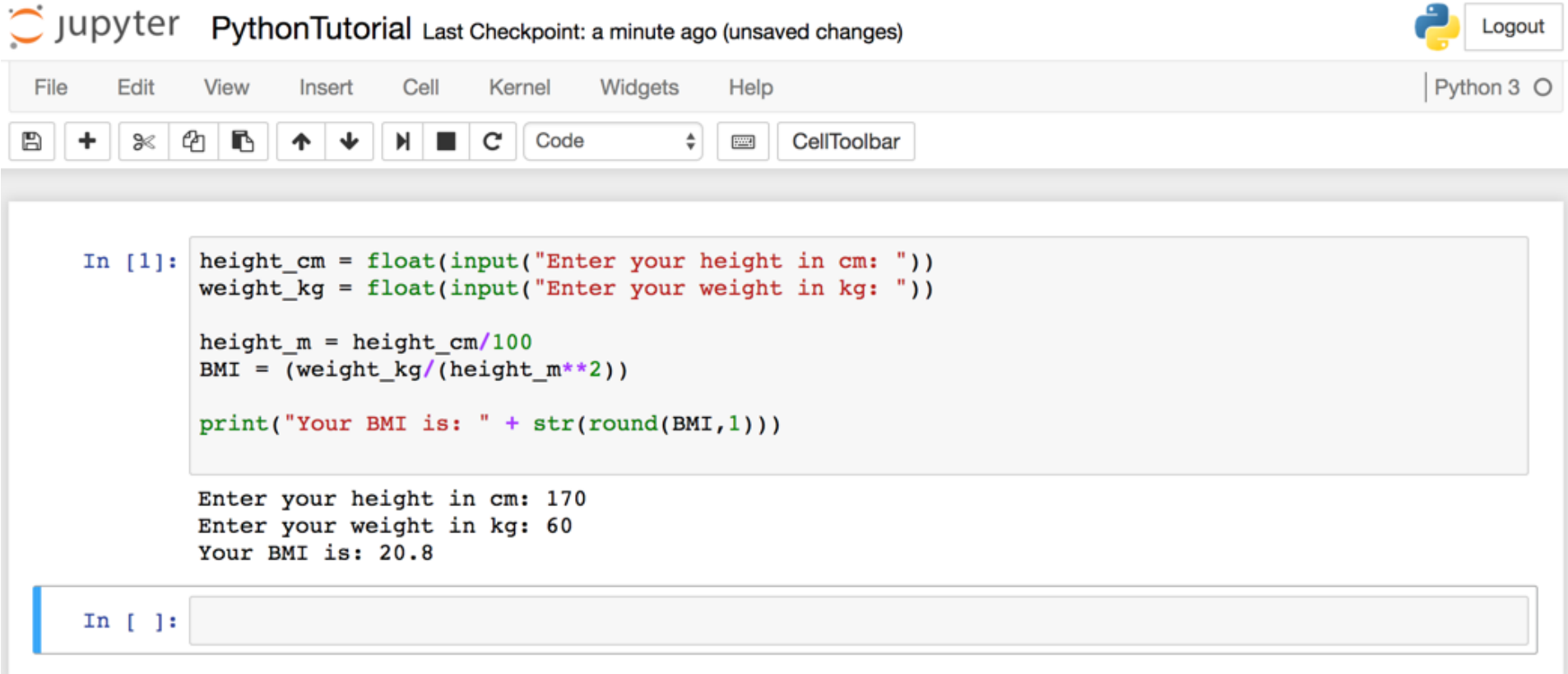
# BMI Calculator in Python

```
height_cm = float(input("Enter your height in cm: "))
weight_kg = float(input("Enter your weight in kg: "))

height_m = height_cm/100
BMI = (weight_kg/(height_m**2))

print("Your BMI is: " + str(round(BMI,1)))
```

# BMI Calculator in Python



The screenshot shows a Jupyter Notebook interface. At the top left, the Jupyter logo is followed by the text "jupyter PythonTutorial" and "Last Checkpoint: a minute ago (unsaved changes)". On the top right, there is a "Logout" button. Below the header is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. To the right of the menu bar, it says "Python 3". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and a dropdown menu set to "Code". To the right of the toolbar is a "CellToolbar" button. The main area of the notebook contains a code cell with the following Python code:

```
In [1]: height_cm = float(input("Enter your height in cm: "))
weight_kg = float(input("Enter your weight in kg: "))

height_m = height_cm/100
BMI = (weight_kg/(height_m**2))

print("Your BMI is: " + str(round(BMI,1)))
```

Below the code, the output of the code is displayed:

```
Enter your height in cm: 170
Enter your weight in kg: 60
Your BMI is: 20.8
```

At the bottom of the notebook, there is an empty code cell labeled "In [ ]:".

**Future value**  
**of a specified**  
**principal amount,**  
**rate of interest, and**  
**a number of years**

# Future Value (FV)

```
# How much is your $100 worth after 7 years?  
print(100 * 1.1 ** 7)  
# output = 194.87
```

```
print(100 * 1.1 ** 7)
```

```
194.87171000000012
```

# Future Value (FV)

```
pv = 100  
r = 0.1  
n = 7
```

```
fv = pv * ((1 + (r)) ** n)  
print(round(fv, 2))
```

```
pv = 100  
r = 0.1  
n = 7  
  
fv = pv * ((1 + (r)) ** n)  
print(round(fv, 2))
```

194.87

# Future Value (FV)

```
amount = 100
interest = 10 #10% = 0.01 * 10
years = 7

future_value = amount * ((1 + (0.01 * interest)) ** years)
print(round(future_value, 2))
```

```
amount = 100
interest = 10 #10% = 0.01 * 10
years = 7

future_value = amount * ((1 + (0.01 * interest)) ** years)
print(round(future_value, 2))
```

194.87

# if statements

> greater than  
< smaller than  
== equals  
!= is not

```
score = 80
if score >=60 :
    print("Pass")
else:
    print("Fail")
```

Pass

```
score = 80
if score >=60 :
    print("Pass")
else:
    print("Fail")
```



# if elif else

```
score = 90
grade = ""
if score >=90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "E"
print(grade)
# grade = "A"
```

A

<http://pythontutor.com/visualize.html>  
<https://goo.gl/E6w5ph>

# for loops

```
for i in range(1,11):  
    print(i)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

# for loops

```
for i in range(1,10):  
    for j in range(1,10):  
        print(i, ' * ', j, ' = ', i*j)
```

```
9 * 1 = 9  
9 * 2 = 18  
9 * 3 = 27  
9 * 4 = 36  
9 * 5 = 45  
9 * 6 = 54  
9 * 7 = 63  
9 * 8 = 72  
9 * 9 = 81
```

# while loops

```
age = 10

while age < 20:
    print(age)
    age = age + 1
```

```
10
11
12
13
14
15
16
17
18
19
```

# def Functions

```
def convertCMtoM(xcm) :  
    m = xcm/100  
    return m
```

```
cm = 180  
m = convertCMtoM(cm)  
print(str(m))
```

1.8

# Lists []

```
x = [60, 70, 80, 90]
print(len(x))
print(x[0])
print(x[1])
print(x[-1])
```

4

60

70

90

# Tuples ()

A **tuple** in Python is a collection that **cannot be modified**.

A tuple is defined using **parenthesis**.

```
x = (10, 20, 30, 40, 50)
print(x[0])
print(x[1])
print(x[2])
print(x[-1])
```

```
10
20
30
50
```

# Dictionary {key : value}

```
k = { 'EN' : 'English' , 'FR' : 'French' }  
print(k[ 'EN' ])
```

## Dictionary

'EN' → 'English'

'FR' → 'French'

English



# Sets {}

```
animals = {'cat', 'dog'}
```

```
animals = {'cat', 'dog'}
print('cat' in animals) # Check if an element is in a set; prints "True"
print('fish' in animals) # prints "False"
animals.add('fish') # Add an element to a set
print('fish' in animals) # Prints "True"
print(len(animals)) # Number of elements in a set; prints "3"
animals.add('cat') # Adding an element that is already in the set does nothing
print(len(animals)) # Prints "3"
animals.remove('cat') # Remove an element from a set
print(len(animals)) # Prints "2"
```

```
True
False
True
3
3
2
```

```
animals = {'cat', 'dog'}
print('cat' in animals)
print('fish' in animals)
animals.add('fish')
print('fish' in animals)
print(len(animals))
animals.add('cat')
print(len(animals))
animals.remove('cat')
print(len(animals))
```

# File Input / Output

```
with open('myfile.txt', 'w') as file:  
    file.write('Hello World\nThis is Python File Input Output')  
  
with open('myfile.txt', 'r') as file:  
    text = file.read()  
print(text)
```

```
with open('myfile.txt', 'w') as file:  
    file.write('Hello World\nThis is Python File Input Output')
```

```
with open('myfile.txt', 'r') as file:  
    text = file.read()  
print(text)
```

```
Hello World  
This is Python File Input Output
```

```
text
```

```
'Hello World\nThis is Python File Input Output'
```

# File Input / Output

```
with open('myfile.txt', 'a+') as file:  
    file.write('\n' + 'New line')
```

```
with open('myfile.txt', 'r') as file:  
    text = file.read()  
print(text)
```

```
with open('myfile.txt', 'a+') as file:  
    file.write('\n' + 'New line')
```

```
with open('myfile.txt', 'r') as file:  
    text = file.read()  
print(text)
```

```
Hello World  
This is Python File Input Output  
New line
```

# try except finally

```
try:
    file = open("myfile.txt")
    #file = open("myfile.txt", 'w')
    file.write("Python write file")
    print("file saved")
except:
    print("Exception file Error")
finally:
    file.close()
    print("finally process")
```

**Exception file Error**  
**finally process**

# class

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("Alan", 20)
p1.myfunc()
print(p1.name)
print(p1.age)
```

```
Hello my name is Alan
Alan
20
```

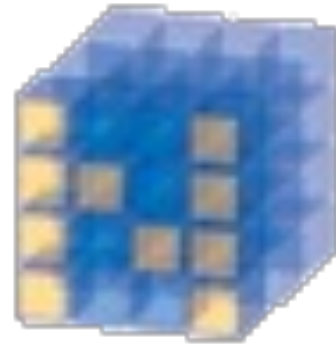
**Big Data Analytics**

**with**

**Numpy**

**in Python**

# NumPy



NumPy

Base

**N-dimensional array**

package

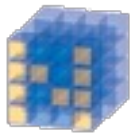
**NumPy**  
is the  
**fundamental package**  
for  
**scientific computing**  
with **Python.**





# NumPy

- **NumPy** provides a **multidimensional array** object to store homogenous or heterogeneous data; it also provides **optimized functions/methods** to operate on this array object.



NumPy

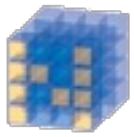
# NumPy ndarray

## One-dimensional Array (1-D Array)

0	1			n-1
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>

## Two-dimensional Array (2-D Array)

	0	1			n-1
0	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
1	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>
m-1	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>



NumPy

# NumPy

```
v = list(range(1, 6))
```

```
v
```

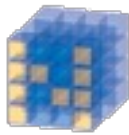
```
2 * v
```

```
import numpy as np
```

```
v = np.arange(1, 6)
```

```
v
```

```
2 * v
```



## NumPy

Base  
N-dimensional  
array package

```
1 v = list(range(1, 6))  
2 v
```

```
[1, 2, 3, 4, 5]
```

```
1 2 * v
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

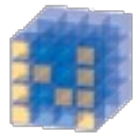
```
1 import numpy as np  
2 v = np.arange(1, 6)  
3 v
```

```
array([1, 2, 3, 4, 5])
```

```
1 2 * v
```

```
array([ 2,  4,  6,  8, 10])
```

---



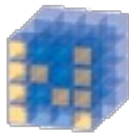
NumPy

# NumPy Create Array

```
import numpy as np  
a = np.array([1, 2, 3])  
b = np.array([4, 5, 6])  
c = a * b  
c
```

```
import numpy as np  
a = np.array([1, 2, 3])  
b = np.array([4, 5, 6])  
c = a * b  
c
```

```
array([ 4, 10, 18])
```



NumPy

# NumPy

```
1 import numpy as np
2
3 a = np.zeros((2,2)) # Create an array of all zeros
4 print(a)           # Prints "[[ 0.  0.]
5                     #           [ 0.  0.]]"
6
7 b = np.ones((1,2)) # Create an array of all ones
8 print(b)           # Prints "[[ 1.  1.]]"
9
10 c = np.full((2,2), 7) # Create a constant array
11 print(c)            # Prints "[[ 7.  7.]
12                    #           [ 7.  7.]]"
13
14 d = np.eye(2)       # Create a 2x2 identity matrix
15 print(d)           # Prints "[[ 1.  0.]
16                    #           [ 0.  1.]]"
17
18 e = np.random.random((2,2)) # Create an array filled with random values
19 print(e)            # Might print "[[ 0.91940167  0.08143941]
20                    #           [ 0.68744134  0.87236687]]"
```

```
[[0.  0.]
 [0.  0.]]
[[1.  1.]]
[[7 7]
 [7 7]]
[[1.  0.]
 [0.  1.]]
[[0.66258211 0.65552598]
 [0.00429934 0.21695824]]
```

```
import numpy as np  
a = np.arange(15).reshape(3, 5)
```

**a.shape**

**a.ndim**

**a.dtype.name**

```
import numpy as np  
a = np.arange(15).reshape(3, 5)  
a
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
print(a.shape)
```

```
(3, 5)
```

```
a.ndim
```

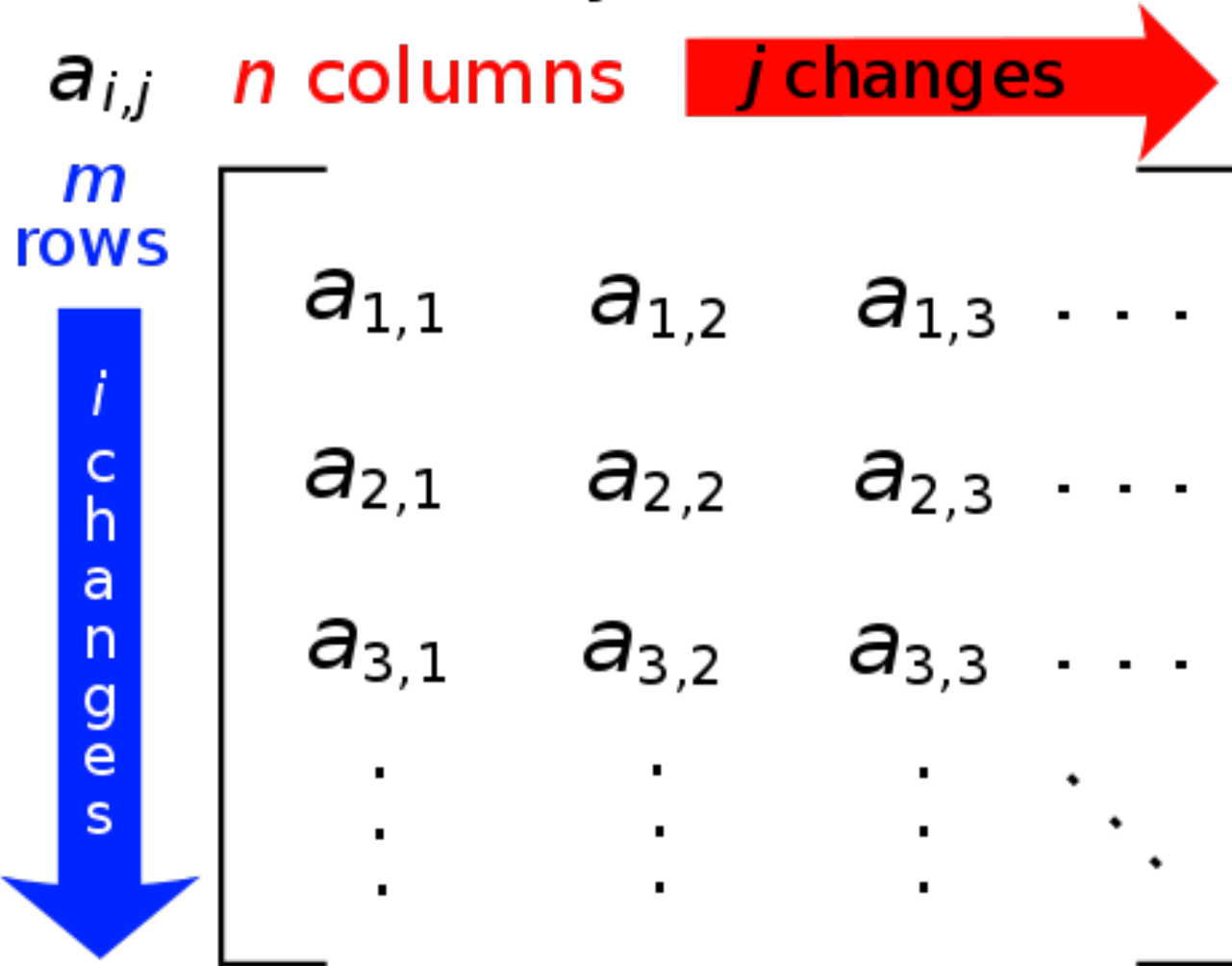
```
2
```

```
a.dtype.name
```

```
'int64'
```

# Matrix

$m$ -by- $n$  matrix





# **NumPy ndarray: Multidimensional Array Object**

# NumPy ndarray

## One-dimensional Array (1-D Array)

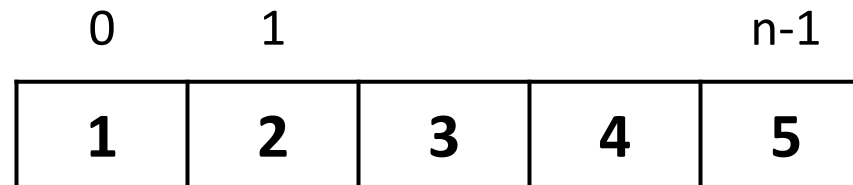
0	1			n-1
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>

## Two-dimensional Array (2-D Array)

	0	1		n-1	
0	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
1	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>
m-1	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>

```
import numpy as np
a = np.array([1,2,3,4,5])
```

## One-dimensional Array (1-D Array)



```
a = np.array([1,2,3,4,5])
a
```

```
array([1, 2, 3, 4, 5])
```

```
a = np.array([ [1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20] ] )
```

## Two-dimensional Array (2-D Array)

	0	1		n-1	
0	1	2	3	4	5
1	6	7	8	9	10
	11	12	13	14	15
m-1	16	17	18	19	20

```
a = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20]])  
a
```

```
array([[ 1,  2,  3,  4,  5],  
       [ 6,  7,  8,  9, 10],  
       [11, 12, 13, 14, 15],  
       [16, 17, 18, 19, 20]])
```

```
import numpy as np
a = np.array([[0, 1, 2, 3],
              [10, 11, 12, 13],
              [20, 21, 22, 23]])
a
```

0	1	2	3
10	11	12	13
20	21	22	23

```
a = np.array ([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]])
```

```
a = np.array([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]])  
a
```

```
array([[ 0,  1,  2,  3],  
       [10, 11, 12, 13],  
       [20, 21, 22, 23]])
```

```
print(a.ndim)
```

```
2
```

```
print(a.shape)
```

```
(3, 4)
```

0	1	2	3
10	11	12	13
20	21	22	23

# NumPy Basics: Arrays and Vectorized Computation

# NumPy Array

**axis 1**

**0**

**1**

**2**

**0**

**0,0**

**0,1**

**0,2**

**axis 0**

**1**

**1,0**

**1,1**

**1,2**

**2**

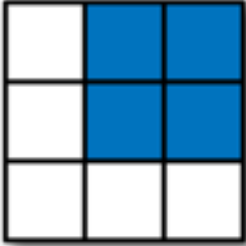

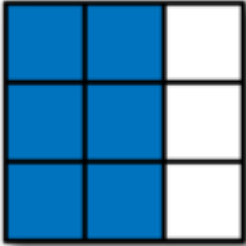
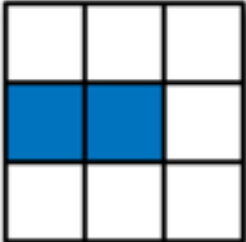
**2,0**

**2,1**

**2,2**



# Numpy Array

	Expression	Shape
	<code>arr[:2, 1:]</code>	<code>(2, 2)</code>
	<code>arr[2]</code> <code>arr[2, :]</code> <code>arr[2:, :]</code>	<code>(3,)</code> <code>(3,)</code> <code>(1, 3)</code>
	<code>arr[:, :2]</code>	<code>(3, 2)</code>
	<code>arr[1, :2]</code> <code>arr[1:2, :2]</code>	<code>(2,)</code> <code>(1, 2)</code>



# Tensor

- 3
  - a rank 0 tensor; this is a **scalar** with shape []
- [1., 2., 3.]
  - a rank 1 tensor; this is a **vector** with shape [3]
- [[1., 2., 3.], [4., 5., 6.]]
  - a rank 2 tensor; a **matrix** with shape [2, 3]
- [[[1., 2., 3.], [7., 8., 9.]]]
  - a rank 3 **tensor** with shape [2, 1, 3]

**Scalar**

80

**Vector**

[50 60 70]

**Matrix**

$$\begin{bmatrix} 50 & 60 & 70 \\ 55 & 65 & 75 \end{bmatrix}$$

**Tensor**

$$\begin{bmatrix} [50 & 60 & 70] & [70 & 80 & 90] \\ [55 & 65 & 75] & [75 & 85 & 95] \end{bmatrix}$$

# pandas

## Python Data Analysis Library

providing high-performance, easy-to-use  
**data structures and data analysis tools**  
for the Python programming language.

# pandas:

## powerful Python data analysis toolkit

- **Tabular data** with **heterogeneously-typed columns**, as in an **SQL table** or **Excel spreadsheet**
- **Ordered and unordered (not necessarily fixed-frequency) time series data.**
- **Arbitrary matrix data** (homogeneously typed or heterogeneous) **with row and column labels**
- **Any other form of observational / statistical data sets.** The data actually need not be labeled at all to be placed into a pandas data structure

# Series

# DataFrame

- **Primary data structures of pandas**
  - **Series (1-dimensional)**
  - **DataFrame (2-dimensional)**
- **Handle the vast majority of typical use cases in **finance**, statistics, social science, and many areas of engineering.**

# pandas DataFrame

- **DataFrame** provides everything that R's `data.frame` provides and much more.
- pandas is built on top of **NumPy** and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

# pandas

## Comparison with SAS

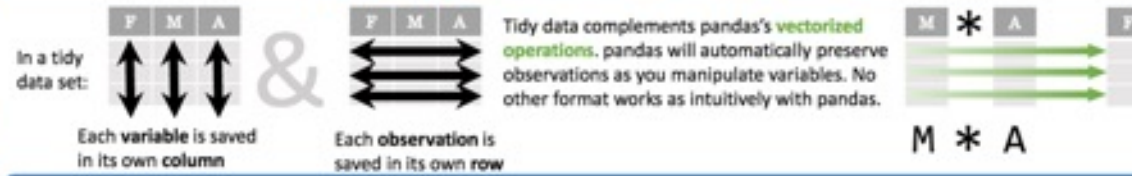
pandas	SAS
DataFrame	data set
column	variable
row	observation
groupby	BY-group
NaN	.



# Python Pandas Cheat Sheet

Data Wrangling  
with pandas  
Cheat Sheet  
<http://pandas.pydata.org>

## Tidy Data – A foundation for wrangling in pandas



## Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
    {"a": [4, 5, 6],
     "b": [7, 8, 9],
     "c": [10, 11, 12]},
    index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

n	v	a	b	c
1	4	7	10	
2	5	8	11	
3	6	9	12	

```
df = pd.DataFrame(
    {"a": [4, 5, 6],
     "b": [7, 8, 9],
     "c": [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2), ('e', 2)],
        names=['n', 'v']))
```

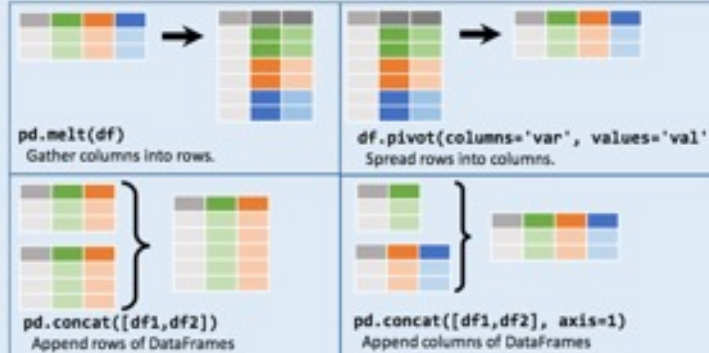
Create DataFrame with a MultiIndex

## Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
      .rename(columns={
          'variable': 'var',
          'value': 'val'})
      .query('val >= 200'))
```

## Reshaping Data – Change the layout of a data set



```
df=df.sort_values('mpg')
Order rows by values of a column (low to high).

df=df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

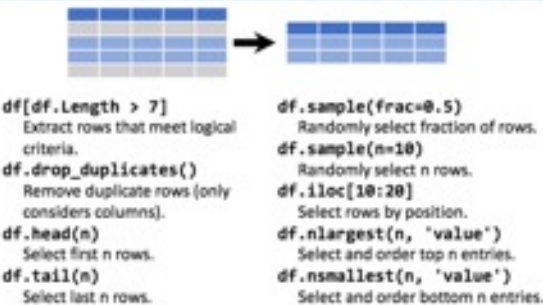
df=df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df=df.sort_index()
Sort the index of a DataFrame

df=df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df=df.drop(['Length', 'Height'], axis=1)
Drop columns from DataFrame
```

## Subset Observations (Rows)



## Subset Variables (Columns)



## Logic in Python (and pandas)

Operator	Logic	Operator	Logic
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is Null
<=	Less than or equals	pd.notnull(obj)	Is not Null
>=	Greater than or equals	&,  , ~, ^, df.any(), df.all()	Logical and, or, not, xor, any, all

<http://pandas.pydata.org/> This cheat sheet inspired by RStudio Data Wrangling Cheatsheet (<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>) Written by its using <https://www.rstudio.com/>

# Creating pd.DataFrame

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
In [1]: import numpy as np
import pandas as pd
df = pd.DataFrame({"a": [4, 5, 6],
                  "b": [7, 8, 9],
                  "c": [10, 11, 12]},
                  index = [1, 2, 3])
```

df

Out[1]:

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
import pandas as pd
df = pd.DataFrame({"a": [4, 5, 6],
                  "b": [7, 8, 9],
                  "c": [10, 11, 12]},
                  index = [1, 2, 3])
```

# Pandas DataFrame

```
type(df)
```

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
print('pandas imported')
```

```
s = pd.Series([1,3,5,np.nan,6,8])
s
```

```
dates = pd.date_range('20181001',
periods=6)
dates
```

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 print('pandas imported')
```

pandas imported

```
1 s = pd.Series([1,3,5,np.nan,6,8]).
2 s
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

```
1 dates = pd.date_range('20181001', periods=6)
2 dates
```

```
DatetimeIndex(['2018-10-01', '2018-10-02', '2018-10-03', '2018-10-04',
               '2018-10-05', '2018-10-06'],
              dtype='datetime64[ns]', freq='D')
```

```
df = pd.DataFrame(np.random.randn(6,4),  
index=dates, columns=list('ABCD'))  
df
```

```
1 df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))  
2 df
```

	A	B	C	D
2018-10-01	-0.336188	0.584621	-1.061433	-0.036278
2018-10-02	0.903683	-0.839723	-0.270219	-1.099606
2018-10-03	0.920208	-0.240353	-0.818598	-1.105489
2018-10-04	0.221045	-0.314589	0.042071	-1.447280
2018-10-05	0.946862	-1.570305	-1.009180	-0.375659
2018-10-06	-0.225148	0.510691	2.002372	-0.335005

```
df = pd.DataFrame(np.random.randn(3,5),  
index=['student1', 'student2', 'student3'],  
columns=list('ABCDE'))  
df
```

```
1 df = pd.DataFrame(np.random.randn(3,5), index=['student1', 'student2', 'student3'], columns=list('ABCDE'))  
2 df
```

	A	B	C	D	E
student1	-0.346884	-1.232934	-0.302072	-1.345084	-0.723880
student2	1.090955	-0.010483	1.280072	-0.253958	-0.030604
student3	0.325660	0.808956	-0.395820	-1.498926	1.603471

```
df2 = pd.DataFrame({ 'A' : 1.,  
                    'B' : pd.Timestamp('20181001'),  
                    'C' : pd.Series(2.5,index=list(range(4)),dtype='float32'),  
                    'D' : np.array([3] * 4,dtype='int32'),  
                    'E' : pd.Categorical(["test","train","test","train"]),  
                    'F' : 'foo' })  
df2
```

```
1 df2 = pd.DataFrame({ 'A' : 1.,  
2 'B' : pd.Timestamp('20181001'),  
3 'C' : pd.Series(2.5,index=list(range(4)),dtype='float32'),  
4 'D' : np.array([3] * 4,dtype='int32'),  
5 'E' : pd.Categorical(["test","train","test","train"]),  
6 'F' : 'foo' })  
7 df2
```

	A	B	C	D	E	F
0	1.0	2018-10-01	2.5	3	test	foo
1	1.0	2018-10-01	2.5	3	train	foo
2	1.0	2018-10-01	2.5	3	test	foo
3	1.0	2018-10-01	2.5	3	train	foo



# df2.dtypes

```
df2.dtypes
```

```
A          float64  
B    datetime64[ns]  
C          float32  
D          int32  
E          category  
F          object  
dtype: object
```

# Python Data Analysis and Visualization



# Python

# Pandas



**Python**  
**matplotlib**  
**matplotlib**

# Python

# seaborn



seaborn

# Python

# plotly



# Python

# bokeh

bokeh

# Python

# Altair



# Altair



# Python matplotlib



Fork me on GitHub

[Installation](#) [Documentation](#) [Examples](#) [Tutorials](#) [Contributing](#)

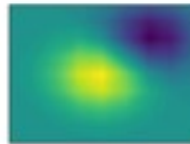
Search

[home](#) | [contents](#) » [Matplotlib: Python plotting](#)

[modules](#) | [index](#)

## Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.



Matplotlib makes easy things easy and hard things possible.

### Create

- Develop **publication quality plots** with just a few lines of code
- Use **interactive figures** that can zoom, pan, update...

### Customize

- **Take full control** of line styles, font properties, axes properties...
- **Export and embed** to a number of file formats and interactive environments

### Extend

- Explore tailored functionality provided by **third party packages**
- Learn more about Matplotlib through the many **external learning resources**

#### Latest stable release

3.3.4: [docs](#) | [changelog](#)

#### Last release for Python 2

2.2.5: [docs](#) | [changelog](#)

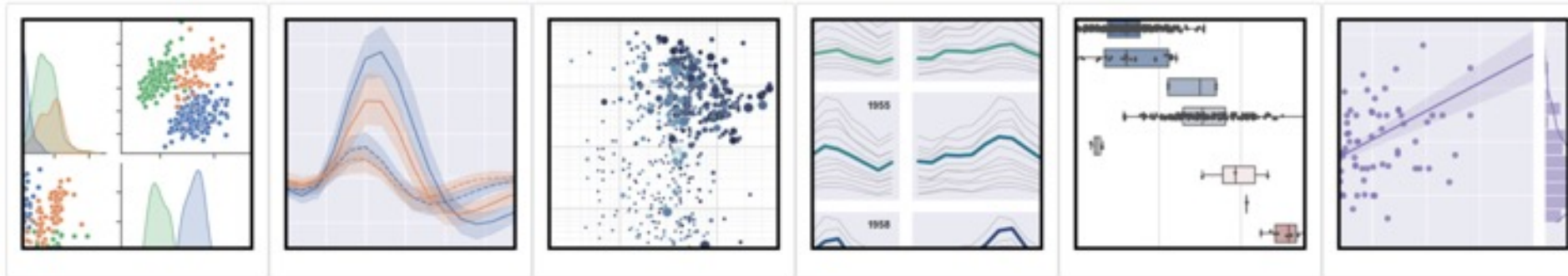
#### Development version

[docs](#)

Support Matplotlib



## seaborn: statistical data visualization



Seaborn is a Python data visualization library based on [matplotlib](#). It provides a high-level interface for drawing attractive and informative statistical graphics.

For a brief introduction to the ideas behind the library, you can read the [introductory notes](#). Visit the [installation page](#) to see how you can download the package and get started with it. You can browse the [example gallery](#) to see what you can do with seaborn, and then check out the [tutorial](#) and [API reference](#) to find out how.

To see the code or report a bug, please visit the [GitHub repository](#). General support questions are most at home on [stackoverflow](#) or [discourse](#), which have dedicated channels for seaborn.

### Contents

- [Introduction](#)
- [Release notes](#)
- [Installing](#)
- [Example gallery](#)
- [Tutorial](#)
- [API reference](#)

### Features

- [Relational: API | Tutorial](#)
- [Distribution: API | Tutorial](#)
- [Categorical: API | Tutorial](#)
- [Regression: API | Tutorial](#)
- [Multiples: API | Tutorial](#)
- [Style: API | Tutorial](#)
- [Color: API | Tutorial](#)

# Python Plotly Graphing Library

## Quick Start

Getting Started

Is Plotly Free?

Figure Reference

API Reference

Dash

GitHub

community.plotly.com

## Examples

Fundamentals

Basic Charts

Statistical Charts

Artificial Intelligence and Machine Learning

Scientific Charts

Financial Charts

Maps

3D Charts

## Plotly Python Open Source Graphing Library

Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts.

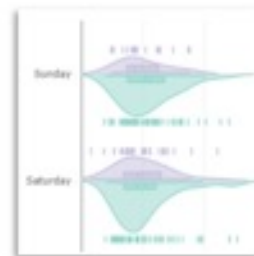
Plotly.py is [free and open source](#) and you can [view the source](#), [report issues](#) or [contribute on GitHub](#).

Our recommended IDE for Plotly's Python graphing library is Dash Enterprise's [Data Science Workspaces](#), which has both Jupyter notebook and Python code file support. [Find out if your company is using Dash Enterprise.](#)

[Install Dash Enterprise on Azure](#) | [Install Dash Enterprise on AWS](#)

## Fundamentals

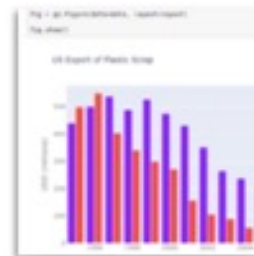
[More Fundamentals -](#)



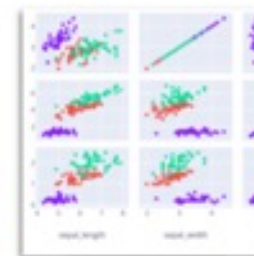
The Figure Data Structure



Creating and Updating Figures



Displaying Figures



Plotly Express

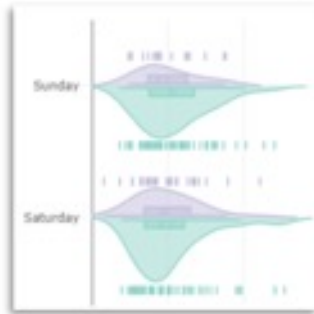


Analytical Apps with Dash

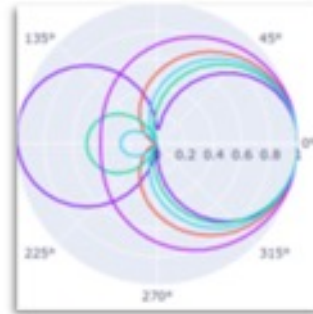
# Python Plotly Graphing Library

## Fundamentals

[More Fundamentals »](#)



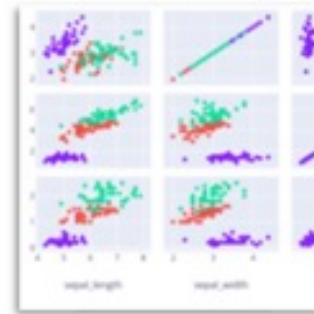
The Figure Data Structure



Creating and Updating Figures



Displaying Figures



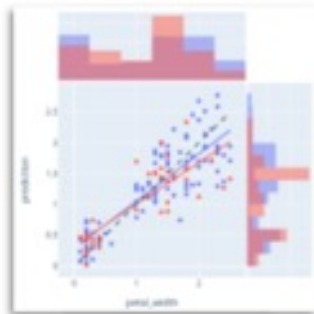
Plotly Express



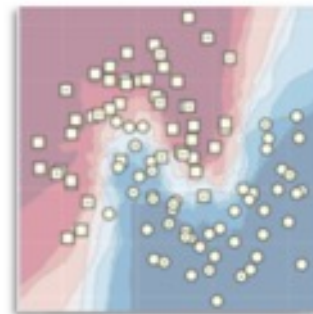
Analytical Apps with Dash

## Artificial Intelligence and Machine Learning

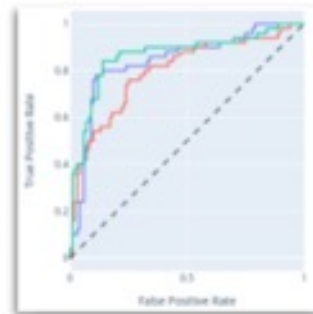
[More AI and ML »](#)



ML Regression



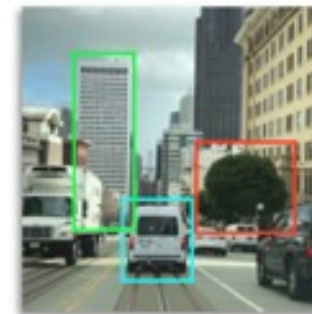
kNN Classification



ROC and PR Curves



PCA Visualization

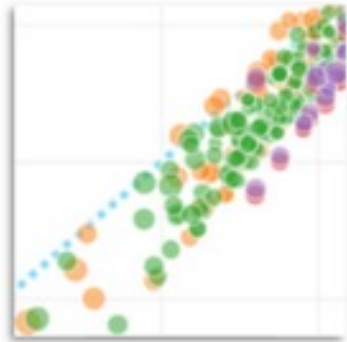


AI/ML Apps with Dash

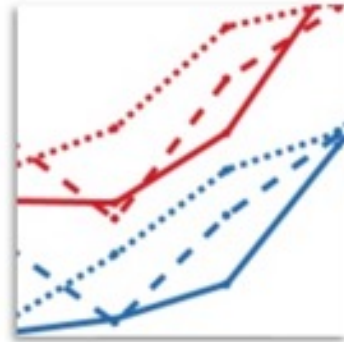
# Python Plotly Graphing Library

## Basic Charts

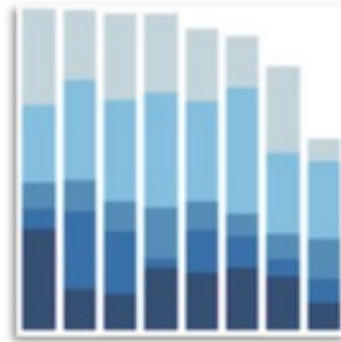
[More Basic Charts >](#)



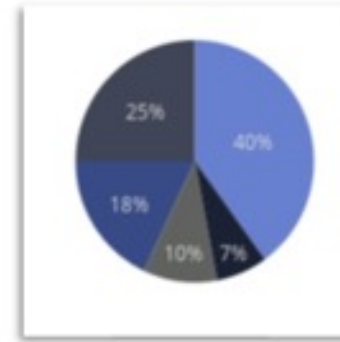
Scatter Plots



Line Charts



Bar Charts



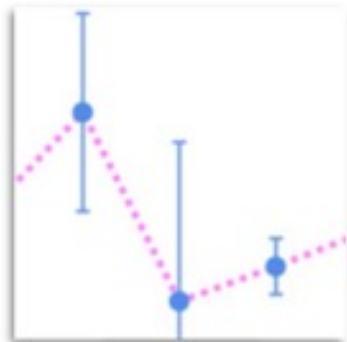
Pie Charts



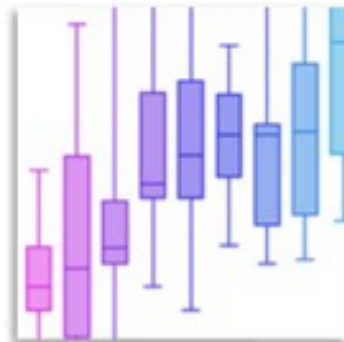
Bubble Charts

## Statistical Charts

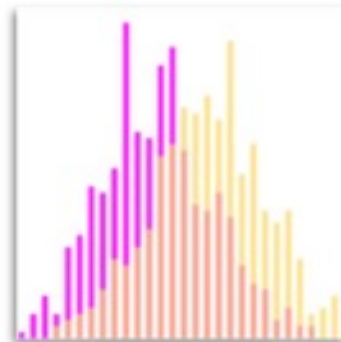
[More Statistical Charts >](#)



Error Bars



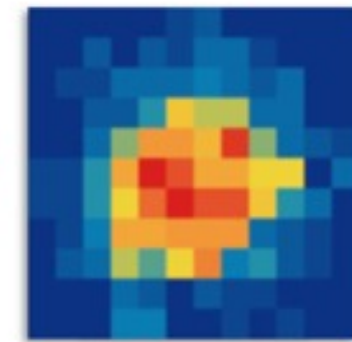
Box Plots



Histograms



Distplots

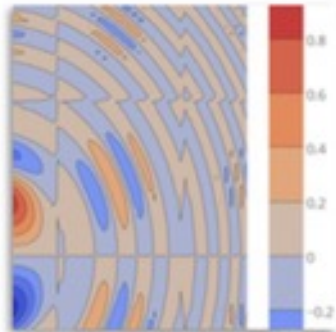


[2D Histograms](#)

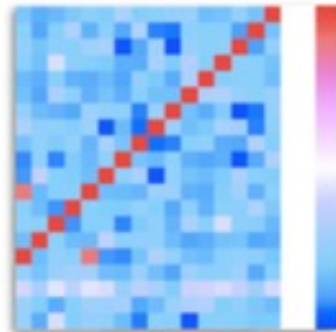
# Python Plotly Graphing Library

## Scientific Charts

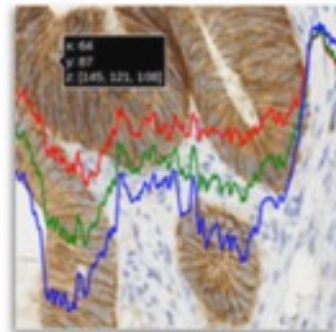
[More Scientific Charts >](#)



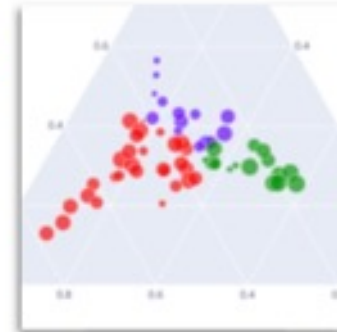
Contour Plots



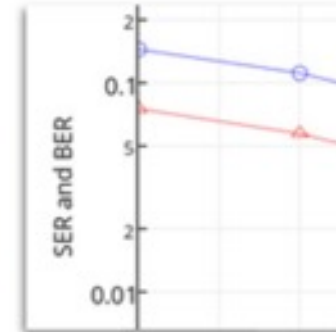
Heatmaps



Imshow



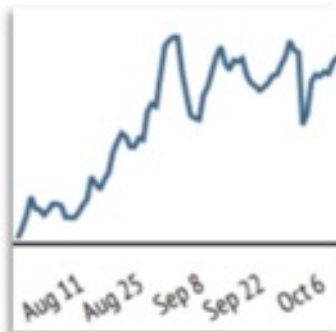
Ternary Plots



Log Plots

## Financial Charts

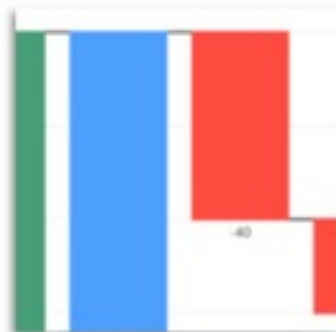
[More Financial Charts >](#)



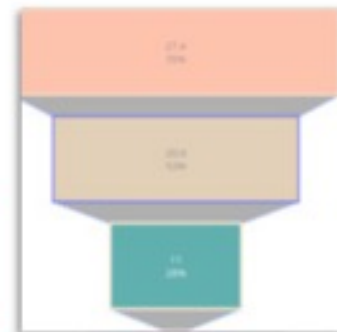
Time Series and Date  
Axes



Candlestick Charts



Waterfall Charts



Funnel Chart



[OHLC Charts](#)

# Python Plotly Graphing Library

## Maps

[More Maps >](#)



Mapbox Choropleth Maps



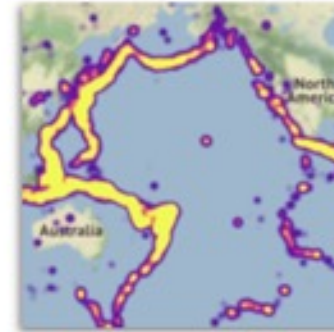
Lines on Mapbox



Filled Area on Maps



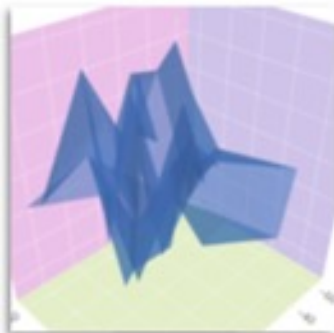
Bubble Maps



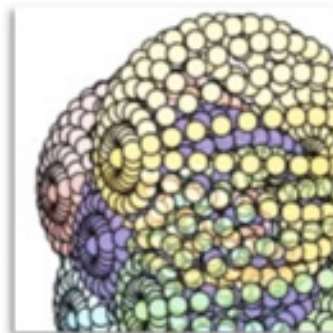
Mapbox Density Heatmap

## 3D Charts

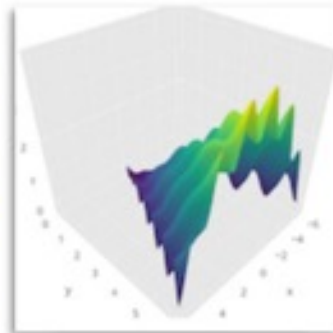
[More 3D Charts >](#)



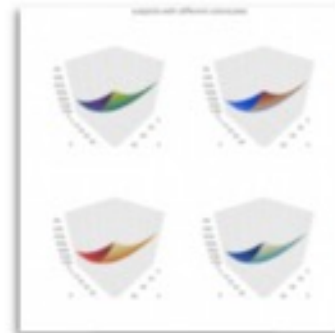
3D Axes



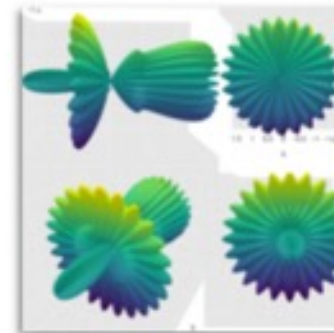
3D Scatter Plots



3D Surface Plots



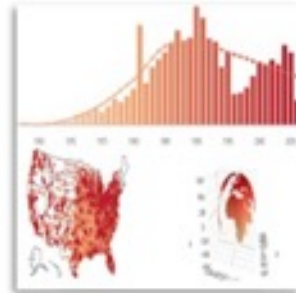
3D Subplots



[3D Camera Controls](#)

# Python Plotly Graphing Library

## Subplots



Mixed Subplots



Map Subplots

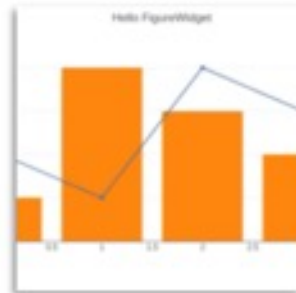


Table and Chart Subplots

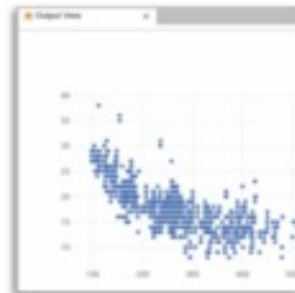


Figure Factory Subplots

## Jupyter Widgets Interaction



Plotly FigureWidget Overview



Jupyter Lab with FigureWidget



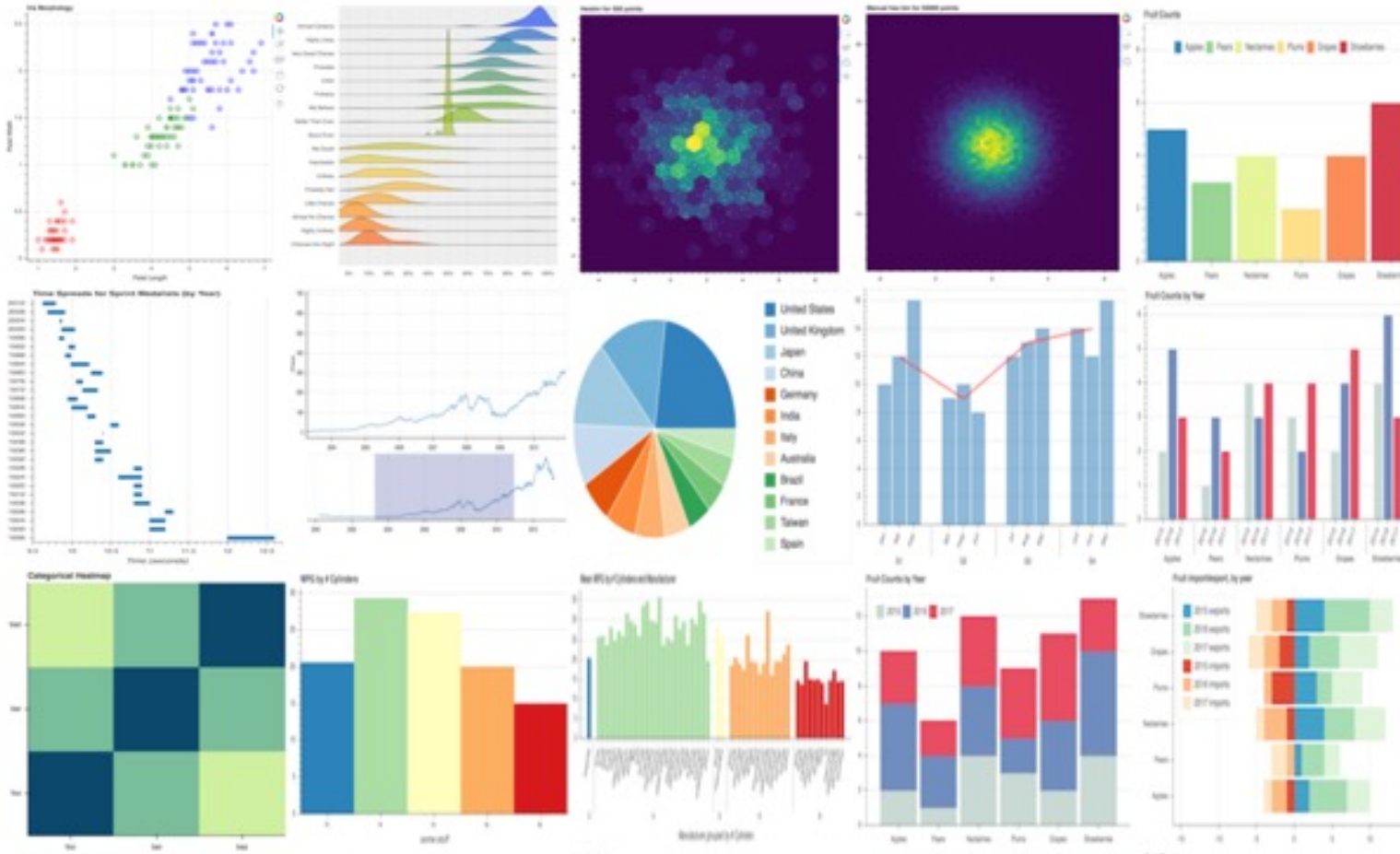
Interactive Data Analysis with FigureWidget ipywidgets

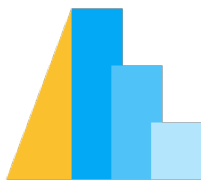


Click Events



# bokeh Python Bokeh





# Python Altair



Vega-Altair Getting Started User Guide Examples API Reference Ecosystem Release Notes



## Vega-Altair: Declarative Visualization in Python



Vega-Altair is a declarative statistical visualization library for Python, based on [Vega](#) and [Vega-Lite](#).

*The Vega-Altair open source project is not affiliated with Altair Engineering, Inc.*

With Vega-Altair, you can spend more time understanding your data and its meaning. Altair's API is simple, friendly and consistent and built on top of the powerful [Vega-Lite](#) visualization grammar. This elegant simplicity produces beautiful and effective visualizations with a minimal amount of code.

You can browse this documentation via the links in the top navigation panel. In addition to reading this documentation page, it can be helpful to also browse the [Vega-Lite documentation](#).

<https://altair-viz.github.io/>

# Iris flower data set

**setosa**



**versicolor**



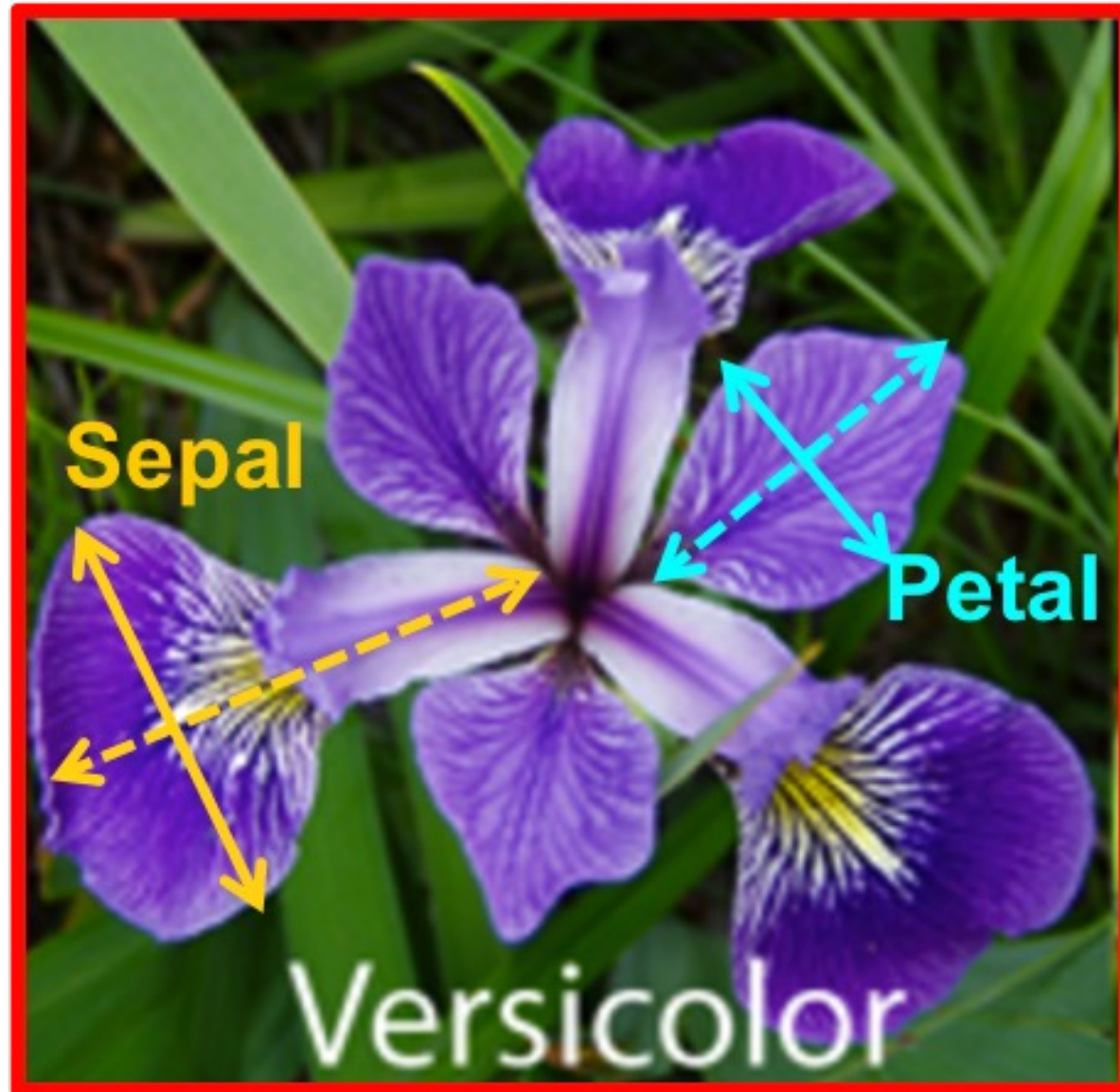
**virginica**



Source: [https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)

Source: <http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/>

# Iris Classification



# iris.data

<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
```

**setosa**



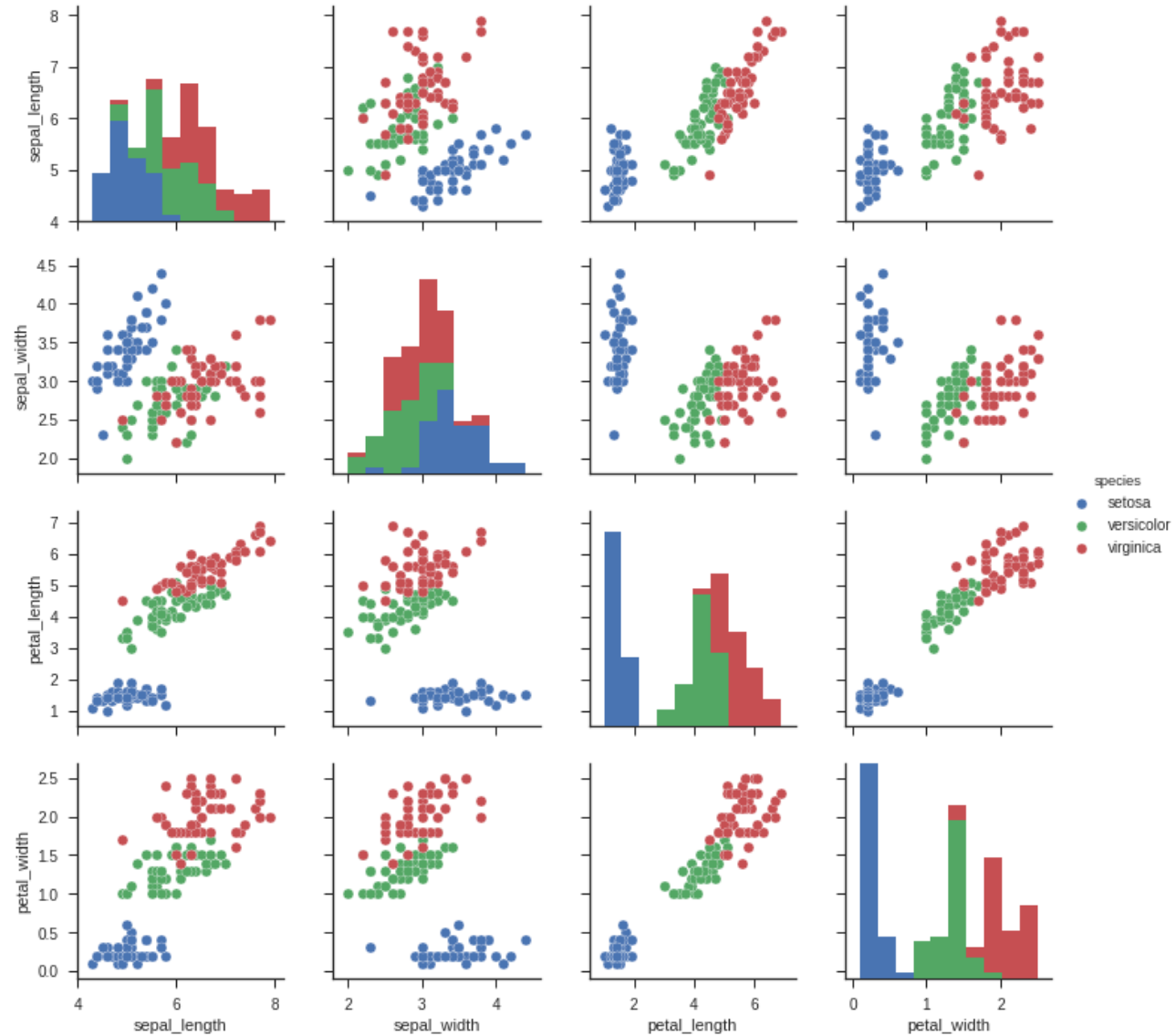
**virginica**



**versicolor**



# Iris Data Visualization



# Data Visualization in Google Colab

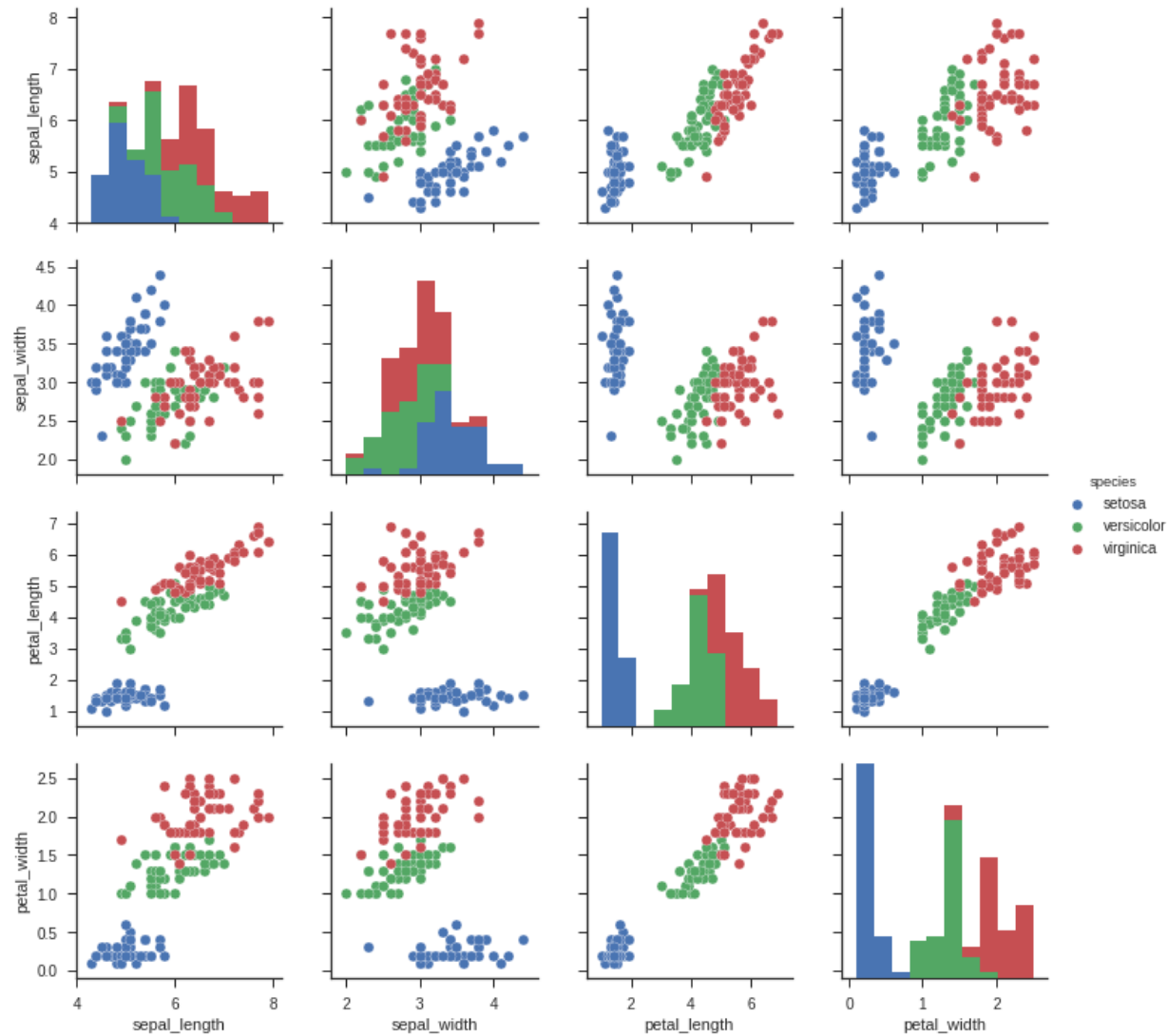
The screenshot displays the Google Colab interface for a notebook titled "python101.ipynb". The top navigation bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", with a status indicator "All changes saved". On the right, there are icons for "Comment", "Share", and a user profile "A". Below the navigation bar, the "Table of contents" sidebar is visible, listing various topics such as "Python101", "Python File Input / Output", "OS, IO, files, and Google Drive", "Python Try Except", "Python Class", "Python Programming", "Python String and Text", "Python Numpy", "Python Pandas", and "Python Data Visualization". The "Python Data Visualization" section is currently selected and expanded, showing sub-topics like "Machine Learning with scikit-learn", "Classification and Prediction", "K-Means Clustering", "Deep Learning for Financial Time Series Forecasting", "Portfolio Optimization and Algorithmic Trading", "Investment Portfolio Optimisation with Python", and "Efficient Frontier Portfolio Optimisation in Python".

The main workspace shows a code cell with the following Python code:

```
(2) 1 import seaborn as sns
     2 sns.set(style="ticks", color_codes=True)
     3 iris = sns.load_dataset("iris")
     4 g = sns.pairplot(iris, hue="species")
```

Below the code, a pairplot is displayed, showing the relationships between the variables "sepal\_length", "sepal\_width", and "th" (likely "petal\_width") for the three species: "setosa" (blue), "versicolor" (orange), and "virginica" (green). The diagonal of the plot shows kernel density estimates (KDEs) for each variable, while the off-diagonal plots show scatter plots of pairs of variables. A legend on the right side of the plot identifies the species by color: blue for "setosa", orange for "versicolor", and green for "virginica".

```
import seaborn as sns
sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris, hue="species")
```





```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
```

```
# Import Libraries
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
print('imported')
```

imported

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']  
df = pd.read_csv(url, names=names)  
print(df.head(10))
```

```
# Load dataset  
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']  
df = pd.read_csv(url, names=names)  
print(df.head(10)).
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

# df.tail(10)

```
print(df.tail(10)).
```

	sepal-length	sepal-width	petal-length	petal-width	class
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

# df.describe()

```
print(df.describe())
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
print(df.info())  
print(df.shape)
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 150 entries, 0 to 149  
Data columns (total 5 columns):  
sepal-length      150 non-null float64  
sepal-width       150 non-null float64  
petal-length      150 non-null float64  
petal-width       150 non-null float64  
class             150 non-null object  
dtypes: float64(4), object(1)  
memory usage: 5.9+ KB  
None
```

```
print(df.shape)
```

```
(150, 5)
```

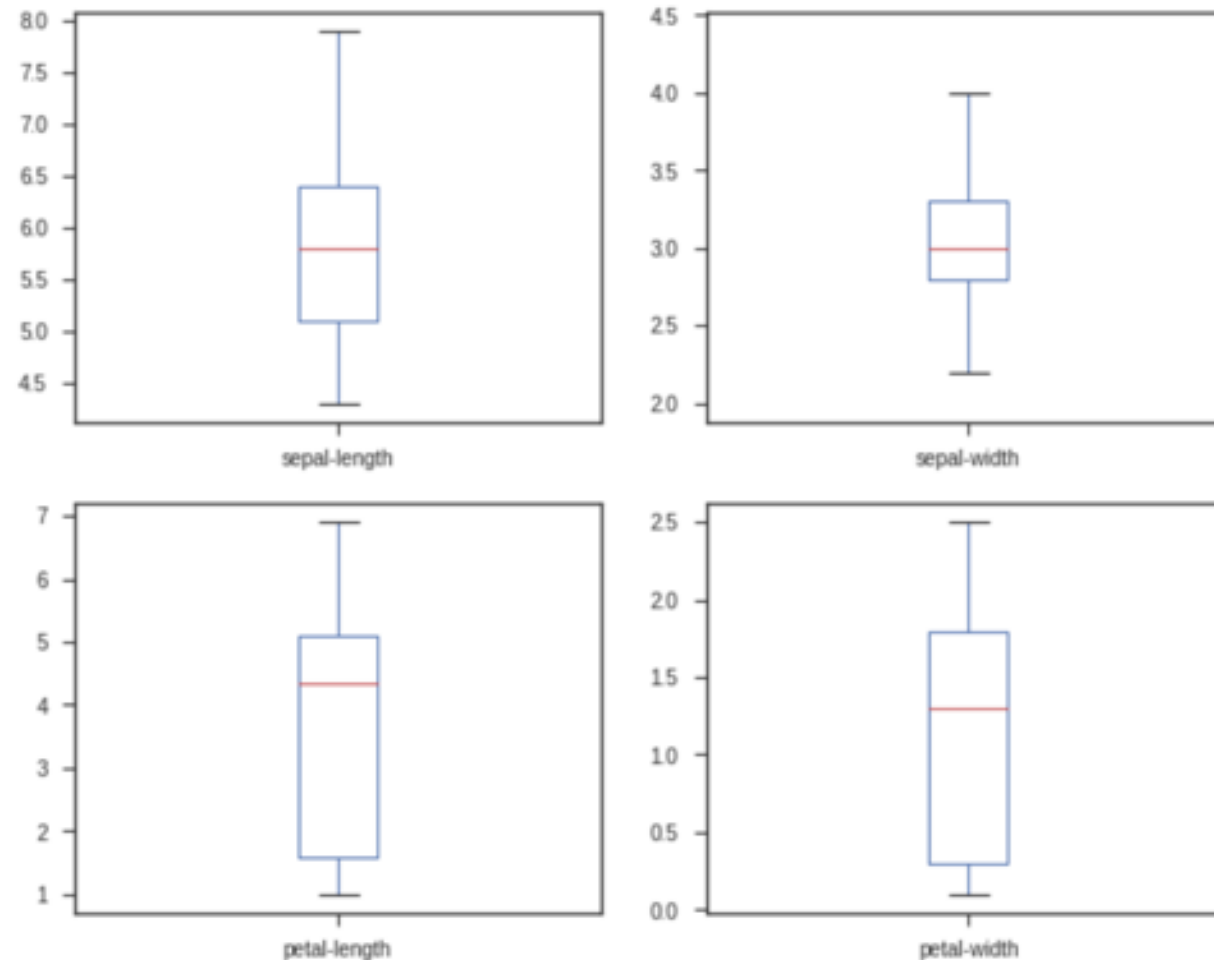
```
df.groupby('class').size()
```

```
print(df.groupby('class').size())
```

```
class
Iris-setosa      50
Iris-versicolor 50
Iris-virginica   50
dtype: int64
```

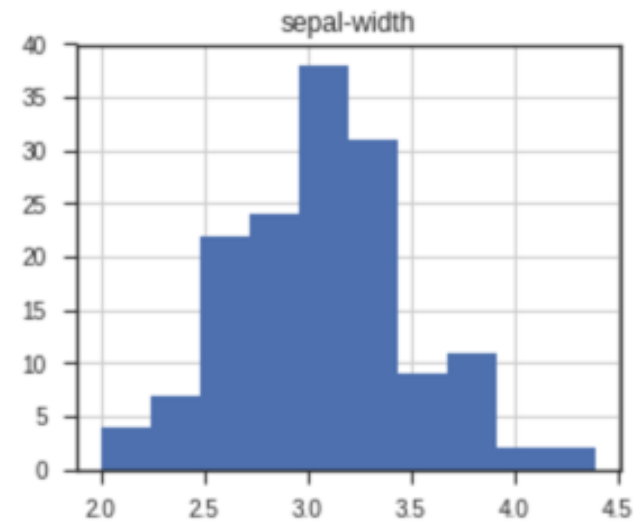
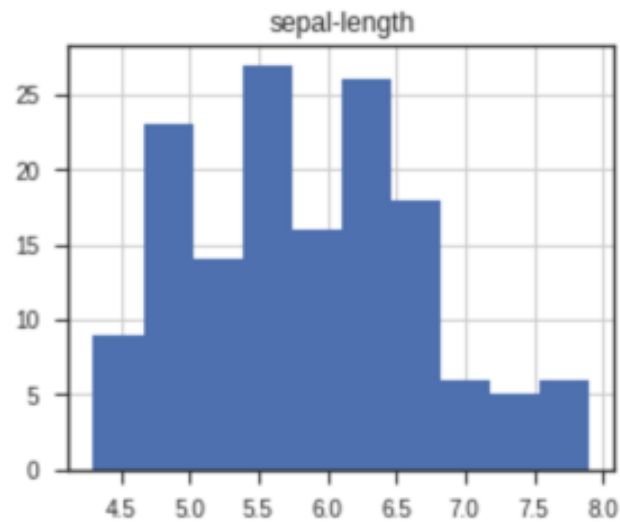
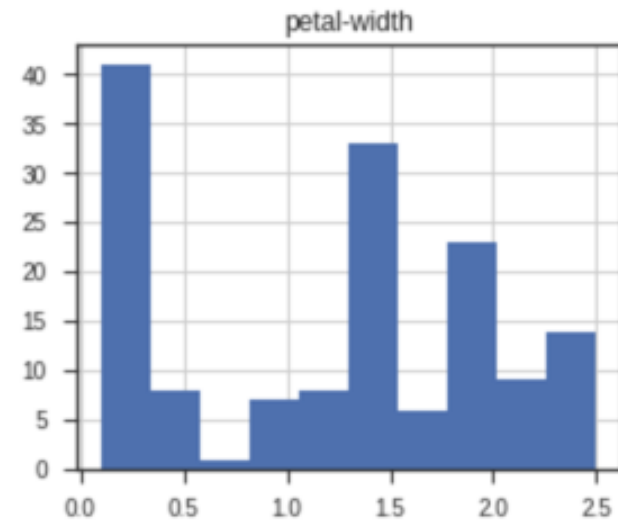
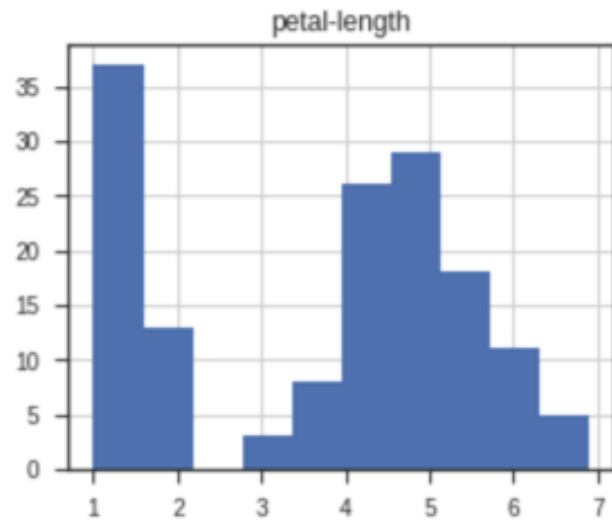
```
plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```

```
plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show(.
```



```
df.hist()  
plt.show()
```

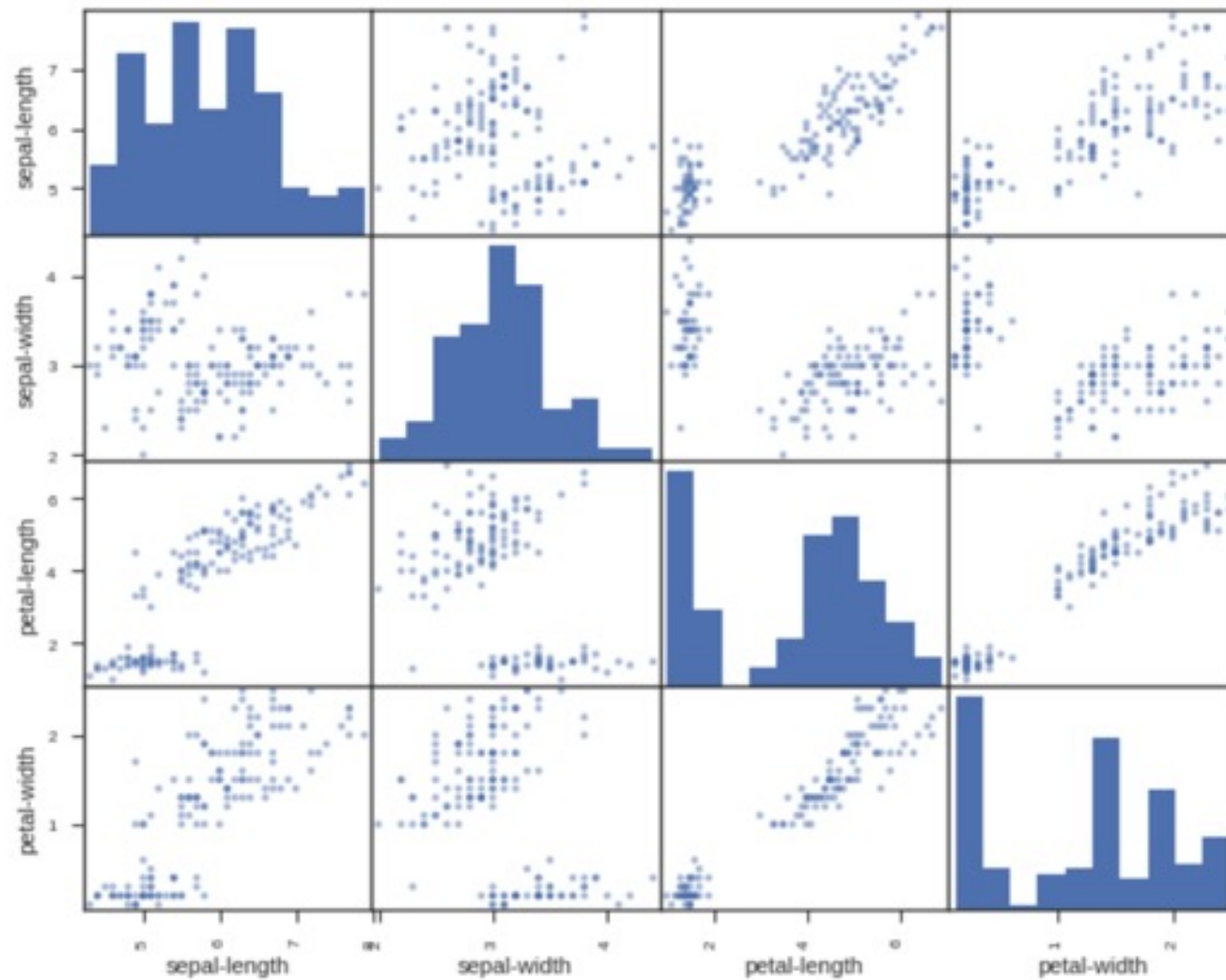
```
df.hist()  
plt.show().
```





```
scatter_matrix(df)
plt.show()
```

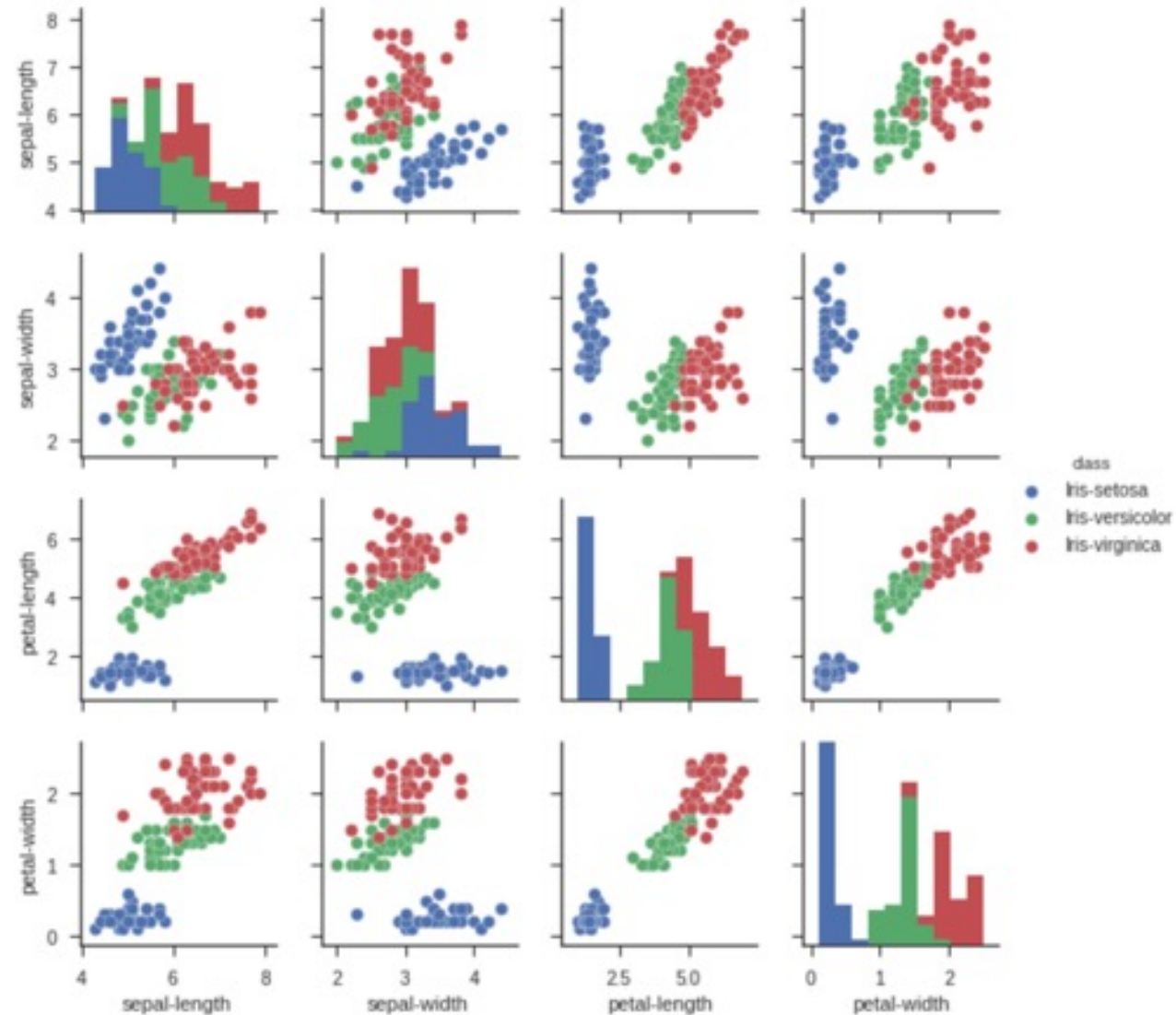
```
scatter_matrix(df)
plt.show()
```



```
sns.pairplot(df, hue="class", size=2)
```

```
sns.pairplot(df, hue="class", size=2)
```

```
<seaborn.axisgrid.PairGrid at 0x7f1d21267390>
```



# Wes McKinney (2022), "Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter", 3rd Edition, O'Reilly Media.

wesm / pydata-book Public

Notifications Fork 14.1k Star 19k

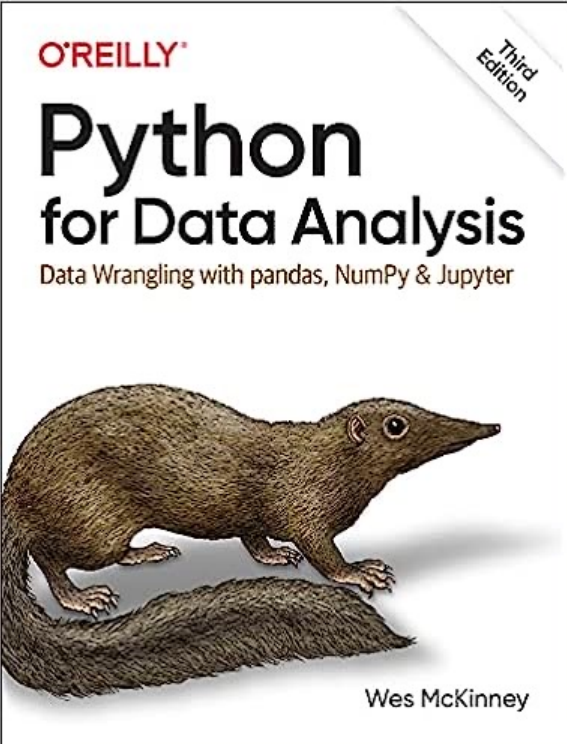
Code Issues 2 Pull requests 1 Actions Projects Wiki Security Insights

3rd-edition 3 branches 0 tags Go to file Code

About

Materials and IPython notebooks for "Python for Data Analysis" by Wes McKinney, published by O'Reilly Media

wesm	Upload cleaner notebook files without internal build toolchai...	f1757b8	3 days ago	70 commits
datasets	Add fec.parquet			10 months ago
examples	Simplifying datasets			10 months ago
.gitignore	Add gitignore			8 years ago
COPYING	Update COPYING			4 months ago
README.md	Update notebooks in advance of publication			7 months ago
appa.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago
appb.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago
ch02.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago
ch03.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago
ch04.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago
ch05.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago
ch06.ipynb	Upload cleaner notebook files without internal build toolchai...			3 days ago



O'REILLY

Python for Data Analysis

Data Wrangling with pandas, NumPy & Jupyter




Wes McKinney

<https://github.com/wesm/pydata-book>

# Wes McKinney (2022), "Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter", 3rd Edition, O'Reilly Media.

3rd-edition ▾ pydata-book / ch04.ipynb Go to file ...

wesm Upload cleaner notebook files without internal build toolchain instru... ... Latest commit f1757b8 3 days ago History

3 contributors   

1224 lines (1224 sloc) | 21.9 KB <> 📄 Raw Blame ✎ ▾ 📄 🗑️

```
In [1]: import numpy as np
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc("figure", figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
```

```
In [2]: import numpy as np

my_arr = np.arange(1_000_000)
my_list = list(range(1_000_000))
```

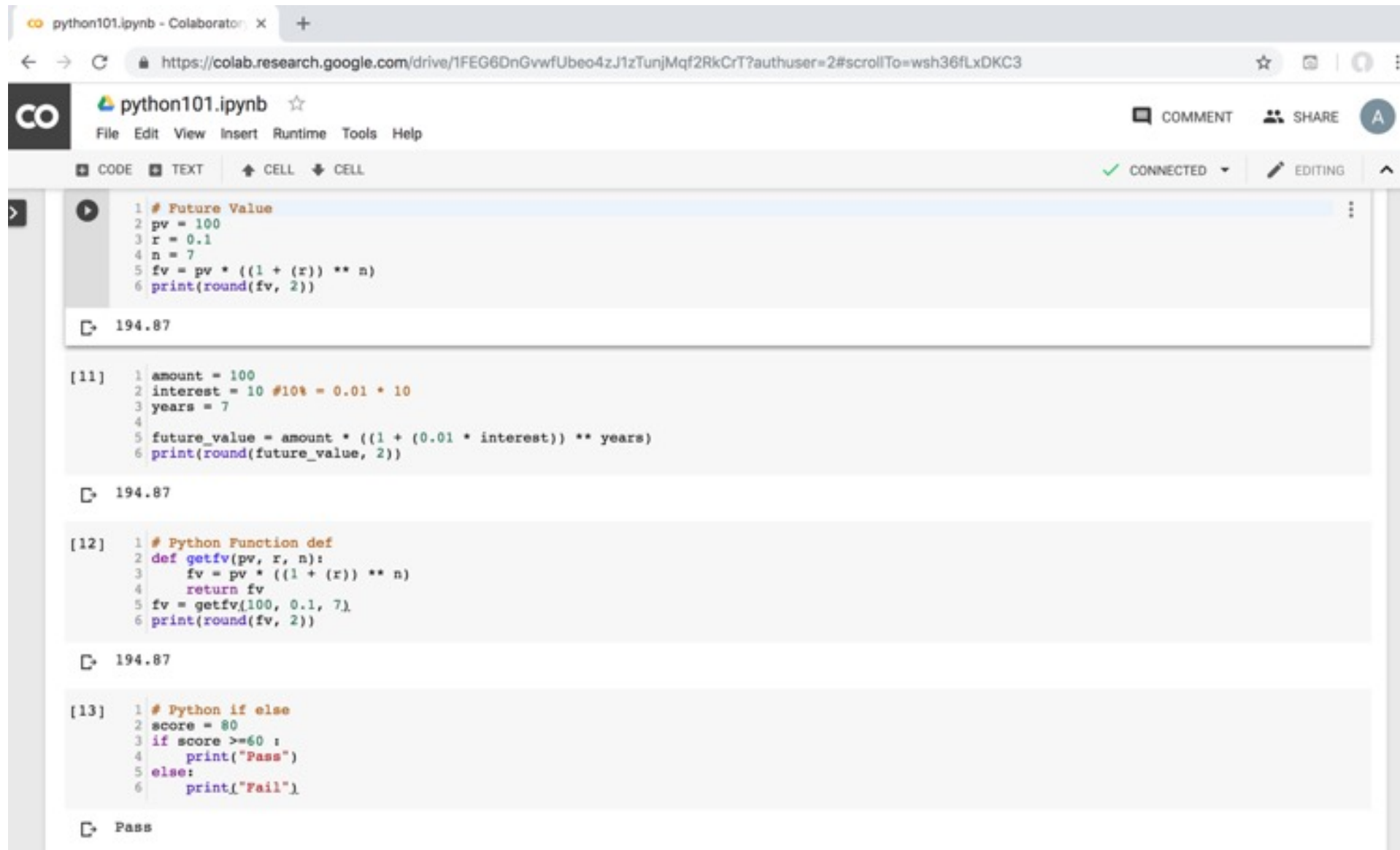
```
In [3]: %timeit my_arr2 = my_arr * 2
%timeit my_list2 = [x * 2 for x in my_list]
```

```
In [4]: import numpy as np
data = np.array([[1.5, -0.1, 3], [0, -3, 6.5]])
data
```

Source: <https://github.com/wesm/pydata-book/blob/3rd-edition/ch04.ipynb>

# Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>



The screenshot shows a Google Colab notebook titled "python101.ipynb". The interface includes a browser address bar, a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), and a toolbar with options like CODE, TEXT, CELL, and a status indicator showing "CONNECTED" and "EDITING".

The notebook contains four code cells:

- Cell 1:** A code cell with the following Python code:

```
1 # Future Value
2 pv = 100
3 r = 0.1
4 n = 7
5 fv = pv * ((1 + (r)) ** n)
6 print(round(fv, 2))
```

The output is "194.87".
- Cell 11:** A code cell with the following Python code:

```
1 amount = 100
2 interest = 10 #10% = 0.01 * 10
3 years = 7
4
5 future_value = amount * ((1 + (0.01 * interest)) ** years)
6 print(round(future_value, 2))
```

The output is "194.87".
- Cell 12:** A code cell with the following Python code:

```
1 # Python Function def
2 def getfv(pv, r, n):
3     fv = pv * ((1 + (r)) ** n)
4     return fv
5 fv = getfv(100, 0.1, 7)
6 print(round(fv, 2))
```

The output is "194.87".
- Cell 13:** A code cell with the following Python code:

```
1 # Python if else
2 score = 80
3 if score >=60 :
4     print("Pass")
5 else:
6     print("Fail")
```

The output is "Pass".

<https://tinyurl.com/aintpupython101>

# Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>

The screenshot displays a Google Colab notebook interface. At the top, the notebook is titled "python101.ipynb" and shows a menu with options like File, Edit, View, Insert, Runtime, Tools, and Help. The left sidebar contains a "Table of contents" with various topics, including "Python Data Visualization" which is currently selected. The main area shows a code cell with the following Python code:

```
[2] 1 import seaborn as sns
     2 sns.set(style="ticks", color_codes=True)
     3 iris = sns.load_dataset("iris")
     4 g = sns.pairplot(iris, hue="species")
```

Below the code, a pairplot is generated, showing a 4x4 grid of plots. The diagonal plots are histograms of the variables: sepal\_length, sepal\_width, and petal\_width. The off-diagonal plots are scatter plots showing the relationships between these variables. The data points are colored by species: setosa (blue), versicolor (orange), and virginica (green). A legend on the right side of the plot identifies the species.

<https://tinyurl.com/aintpupython101>

# Papers with Code State-of-the-Art (SOTA)

## Computer Vision



▶ [See all 1415 tasks](#)

## Natural Language Processing



▶ [See all 664 tasks](#)

# Summary

- **Foundations of Big Data Analysis in Python**
  - **Python Ecosystem for Data Science**
  - **Python**
    - Programming language
  - **Numpy**
    - Scientific computing
  - **Pandas**
    - Data structures and data analysis tools



# References

- Wes McKinney (2022), "Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter", 3rd Edition, O'Reilly Media.
- Steven D'Ascoli (2022), Artificial Intelligence and Deep Learning with Python: Every Line of Code Explained For Readers New to AI and New to Python, Independently published.
- Aurélien Géron (2019), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition, O'Reilly Media.
- Python Programming, <https://pythonprogramming.net/>
- Python, <https://www.python.org/>
- Python Programming Language, <http://pythonprogramminglanguage.com/>
- Numpy, <http://www.numpy.org/>
- Pandas, <http://pandas.pydata.org/>
- Skikit-learn, <http://scikit-learn.org/>
- Min-Yuh Day (2023), Python 101, <https://tinyurl.com/aintpupython101>