

## QMSS5019 Data Analysis with Python - Summer 2021 - Cheat Sheet

Purpose of this cheat sheet is to share a few useful tips / highlights for Python code in this course. Lectures, recitations and lab documents are sufficient for the course and all the assignments. This document is definitely not comprehensive for the content in this course and is completely optional if you'd like to have a look at it anytime.

### **Tips on Where to Code:**

We will primarily work with Jupyter Notebook on your PC. Instructions are provided in Week1 course material. If you have troubles with Jupyter setup at any point, check out [Anaconda Navigator](#), usually it is the most helpful resource.

You can use [Google Colab](#) (simply Google docs for coding) to access your notebooks anywhere and work with others collaboratively. You need to enable Google Colab add-on through your Google account if you never used it. It's for free!

### **Tips on Reading Files:**

Jupyter Notebook: You can set a working directory, simply telling your notebook "where to look for all files" and then specify file names. See the code example below. This is especially a useful shortcut specifying where your notebook should look if you need to read multiple files.

```
os.chdir('C:/Users/...') #set directory by adding your file path between ''
d = pd.read_csv("GSS.2006.csv") #read the CSV file from the directory
```


Google Colab: Simplest way is to refer to a URL that the file is available at. People mostly refer to files through Github or Google Drive. Each method has a certain setup. [This link explains them all](#) well and [this link](#) has more advanced ways. I usually prefer choosing the file manually from my PC following the code below. If you lock your PC for a long time, you'll need to re-read the file and re-run the code. This is usually not a big concern for the size of our notebooks.

```
from google.colab import files
uploaded = files.upload()
```

### **Python Basics:**

A great cheat sheet (cheat notebook actually, since it is 26 pages :D) is available [on this link](#) with a more helpful resources guide [on this link](#).

Adding comments is a useful practice to add explanations in your code. In Python, we add the symbol # before comments, which means that the following line is not interpreted as a piece of code.

Beginner's Python Cheat Sheet		
<b>Variables and Strings</b> <i>Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.</i> <b>Hello world</b> <pre>print("Hello world!")</pre> <b>Hello world with a variable</b> <pre>msg = "Hello world!" print(msg)</pre> <b>Concatenation (combining strings)</b> <pre>first_name = 'albert' last_name = 'einstein' full_name = first_name + ' ' + last_name print(full_name)</pre>		
<b>Lists</b> <i>A list stores a series of items in a particular order. You access items using an index, or within a loop.</i> <b>Make a list</b> <pre>bikes = ['trek', 'redline', 'giant']</pre> <b>Get the first item in a list</b> <pre>first_bike = bikes[0]</pre> <b>Get the last item in a list</b> <pre>last_bike = bikes[-1]</pre> <b>Looping through a list</b> <pre>for bike in bikes:     print(bike)</pre> <b>Adding items to a list</b> <pre>bikes = [] bikes.append('trek') bikes.append('redline') bikes.append('giant')</pre> <b>Making numerical lists</b> <pre>squares = [] for x in range(1, 11):     squares.append(x**2)</pre>		
<b>Lists (cont.)</b> <b>List comprehensions</b> <pre>squares = [x**2 for x in range(1, 11)]</pre> <b>Slicing a list</b> <pre>finishers = ['sam', 'bob', 'ada', 'bea'] first_two = finishers[:2]</pre> <b>Copying a list</b> <pre>copy_of_bikes = bikes[:]</pre>		
<b>Tuples</b> <i>Tuples are similar to lists, but the items in a tuple can't be modified.</i> <b>Making a tuple</b> <pre>dimensions = (1920, 1080)</pre>		
<b>If statements</b> <i>If statements are used to test for particular conditions and respond appropriately.</i> <b>Conditional tests</b> <pre> equals      x == 42 not equal   x != 42 greater than x &gt; 42 or equal to x &gt;= 42 less than   x &lt; 42 or equal to x &lt;= 42 </pre> <b>Conditional test with lists</b> <pre>'trek' in bikes 'surly' not in bikes</pre> <b>Assigning boolean values</b> <pre>game_active = True can_edit = False</pre> <b>A simple if test</b> <pre>if age &gt;= 18:     print("You can vote!")</pre> <b>If-elif-else statements</b> <pre> if age &lt; 4:     ticket_price = 0 elif age &lt; 18:     ticket_price = 10 else:     ticket_price = 15 </pre>		
<b>Dictionaries</b> <i>Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.</i> <b>A simple dictionary</b> <pre>alien = {'color': 'green', 'points': 5}</pre> <b>Accessing a value</b> <pre>print("The alien's color is " + alien['color'])</pre> <b>Adding a new key-value pair</b> <pre>alien['x_position'] = 0</pre> <b>Looping through all key-value pairs</b> <pre>fav_numbers = {'eric': 17, 'ever': 4} for name, number in fav_numbers.items():     print(name + ' loves ' + str(number))</pre> <b>Looping through all keys</b> <pre>fav_numbers = {'eric': 17, 'ever': 4} for name in fav_numbers.keys():     print(name + ' loves a number')</pre> <b>Looping through all the values</b> <pre>fav_numbers = {'eric': 17, 'ever': 4} for number in fav_numbers.values():     print(str(number) + ' is a favorite')</pre>		
<b>User input</b> <i>Your programs can prompt the user for input. All input is stored as a string.</i> <b>Prompting for a value</b> <pre>name = input("What's your name? ") print("Hello, " + name + "!")</pre> <b>Prompting for numerical input</b> <pre>age = input("How old are you? ") age = int(age)  pi = input("What's the value of pi? ") pi = float(pi)</pre>		
<div> <div> <b>Python Crash Course</b>  Covers Python 3 and Python 2  <a href="http://nostarchpress.com/pythoncrashcourse">nostarchpress.com/pythoncrashcourse</a> </div>  </div>		

## Importing Packages:

We need to refer to packages whenever we use a function or method defined in that package. Below is an example:

```
import pandas as pd # we import 'pandas' and will refer to it as "pd" in our code. Giving a short name is helpful since we will have long names like statsmodels.formula.api package
```

## Week1 (Descriptive Statistics)

Examine a column of your data frame: Use describe() function. Insert dots between the data frame, column name and the function. See the code below.

```
summary = data_frame.column_name.describe() #you can call it anything instead of summary
```

Look at descriptive statistics for 2 columns in comparison:

```
d.groupby([column1])['columns2'].median() # you can also look at other statistics e.g. std(), mean(), count()...
```

Define and apply helpful transformation functions:

Below is the basic structure to define any function that does “certain transformations” on a variable e.g. calculations, formatting...

```
def function_name(variable_name):  
    output = ..... (certain transformations with variable_name)  
    return output
```

Rename Columns:

```
your_data_frame.rename(columns={'original_name':'newname',...},  
inplace=True)
```

Basic structure to apply a function on a variable or dataframe:

```
your_input.apply(function_name, axis=...)
```

Note: ‘Axis’ argument enables you to apply a function for entire columns / rows of a dataframe.

**axis{0 or ‘index’, 1 or ‘columns’}, default 0**

Axis along which the function is applied:

- 0 or ‘index’: apply function to each column.
- 1 or ‘columns’: apply function to each row.

Boxplot: A visualized version of descriptive statistics to see how variables are distributed:

<https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51>

## Week 2 (Recoding Variables):

Choose a certain list of variables depending on a condition:

`numpy.select`(*condlist*, *choicelist*, *default=0*)

Return an array drawn from elements in choicelist, depending on conditions.

### **Parameters:**

**condlist**list of bool ndarrays

The list of conditions which determine from which array in choicelist the output elements are taken. When multiple conditions are satisfied, the first one encountered in condlist is used.

**choicelist**list of ndarrays

The list of arrays from which the output elements are taken. It has to be of the same length as condlist.

**default**scalar, optional

The element inserted in output when all conditions evaluate to False.

### **Returns**

**output**ndarray

The output at position m is the m-th element of the array in choicelist where the m-th element of the corresponding array in condlist is True.

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.div.html>

`DataFrame.div`(*other*, *axis='columns'*, *level=None*, *fill\_value=None*)

Get Floating division of dataframe and other, element-wise (binary operator truediv).

Equivalent to dataframe / other, but with support to substitute a fill\_value for missing data in one of the inputs. With reverse version, rtruediv.

Among flexible wrappers (add, sub, mul, div, mod, pow) to arithmetic operators: +, -, \*, /, //, %, \*\*.

### **Parameters:**

**other**scalar, sequence, Series, or DataFrame

Any single or multiple element data structure, or list-like object.

**axis{0 or 'index', 1 or 'columns'}**

Whether to compare by the index (0 or 'index') or columns (1 or 'columns'). For Series input, axis to match Series index on.

**levelint or label:**

Broadcast across a level, matching Index values on the passed MultiIndex level.

**fill\_value: float or None, default None**

Fill existing missing (NaN) values, and any new element needed for successful DataFrame alignment, with this value before computation. If data in both corresponding DataFrame locations is missing the result will be missing.

**Returns: DataFrame**

Result of the arithmetic operation.

Note: We will use cross tabulation of 2 values very frequently. See example below

```
your_cross_tab= pd.crosstab(data_frame.var1, data_frame.var2)
your_cross_tab.astype('float').div(res.sum(axis=0), axis=1)
```

### **Week 3**

- **Getting rid of “NA” values:** Skip respondents / data points that do not have data for your variable of interest. If included, NA values may skew the result of your analysis. You can create a new dataset excluding entries that have NA value for your variable of interest as in the following code: `new_dataframe = old_dataframe.dropna(subset = ["variable_of_interest"])`
- **Subsetting the data:** If you are interested in only a subset of your data e.g. people from a country, you can subset the data within OLS model as in the following code:  
`lm1 = smf.ols(formula = 'dependnt_variable ~ independent_variables', data = d, subset = (d['variable_of_interest']==value) ).fit()`
- **Turn categorical variables into dummy variables:** We need numerical variables in regression analysis. How can we include categorical variables that are only text e.g. country, region... We can turn them into “dummy” numerical variables. Simply, each category would stand for a number and we would interpret it based on which number stands for which category. We can do this via the following code:  
`pd.get_dummies(data_frame.variable_of_interest.astype(int), prefix='name_of_variable')`

- **ANOVA and F-test:** F-tests look for whether there is a meaningful difference in the variance of two different groups. We can use this logic by applying ANOVA analysis. ANOVA will tell you whether two different linear models yield meaningfully different results. In other words, ANOVA test tells you whether a new linear model is any better / different than an earlier model. [This link explains F-test and ANOVA in detail.](#)

## **Week 4 (interactions):**

**Interaction Variables:** Certain independent variables may have an internal relationship that may change the dependent variable. For example, model1 below looks at a and b within a linear model. Model2 looks at the interaction between a and b, how different levels of a may cause a difference in b, which would influence the result of your model (your dependent variable). [More detailed explanation here!](#)

- `model1=ols(formula = 'dependent_variable ~ a+b`
- `model2=ols(formula = 'dependent_variable~ a*b`

**Factoring categorical variables:** You can look into different values of a categorical variable by introducing it as “factors” in your model. For example, if *country* had been an integer variable that we wanted to treat explicitly as categorical, we could have done so by using the `C()` operator as in the following code: `ols(formula = 'thirtyyroid ~ age + C(country)`