Python Supplement to 410

A general guide to using Python models where possible in STAT 410.

Keanu Hua
A gift produced for Applied Statistics students.

Department of Mathematics and Statistics
California State University
United States
October 8, 2023

Acknowledgments

This project would not have been possible without Google, stackexchange, and countless online guides. While ChatGPT probably could have found a lot of the information, I was too lazy to make an account. I would also like to thank my bizarre online mentor, Dianoia, a Malaysian PhD candidate in computer science who I met online playing a really bad probability-based game. He was the one who suggested Miniconda, VSCode, and the digression on virtual environments, which took a bit for me to understand.

As for CSULB influences, I would also like to thank Dr. Olga Korosteleva for the source codes and maths that I based much of this on, plus our newcomer-as-of-2023 Dr. Seungjoon Lee for his very brief code review. I mainly asked him just to make sure I wasn't doing anything egregious, and I apparently wasn't.

1 Introduction

Python is one of the most popular programming languages for its ease of use, and this has translated to its popularity in data science, data analysis, and machine learning circles. While R can do much of the same, what is undeniable is that both are popular, and that for the purpose of the ever-inscrutable job market, you should have familiarity with the syntax and basics of both. Besides

2 Setting up Python

While you could set up Python using just the CECS 174 expectations, there's ways to optimally set up a workspace to mitigate issues and hasten workflow.

2.1 Miniconda

You may have already encountered this as Anaconda, but Miniconda does most of what's necessary at a much smaller footprint. The -conda programs manage packages, among other functions. Miniconda's installation is simple enough and can be found on their site, here: `https://docs.conda.io/projects/miniconda/en/latest/miniconda-install.html/`

Once conda has been installed successfully, an environment can be developed as follows, with square brackets denoting replaceable variables:

1. Open powershell.
2. `conda create --[environment name] [packages]`
3. Environment name can be as you please. Packages should include (in easy copy-paste format): pandas statsmodel scipy numpy
4. Proceed.
5. `conda activate [environment name]`

Issues that I had, and their corresponding fixes, are also listed below:

If you still have Python from CECS 174, there is a possibility that Windows won't immediately register the newest Python/Conda version. If this happens, search for Advanced System Settings, then go to Environment Variables. Find the variable labeled Path and click edit, then scroll down. Check that miniconda3\scripts, miniconda3, and miniconda3\Library\bin are all there. Taken from `https://stackoverflow.com/questions/44515769/conda-is-not-recognized-as-internal-or-external-command`.

If you get an error about "running scripts is disabled on this system," run Powershell as an administrator. Then, input Set-ExecutionPolicy -Scope CurrentUser and then Set-ExecutionPolicy RemoteSigned. Taken from `https://stackoverflow.com/questions/4037939/powershell-says-execution-of-scripts-is-disabled-on-this-system`.

2.1.1 Environments

Computers are very complicated machines. While you can technically run a lot in a single base Python installation, having so many moving parts means that if some random thing breaks, you can end up spending hours trying to hunt down an obscure error.

Environments mitigate this issue. Creating isolated environments allows you to freeze the versions of the packages and Python used to construct the model. Below are a few instances where environments are crucial:

Incompatible packages: Some packages will just conflict with one another. Having an environment for one and a different one for another allows easy switching back and forth.

Updates: Updates can break old code.

Conflicting versions: Suppose you need Package C to run Package A and Package B, but Package A wants Package C 1.1 and Package B wants Package C 1.2. You can't have two different versions of Package C, but you need both packages to accomplish 2 separate, concurrent tasks. With 2 environments for both packages, this wouldn't be an issue.

2.2 Recommended packages

The following packages are used. To download a package, use `conda install packages` in the conda terminal. This can support one or several packages, with each package having internal modules as well:

pandas: Creates and manipulates dataframes.

statsmodel: Hosts several regression models. statsmodel.formula.api module allows easier typing and development for regression models.

scipy: scipy.stats module is used for the Anderson-Darling and Shapiro-Wilks tests, as well as the chi^2 cdf for deviance testing.

numpy: Required for computations with Box-Cox and Pandas.

matplotlib: For histograms.

2.3 VSCode

A basic IDE like Python IDLE is usable, but VSCode is generally better. It is highly recommended that you set it up:

1. Through Jupyter notebook, VSCode can isolate certain functions. This means you don't need to run an entire program just to test one change or because of a single typo.

2. VSCode allows third-party extensions with myriad uses. Some of these are quality-of-life or tedious task automation, while others integrate other software like SQLite, SAS (which is slightly buggy), and R.

3. Since VSCode can aggregate other code software through extensions, you can use the same sets of hotkeys in one program.

4. VSCode has autocomplete. It takes a bit of getting used to, but it has autocomplete for SAS, which even SAS doesn't have!

5. VSCode allows you to select which environment you want to use.

6. If variables exist, VSCode lets you find them, their type, their size, and their value.

7. Various text editing supports, such as Alt + LClick allowing you to select multiple values.

8. You can download themes and make your VSCode look very pretty.

3 The actual code

Note that all items in allcaps are generic variables that can be freely replaced.

3.1 Initialization

The first step is to summon all of the libraries using import. Note that not all regressions will use every library:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
import matplotlib
import scipy
import numpy as np
import math
```

Then, call the data with PANDAS. DATANAME.head(0) lists out the variable names for usage later. Keep track of your RESPONSE and PREDICTOR variables — they will be used later.

```
DATANAME = pd.read_csv('DATA FILE PATH')
DATANAME.head(0)
```

If necessary, data can also be scaled/modified with the following format. Any arithmetic can be performed on the right side of the equation to scale/modify the original data as needed. Note that using the same variable names will REPLACE the variables, so feel free to use a different variable name if yoju'd like to keep the original.

```
DATANAME['VARIABLE'] = DATANAME['VARIABLE']
```

3.2 Normality

```
# Sets the y-value to the y-value[name] for normality tests.
y_val = DATANAME["RESPONSE"]

from matplotlib import pyplot
# Summon histogram. Uses matplotlib.
pyplot.hist(y_val)
pyplot.show()

# Uses scipy.stats for an Anderson-Darling test.
from scipy.stats import anderson
# Assign result from an Anderson-Darling test using the y-value.
result = anderson(y_val)
# Dump out result's test statistic.
print(f'Anderson-Darling test statistic: {result.statistic:.3f}')

# Uses scipy.stats for a Shapiro-Wilks test.
from scipy.stats import shapiro
# Using the y-value, perform a Shapiro-Wilks test. Prints test statistic and
    p-value rounded to 3 decimals.
statistic, p_value = shapiro(y_val)
print(f"""Shapiro-Wilks test statistic: {statistic:.3f}
Shapiro-Wilks p-value: {p_value:.3f}""")
```

3.2.1 Box-Cox

Box-Cox transformation is unique in the sense that it's the only regression model that openly requires its user to transform the response variable significantly. Note that the automatic boxcox transformation from scipy is not useful; we still want to use the rounded-off values.

```
from scipy.stats import boxcox
DATANAME['RESPONSE VARIABLE'], fitted_lambda = boxcox(DATANAME['RESPONSE
    VARIABLE'])
print(f'Lambda is {fitted_lambda}')
y_val = DATANAME["RESPONSE VARIABLE_tr"]

# for later usage
DATANAME['RESPONSE VARIABLE_tr'] = BOXCOX TRANSFORMATION
print(DATANAME)

# Normality tests and histogram
result = anderson(y_val)
print(f'Anderson-Darling test statistic: {result.statistic:.3f}')
statistic, p_value = shapiro(y_val)
print(f"""Shapiro-Wilks test statistic: {statistic:.3f}
```

```
Shapiro-Wilks p-value: {p_value:.3f}""")

pyplot.hist(y_val)
pyplot.show()
```

3.3 Fitted and null models

Note that, in the below models, family will vary. For now, we're using a general linear model; examples will be enclosed for other models.

```
# Develop model.
formula = 'RESPONSE ~ PREDICTORS'
fitted = smf.glm(formula=formula, data=DATANAME,
    family=sm.families.Gaussian()).fit()
print(fitted.summary())
fitloglike = (fitted.llf)
print(f'Sigma of fitted model is {np.sqrt(fitted.scale)}.')

# Null model.
formula = 'RESPONSE ~ 1'
null = smf.glm(formula=formula, data=DATANAME,
    family=sm.families.Gaussian()).fit()
nullloglike = (null.llf)
```

3.4 Deviance test

You will need degrees of freedom for this. The rest of this is automated because it pulls variables from the models.

```
# Uses null and fitted log likelihoods to perform the deviance test.
deviance= -2 * (nullloglike-(fitloglike))
print(f"Deviance statistic is {deviance}.")

# Chi2.cdf is from scipy.stats.
from scipy.stats import chi2
pvalue = 1 - chi2.cdf(deviance,DEGREES OF FREEDOM)
print(f"p-value is {pvalue}.")
```

3.5 Predictions

Note that character entries must be enclosed by apostrophes. Naturally, the number of predictors can be extended.

```
predict_val = pd.DataFrame(
    {"PREDICTOR 1" : 'CHARACTER VAL', "PREDICTOR 2" : NUMERIC_VAL,
        "PREDICTOR N" : VAL,}, index=[0])
```

```
predict_val = sm.add_constant(predict_val)
fitted.predict(predict_val)
```

4 Citations

4.1 General

Statsmodel formula: https://www.statsmodels.org/dev/examples/notebooks/generated/glm_formula.html

Normality: https://machinelearningmastery.com/a-gentle-introduction-to-normality-tests-in-python/

Finding names of objects via dir(): https://stackoverflow.com/questions/2675028/list-attributes-of-an-object

Prediction: https://www.statology.org/statsmodels-predict/

4.2 Box-Cox, Gamma

Box-cox: https://www.geeksforgeeks.org/box-cox-transformation-using-python/

4.3 Logit, Probit, Cloglog

Cloglog format: https://github.com/statsmodels/statsmodels/issues/827

Model comparisons in Logit/Probit/Cloglog: https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/

Automatic comparison: https://www.geeksforgeeks.org/get-minimum-values-in-rows-or-columns-with-their-index-position-in-pandas-dataframe/