75 Minutes of Python

Alexander Clark
Columbia University - Stats 1101

August 19, 2023

1 What is Python?

Python is a general-purpose programming language. Our purposes are statistical calculations and the occasional graph. We will run code in Google Colaboratory notebooks. This roughly mirrors running Jupyter notebooks on your own machine, but we skip the setup by running everything in the browser. If you master intro stats and become proficient in Python and SQL, there is a six-figure job waiting for you at any number of tech companies.[1]

Python is popular because the code is readable. More specifically, "Pythonic" code is readable. Pythonic code not only gets you from point A to B, but it uses Python as it was intended and conforms to the standards of beauty described in *The Zen of Python* and elsewhere.

> Beautiful is better than ugly.
> Explicit is better than implicit.
> Simple is better than complex.
> Complex is better than complicated.
> Flat is better than nested.
> Sparse is better than dense.
> Readability counts.
> Special cases aren't special enough to break the rules.
> Although practicality beats purity.
> Errors should never pass silently.
> Unless explicitly silenced.
> In the face of ambiguity, refuse the temptation to guess.
> There should be one—and preferably only one—obvious way to do it.
> Although that way may not be obvious at first unless you're Dutch.
> Now is better than never.
> Although never is often better than *right* now.
> If the implementation is hard to explain, it's a bad idea.
> If the implementation is easy to explain, it may be a good idea.
> Namespaces are one honking great idea—let's do more of those!

The Zen of Python, by Tim Peters

Python can seem endlessly complex. *Fluent Python* is 1,012 pages. Luckily, we don't need to aspire to fluency. Let's learn just enough of the language to order off the kids' menu.

2 Python as a Calculator

2.1 Variables

A Python variable holds a value. It can be a string, a number, or perhaps a more complicated data type. Variable assignment is done with the equals sign, `=`. Below, we create a string variable called `greeting` and a integer variable called `my_favorite_number`.

```
greeting = "Hello, World!"
my_favorite_number = 91
```

vars.py

[1] At least there was in early 2022. Data scientists are over-represented in layoffs.

Now compare the output you get from the following.

```
print(greeting)
print("greeting")
print("Hello, World!")
print(91)
print(my_favorite_number)
print("my_favorite_number")
```

print-vars.py

Note that while $x = x + 1$ is nonsense as a mathematical equation, in Python `x = x+1` simply adds one to the value of `x`. What's the value of `x` at the end of the following program?[2]

```
x = 0
x = x + 1
x = x + 1
x += 1 # syntatic sugar
print(x)
```

reassignment.py

2.2 Comments

Commenting your code is helpful if you care about your colleagues or your future self. Comments should add clarity to the intention and workings of code. A comment is a piece of code that isn't actually executed—it's a comment left for the reader or the person who inherits and modifies your code. Everything after a `#` will be ignored by Python.

```
# This will print a greeting.
print("Hello, World!")
```

comment.py

You might also use end-line comments like the following

```
print("Hello, World!") # Prints a greeting
```

inline.py

Perfect comment technique does not correct for bad code though. Compare the following blocks of code.

```
x = 90 # Wins
y = 10 # Losses
a = 100 * x/(y+x) # Winning Percentage
```

compensation.py

```
wins = 90
lossses = 10
winning_percentage = 100 * wins / (losses + wins)
```

good-names.py

The first block is commented and the second is not. Still, the second code block is better because the variable names are chosen so you don't *need* comments. Good naming becomes even more important as the program becomes longer and variables are used over and over.

2.3 Calculations

Below is a list of some typical operations. What might stand out is that exponentiation is done with `**` and not `^`. It happens that `^` still works, but it does something else.

[2] Three.

Operator	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Floating point (normal) division
`**`	Exponentiation

We can use parenthesis for readability and to order operations in the usual way.

```
a = 1 - 1 * 0
b = (1 - 1) * 0
print(a, b)
```

parens.py

For our first substantive program, let's calculate the test statistic for $\hat{p} = 0.34$ with $n = 116$ if we assume a null hypothesis of $p = \frac{1}{3}$. We will obtain `z = 0.15231546211727914`.

```
p_hat = 0.34
p = 1/3
n = 116
sd = (p*(1-p) / n)**0.5 # st dev from sampling distribution
z = (p_hat - p)/sd # test stat
```

z-stat.py

2.4 Truth and Decision Structures

You'll notice the test statistic we just calculated is small. For a 95% confidence level and a two-sided test, we would fail to reject the null hypothesis because $|z| < 1.96$. It would be nice if Python could make that call without us even having to look at the statistic. We can do that with if statements. If statements are based on truth statements, or boolean expressions.

```
if True:
    print(1)
else:
    print(0)
```

if-true.py

```
if False:
    print(4)
else:
    print(5)
```

if-false.py

We don't work directly with `True` and `False` in practice. We have to make use of comparison operators.

Operator	Description
`==`	Equality, `1 == 1`
`>`	Greater Than, `1 > 0`
`<`	Less Than, `1 < 2`
`>=`	Greater Than or Equal, `1 >= 1`
`<=`	Less Than or Equal, `9 - 1 <= 7 + 1`

Observe.

```
if 1 == abs(-1):
    print("Equality in absolute value")
else:
    print("Inequality")
```

if-calc.py

Using the `abs()` function above to take an absolute value, Python knows $1 = |-1|$ and executes only the first print statement. Finally, let's have Python make the call on our hypothesis test.

```
if abs(z) < 1.96:
    print("Fail to reject.")
else:
    print("Reject.")
```

reject-rule.py

3 Libraries for Statistical Functions and Graphs

We've hit a ceiling in Python's usefulness. You, the coder, supplied the critical value 1.96 in the previous program and we couldn't calculate a P-value. To go further, we have to import additional libraries that contain the stuff we want to use. SciPy, and specifically the stats module, will replace a z-table and then some. Below, we import the SciPy stats module and use it to calculate $P(Z < 1.96)$, using the standard normal CDF.

```
import scipy.stats as stats

z_table_value = stats.norm.cdf(1.96)
print(z_table_value)
```

scipy-norm.py

We can go in the opposite direction with `stats.norm.ppf`.

```
# Two-sided Critical Value
confidence_level = 0.9
alpha = 1 - confidence_level
critical_value = stats.norm.ppf(1 - alpha/2)
print(critical_value)
```

crit-val.py

Occasionally, it's useful to simulate data. SciPy again suits this purpose. For this call, we make use of keyword parameter `size`, specifying that we want 75 iid draws from the standard normal distribution.

```
# Normal random variables
random_draws = stats.norm.rvs(size = 75)
random_draws
```

rand-draws.py

If you display `random_draws`, you'll see we have an array—a new data type.

```
array([ 0.26670047,  0.90571598,  0.58999261, -0.11483403, -0.11317713,
        0.24380578, -1.34662353, -2.38132282, -1.06133942, -0.51280499,
       -0.20017384, -0.99053684,  1.39045452, -0.03411639,  0.86640634,
       -0.58858272, -0.07981447, -1.15327852,  0.75582911,  0.6196648 ,
       -1.21089585,  0.71239498,  0.53601251, -0.60518285,  0.81547665,
        0.87444887,  0.17482955, -0.30222337,  0.70213597, -0.48985895,
        0.58287787,  0.43369743,  0.47753534,  0.4218932 ,  1.28159422,
        0.98054053,  0.47001209,  0.63269663,  1.48985604, -2.17641724,
        0.70551392,  0.65769517, -0.19576411,  0.09185508, -0.53660252,
       -0.15239094,  1.01012891,  0.52824524,  1.28593209, -1.12874119,
        1.4837867 ,  1.11762774, -1.44385417, -0.54467961,  2.1815505 ,
        0.82446247,  1.52427836, -0.89112124,  0.79242734,  0.81785614,
       -0.11150023, -0.24244312,  0.66204106,  1.46663459,  1.30903711,
        1.49169821, -0.36590489, -0.16852676,  2.11081052, -0.45121903,
        0.88317932, -0.55096038,  0.6298471 , -0.1914023 , -0.71709689])
```

We'll sneak in a histogram. The standard data visualization library for Python is matplotlib. We use the pyplot module for a bare-bones histogram in Figure 1.

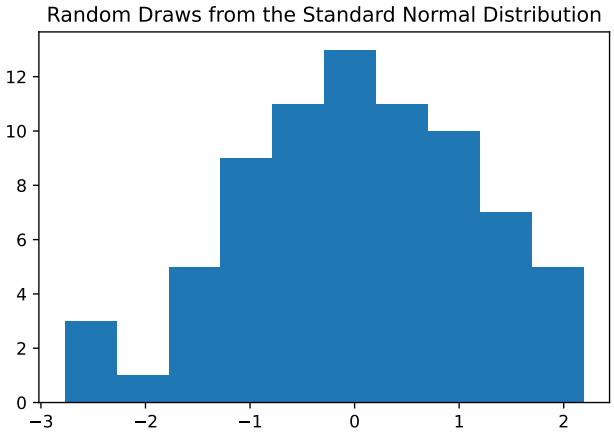

Figure 1: Our first histogram.

```
import matplotlib.pyplot as plt
plt.hist(random_draws)
plt.title("Random Draws from the Standard Normal Distribution")
plt.savefig("first_histogram.pdf")
plt.show()
```

mpl-hist.py

We can simulate Bernoulli draws and run a hypothesis test. Python has a small selection of mathematical functions available apart from any library. Previously, we used `abs()`. Below, we use `sum()` and then `len()` which counts the number of elements in an array.

```
data = stats.bernoulli.rvs(p = .52, size = 30)
p = 0.5 # null hypothesis
p_hat = sum(data) / len(data)
sd = (p*(1-p) / n)**0.5 # st dev from sampling distribution
z = (p_hat - p)/sd # test stat
```

bernoulli.py

We can run many with the help of a special repetition structure called a *for loop*. The code below simulates 30 flips of a fair coin 10,000 times. `k` records how often our test statistic, under a null of $p = 0.5$, would lead us to incorrectly reject the null hypothesis using $\alpha = 0.05$ and a two-sided alternative. What value do you expect for `k/N`?[3]

```
N = 10_000
k = 0

n = 30
p = 0.5 # null hypothesis
sd = (p*(1-p) / n)**0.5 # st dev from sampling distribution

for sim in range(N):
    data = stats.bernoulli.rvs(p = .5, size = n)
    p_hat = sum(data) / len(data)
    z = (p_hat - p)/sd
    if abs(z) > 1.96:
        k += 1

print(k/N)
```

simulation.py

[3] `k/N` should be close to 0.05, using a Normal approximation. Large simulations will show this approximation to be inexact.

4 Libraries for Data

The main library for data wrangling and anything tabular (like a spreadsheet) is pandas. Pandas is famous—it's no wonder the bear is named after this library. Primarily, we will read csv and xlsx files into *DataFrames*. DataFrames can also be constructed by hand like in the program below.

```
import pandas as pd

atus_df = pd.read_csv("ATUS_activity_2019.csv")
other_df = pd.read_excel("helper.xlsx")

data = {'day' : ['Monday', 'Tuesday'],
        'frosty_expenditure' : [0,2],
        'mcflurry_expenditure': [3,0]}
df = pd.DataFrame(data)
```

pandas-df.py

Individual columns can be accessed as a *Series* with `df['frosty_expenditure']` or `df.frosty_expenditure`.[4] We can then create new columns based on existing columns.

```
df['total_expenditure'] = df.frosty_expenditure + df.mcflurry_expenditure
```

new-column.py

DataFrames and Series also have special *methods* available.

```
df = pd.read_csv('file.csv')

mean = df.response_variable.mean()
grouped_means = df.groupby("assignment").response_variable.mean()

var = df.var() # variance of each column
corr = df.corr() # correlation table
```

pd-methods.py

5 Further Resources and Caution

5.1 Resources

I got my start with DataCamp courses. DataCamp uses a subscription model that is affordable. There are also many excellent, free resources available that will help you learn Python. *Python for Data Analysis* by Wes McKinney was updated in 2022, making it one of the most current resources. I also recommend Jake VanderPlas's *Whirlwind Tour of Python* and *Python Data Science Handbook*. For plotting with matplotlib, try my book.

5.2 Caution

Recall when we calculated sample variance as $\sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$. We used $n-1$ instead of n in the denominator as correction for unbiasedness. Not all Python libraries do this by default! NumPy is one such library that does not use the correction by default. The lesson here is that, for this class, do your research if you are using other libraries and functions. There are many ways to do what sounds like the same thing in statistics and some disjointedness across all of the Python libraries.

Below is another example where default behavior in the StatsModels library does not match our intro conventions. For now, our calculations are simple enough that I don't recommend finding random libraries and functions to shorten your code.

[4] This pattern will not work if there is a space in the column name or if the column doesn't adhere to other naming requirements.

```python
from statsmodels.stats.proportion import proportions_ztest

# Hypothesis testing with null of p = 0.6

# Wrong/Different
z1, pval1 = proportions_ztest(10, 20, value = .6)

# Correct/Matches Textbook
z2, pval2 = proportions_ztest(10, 20, value = .6, prop_var = .6)

# By hand
p_hat = 0.5
p = .6
n = 20
sd = (p*(1-p) / (n))**0.5 # st dev from sampling distribution
z3 = (p_hat - p)/sd # test stat
```

prop-ztest.py