Introduction to data analysis

Part 9. Python

Let's get started.

the master labs • academy

NCOI LEARNING

CLEAN THE DATA

EXPLORE THE DATA

ANALYZE IN DEPTH

VISUALIZE THE RESULT

PYTHON

What will we do in this course?

1. Introduction to Python

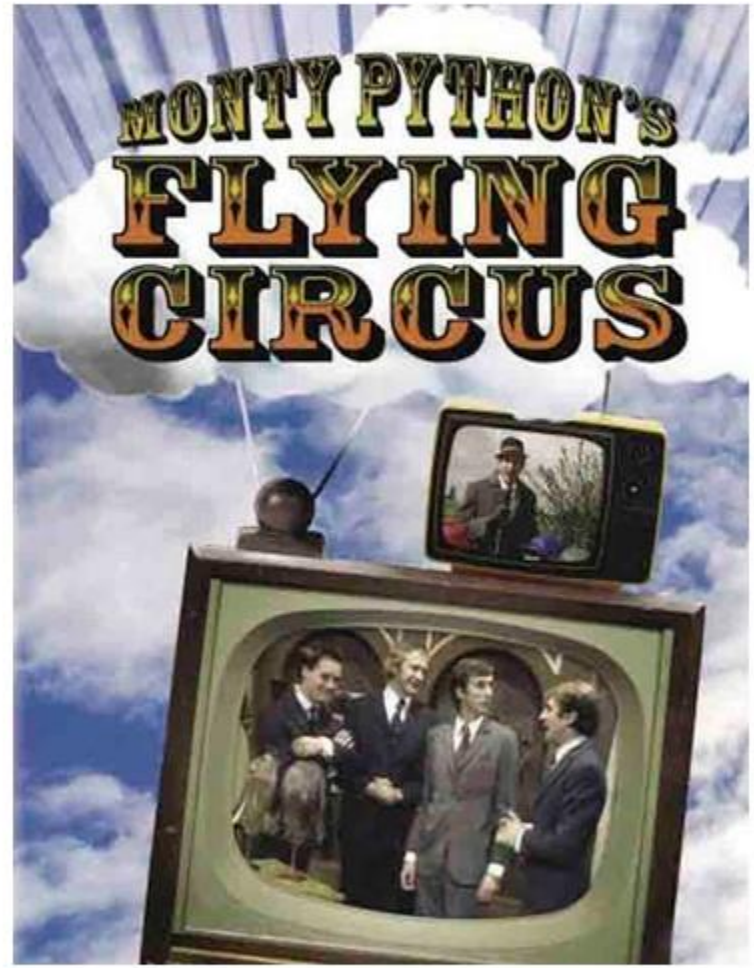
2. Python basics
   - Syntax
   - Control flow
   - Loops
   - Dictionaries

3. Python packages

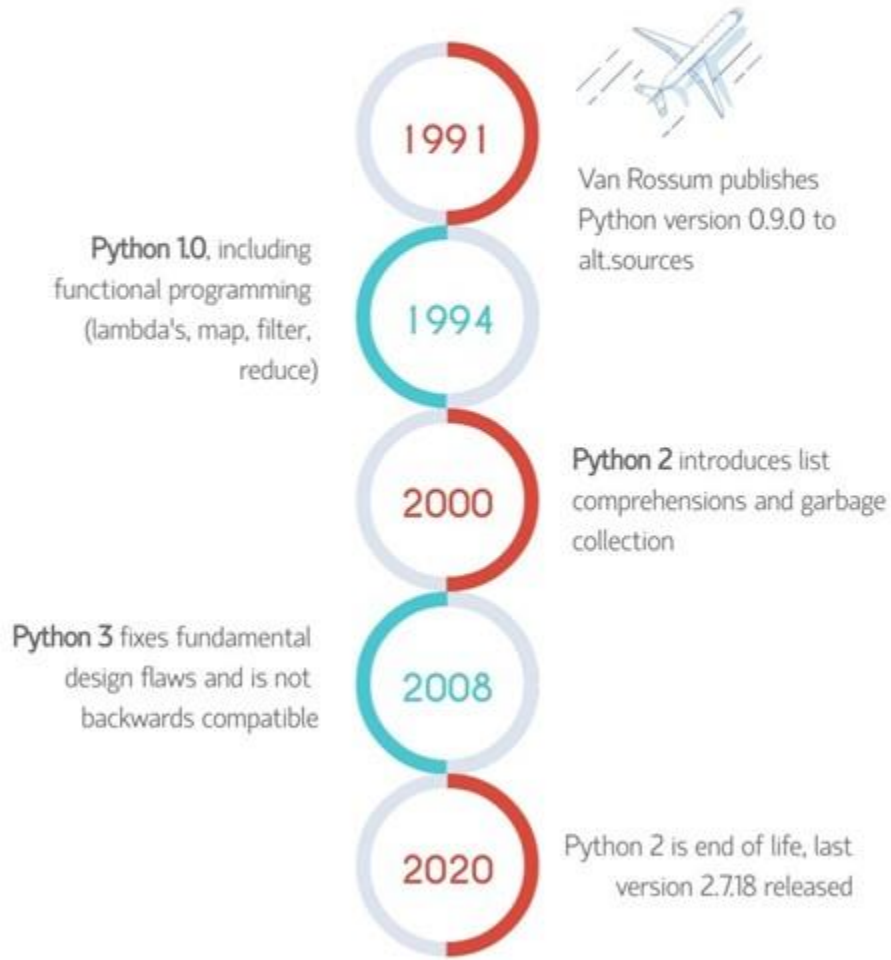
What is Python

What is Python

- High level programming language
- Open source
- Dynamic type system
- Readability
- Interpreted language
- Multi paradigm
- General: very powerful and
  used in many different ways and areas



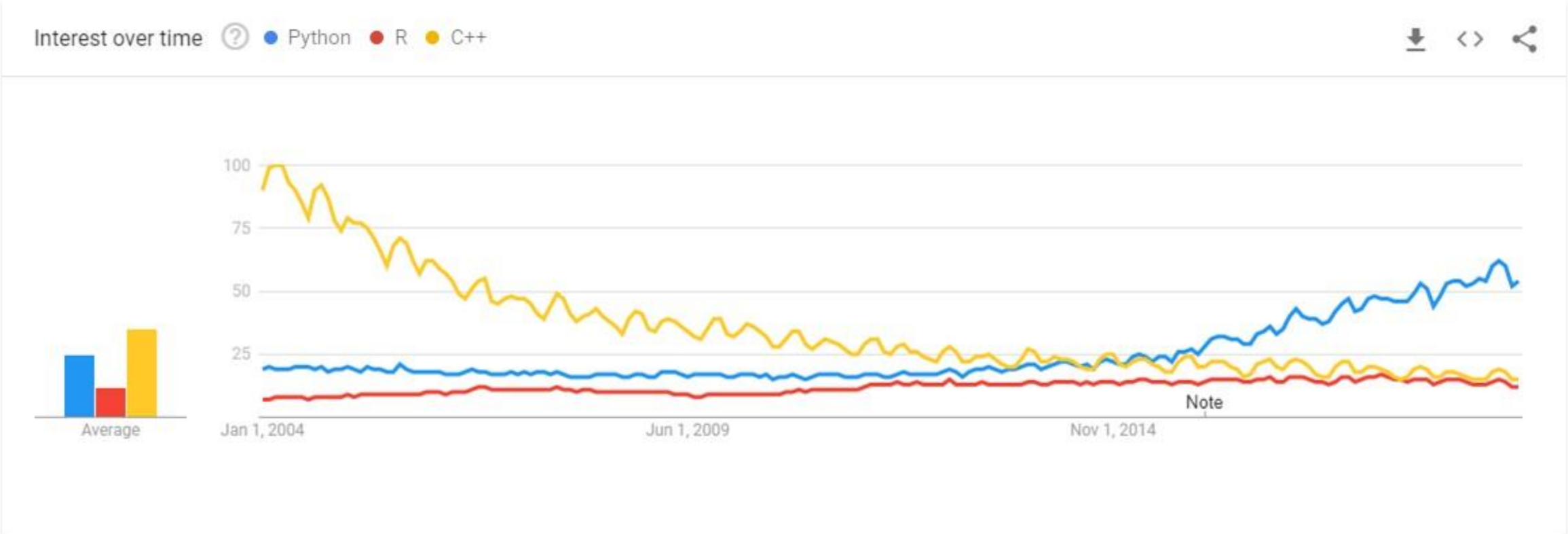
History of Python



History of Python

- “Conceived” in late 1980s by Guido Van Rossum
- Successor to the ABC language
- Core philosophy:
  - Beautiful is better than ugly
  - Explicit is better than implicit
  - Simple is better than complex
  - Complex is better than complicated
  - Readability counts
- Current version is Python 3.9.1, introduced in December 2020
- Python 2.7 end of life as of January 2020

History of Python



Python vs. other programming languages

In general

- Extensible design
- Community involved design
- Emphasizing fun
- Culture

Python vs. R

Python

- Open-source
- General approach to data science
- Catching up on libraries
- Linear, smooth learning curve

R

- Open-source
- Main use is statistical analysis
- 12000 packages in CRAN
- Difficult at the beginning

Python vs. C++

Python

- Interpreted language
- Easier to learn
- Fastest growing language for embedded computing
- Readability

- C++
- Compiled language
- More difficult to learn
- Creates more compact and faster runtime code
- Complex syntax

Why Python in data science

- Interpreted language, so no compiling is needed
- Fast prototyping
- Easily explore data and work out concepts
- Extensible to include popular algorithms from other languages
- Libraries such as matplotlib, numpy and pandas make data science accessible
- Jupyter notebooks for combining text and code in a single, easy to share document

Installing Python

Installing Python

- Python 2.7 vs Python 3.x
- Long story short, Python 2.x was replaced by Python 3 in 2008. Support for Python 2.7 has ended as of 1 January 2020. Most, but not all libraries have been ported to Python 3.x. For data science purposes, all we need is Python 3.

Anaconda

- Python distribution for large-scale data analytics
- Provides many of the tools needed to analyse large sets of data
- Includes:
  - Core Python language
  - Over 1000 packages, many for data science
  - Package management with conda
  - IPython
  - Much more

Miniconda

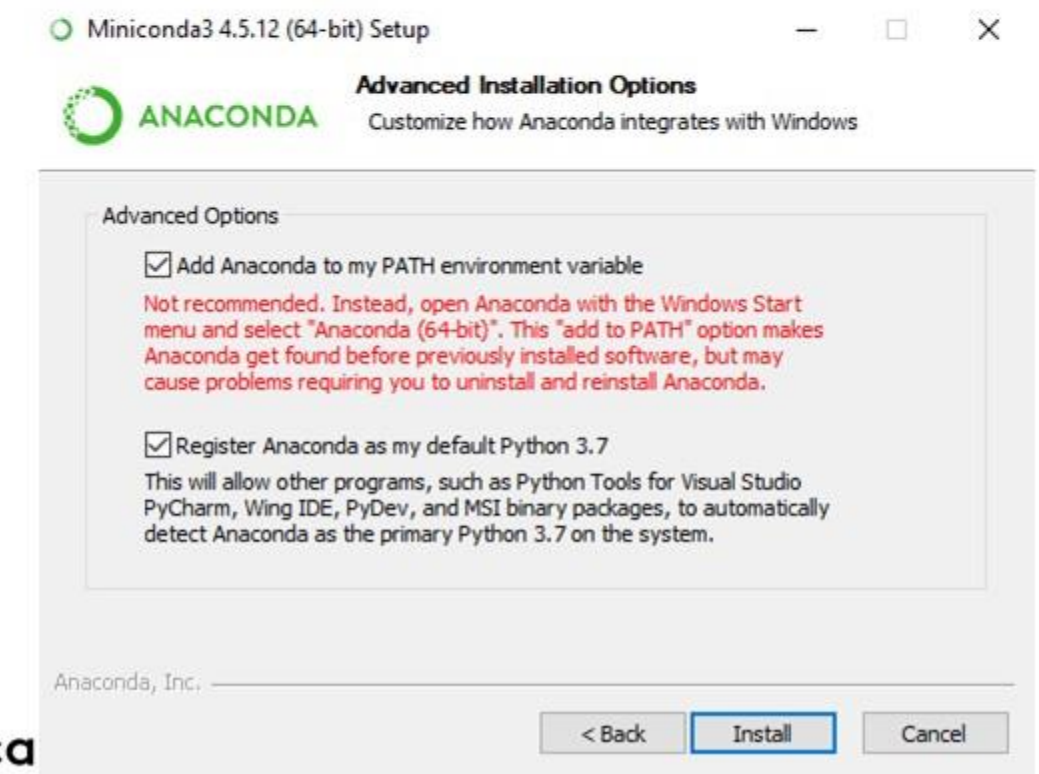
- Slimmed down version of Anaconda
- Includes the basic requirements
- Allows you to install packages as needed, decreasing download and installation time

Anaconda, Miniconda, or Python

- Anaconda and Miniconda both install Python
- Anaconda and Miniconda make it easier to install Jupyter Notebooks
- For your own machine it's your own choice
- We will be installing Miniconda due to the shorter installation time and the fact that we'll be using Jupyter Notebooks

Installing Miniconda

- Go to https://conda.io/en/latest/miniconda.html
- Download the latest version for your OS
- Run the installer



Python basics

Python syntax

- No curly brackets
- Semicolons (;) optional
- Whitespace indentation used to delimit blocks

Example Python code

```
def SayHello():
    name = input("What is your name? ")
    if (name == "Rik"):
        print("Oh, hello there " + name + ".")
    elif (name == "Sven"):
        print(name + " please... You're not supposed to be in this Python course.")
    else:
        print(name + "... I've never heard of you before.")

SayHello()
```

Comments

- Use # sign to write comments in your code
- Comments are ignored when running code

```
# This is a comment
print("This is no comment")
```

Variables

- Declare a variable by giving it a value
- Use the = sign

```
my_variable = 'Some text'
print(my_variable)
```

Errors

- Two types of common error messages
  - SyntaxError
    - Issue with Python syntax
  - NameError
    - Undefined variables are used

```
NameError                                 Traceback (most recent call last)
<ipython-input-2-cbb98db30279> in <module>
----> 1 print(Abracadabra)

NameError: name 'Abracadabra' is not defined
```

Strings

- Strings contain text
- Declare strings by using either single (‘Text’) or double (“Text”) quotation marks

```
my_variable = 'Some text'
print(my_variable)
```

Multi-line strings

- Use triple quotation marks to indicate a string that spans over multiple lines
- Considered as a comment if not assigned to a variable

```
leaves_of_grass = """
Poets to come! orators, singers, musicians to come!
Not to-day is to justify me and answer what I am for,
But you, a new brood, native, athletic, continental, greater than
  before known,
Arouse! for you must justify me.
"""
```

Numbers

- Multiple types of numbers: integers and floats
- Integers are whole numbers
- Floats are decimal numbers

```
an_int = 12
a_float = 1.25

print(an_int - 10)
print(a_float)
```

Calculating

- Use the mathematical signs to perform calculations on numbers
- +
- -
- *
- /

```
# Prints "500"
print(573 - 74 + 1)

# Prints "50"
print(25 * 2)

# Prints "2.0"
print(10 / 5)
```

Exponents

- Use ** to take an exponent of a number
- 3 ** 2 = 9

```
exponent = 3 ** 2
print(exponent)
```

Modulo

- Use modulo (%) to calculate the remainder of a division

```
# Prints 4 because 29 / 5 is 5 with a remainder of 4
print(29 % 5)

# Prints 2 because 32 / 3 is 10 with a remainder of 2
print(32 % 3)

# Modulo by 2 returns 0 for even numbers and 1 for odd numbers
# Prints 0
print(44 % 2)
```

Concatenation

- Add multiple strings together into a single string
- Use the + sign

```
greeting_text = "Hey there!"
question_text = "How are you doing?"
full_text = greeting_text + question_text

# Prints "Hey there!How are you doing?"
print(full_text)
```

Plus equals

- Increment a number by another specified number

```
# First we have a variable with a number saved
number_of_miles_hiked = 12

# Then we need to update that variable
# Let's say we hike another two miles today
number_of_miles_hiked += 2
```

Booleans

- Two possible values
  - True
  - False
- Comparison operators
- Logical operators

```
my_bool = True
my_other_bool = False
```

Comparison operators

- ==
- !=
- >
- >=
- <
- <=

```
print(1 == 1)
print(2 != 4)
print(3 == 5)
print('7' == 7)
```

Logical operators

- and
- or
- not

```
my_first_var = True and True
my_second_var = False or True
my_third_var = not False
```

Functions

- Contain several lines of code which can be reused later
- Consider:
  - Name of function
  - Input(s) of function
  - Output(s) of function
- First define a function, and then call it later

```
def greeting(name, company_name):
    print("Hello " + name + " and welcome to " + company_name)

first_name = "Rik"
company = "The Master Labs"

greeting(first_name, company)
```

Returns

- Return the result of your function and assign it to a variable

```
def divide_by_four(input_number):
    return input_number/4
```

Scope

- Variables declared within a function are only available in that function
- Variables declared outside are available everywhere

```
def create_special_string(special_item):
    return "Our special is " + special_item + "."

print("I don't like " + special_item)
```

Control flow

- Make decisions based on certain conditions
- Decide which part of the program should run based on the result of those decisions
- Boolean expressions
- Comparison operators
- Logical operators
- If, else statements

If statements

- If condition is True, do something

```
if 2 == 4 - 2:
    print("apple")
```

Else statements

- If no condition is True, do this

```
def age_check(age):
  if age >= 13:
    return True
  else:
    return "Sorry, you must be 13 or older to watch this movie."
```

Else-if (elif)

- Why don't we just use if statements?
  - Check each condition sequentially. If True, do something and then stop. If not True, move on to the next condition

```
def thank_you(donation):
  if donation >= 1000:
    print("Thank you for your donation! You have achieved platinum donation status!")
  elif donation >= 500:
    print("Thank you for your donation! You have achieved gold donation status!")
  elif donation >= 100:
    print("Thank you for your donation! You have achieved silver donation status!")
  else:
    print("Thank you for your donation! You have achieved bronze donation status!")
```

Lists

- Store multiple data in a single variable
- Elements can be different data types
- Use square brackets [ ] to define a list
- Use commas to differentiate elements

```
my_list = ['Gianni', 25, 'Analyst', True]
```

List operations

- Get a single element from a list
- Get a range of elements
- Get the length of a list
- Zip multiple lists together
- Add new items to a list

```
my_list[2]
my_list[1:3]
len(my_list)
zipped_list = zip(list1, list2)
my_list.append('Value')
```

Range

- Function that creates a list of consecutive numbers
- Takes three input parameters: x, y and i
  - x is the starting number for our list
  - y is one more than the final number of our list
  - i is the interval between each number of our list

```
my_list = range(10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Loops

- Repeat the same block of code x number of times
- Three common types of loops in Python
  - For loops: iterate through every item in a list
  - While loops: loop until as long as a certain condition is true
  - List comprehension: create new lists using a loop in a single line

For loops

- General syntax:

```
for <temporary variable> in <list variable>:
    <do something>
```

  - Example:

```
dog_breeds = ['french bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']
for breed in dog_breeds:
    print(breed)
```

While loops

- General syntax:

```
while <condition is true>:
    <do something>
```

  - Example:

```
dog_breeds = ['bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']
index = 0
while index < len(dog_breeds):
  print(dog_breeds[index])
  index += 1
```

List comprehension

- General syntax:

```
new_list = [<item>/<value> for <item> in <list> if <condition>]
```

  - Example:

```
words = ["@coolguy35", "#nofilter", "@kewldawg54"]
usernames = [word for word in words if word[0] == '@']
```

Dictionaries

- Store multiple data in a single variable
- Key/value pairs
- Use angular brackets {} to define a dictionary
- Use commas to differentiate key/value pairs, colon (:) between key and value

```
greetings_dict = {
    'nl': 'Hallo!',
    'fr': 'Bonjour!',
    'pl': 'Cześć!'
}
```

Dictionary operations

- Get an element from a dictionary

```
my_dictionary['my_key']
```

- Add an element to a dictionary

```
my_dictionary['other_key'] = 'updated value'
```

- Update an element on a dictionary

```
my_dictionary['my_key'] = 'updated value'
```

Dictionary values

- Values on a dictionary can be different data types, even dictionaries

```
interstellar = {
    'title': 'Interstellar',
    'release_year': 2014,
    'runtime': 169,
    'imdb_score': 8.6,
    'genres': ['Adventure', 'Drama', 'Sci-Fi'],
    'director': {
        'name': 'Christopher Nolan',
        'date_of_birth': '1970-07-30'
    }
}
```

List vs. Dictionary

List

- Store data in a single variable
- Use square brackets [] to define
- Elements can be different data types
- Retrieve elements by index

Dictionary

- Store data in a single variable
- Use angular brackets to define {}
- Elements can be different data types
- Retrieve elements by key

Python packages

Jupyter

- A web application which is closely integrated with Python
- Create and share documents
- Includes live code, equations, visualisations and more

Numpy

- Library for scientific computing using Python
- Allows us to:
  - Efficiently work with numbers
  - Generate random numbers
  - Perform many numerical functions (such as calculating sin, cos, mean, median, and many others)

SciPy

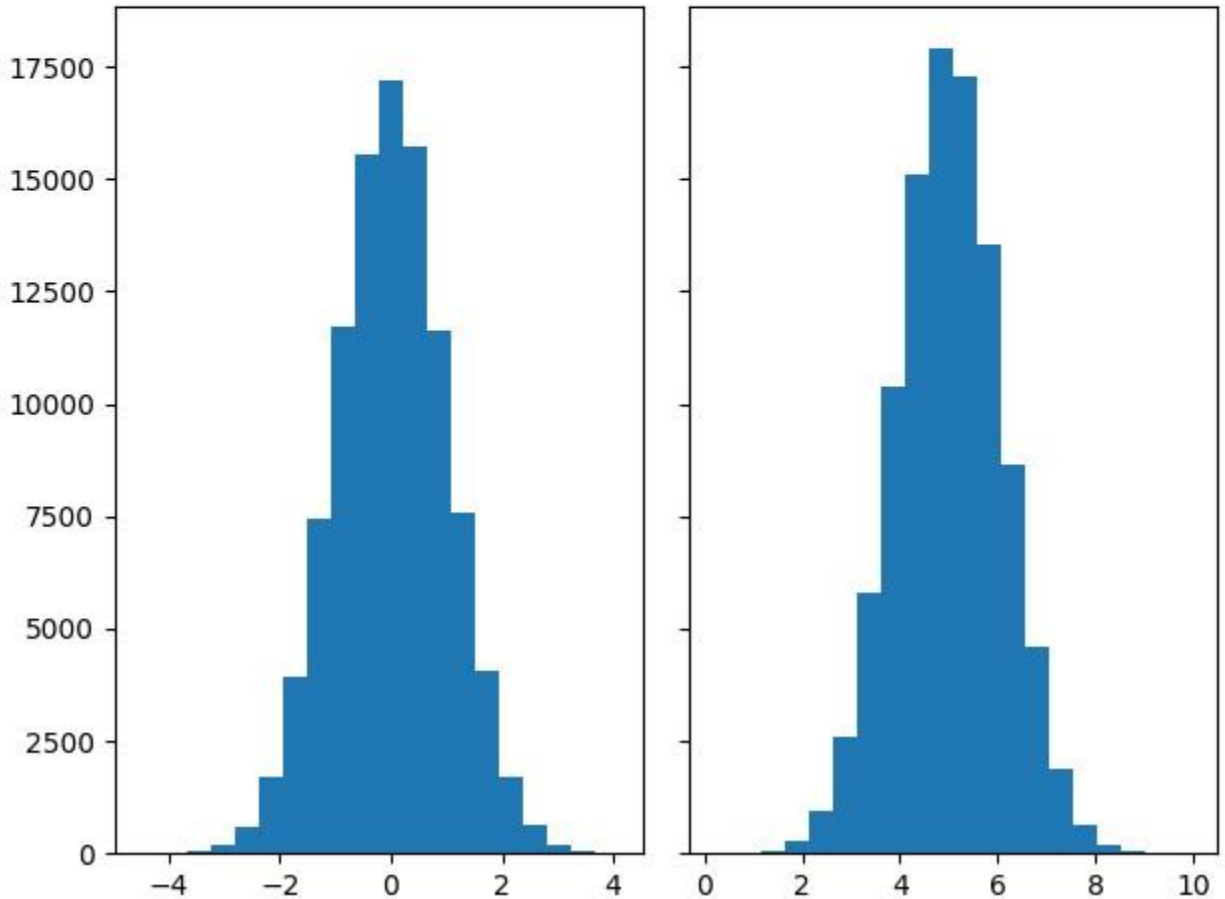
- Python library for statistical analysis
- Allows us to:
  - Perform a number of statistical tests, including One Sample T Tests, Two Sample T Tests, ANOVA, Tukey Tests, Binomial Tests, Chi Square

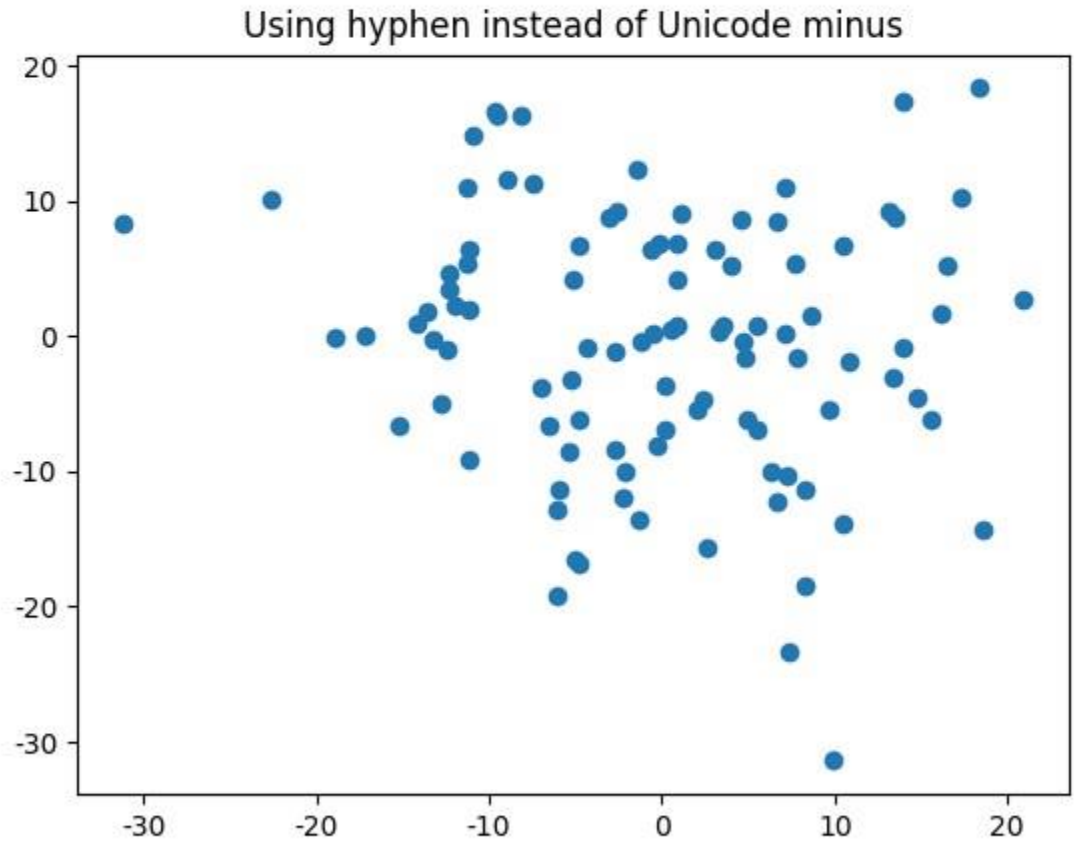
Pandas

- Library for data manipulation and analysis
- Gives tools to work with tabular data
- Similar functionality to SQL or Excel, but within Python
- Create tables, load data from files, select rows or columns

Matplotlib

- 2D plotting library for Python





Scikit-learn

- Aggregation package
- *"features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN" (Wikipedia)*
- Machine learning in python