Introduction to Python: Workshop 1

Content created by Francesca Pontin [francescapontin.com (http://francescapontin.com/)](http://francescapontin.com/)

Source: [Reddit (https://www.reddit.com/r/ProgrammerHumor/comments/9c55gd/python/)](https://www.reddit.com/r/ProgrammerHumor/comments/9c55gd/python/)

Welcome to coding in Python! These workshops are designed to introduce you to the basics of coding in Python then get you to apply your new found coding skills to carry out some data analytics! This workshop is written in a jupyter notebook. A Jupyter Notebook ["is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text" (https://realpython.com/jupyter-notebook-introduction/)](https://realpython.com/jupyter-notebook-introduction/). A Jupyter notebook is made up of cells (the grey boxes). These cells can be text (known as markdown), code or visulisations/images, to name a few. To select a cell click anywhere in the box and a large blue box will appear arround it. To edit a cell click in it twice and the surrounding box will turn green.
As shown bellow.

> Welcome to coding in Python! These workshops are designed to introduce you to the basics of coding in Python then get you to apply your new found coding skills to carry out some data analytics! This workshop is written in a jupyter notebook. A Jupyter Notebook "is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text". A Jupyter notebook is made up of cells (the grey boxes). These cells can be text (known as markdown), code or visulisations/images to name a few. To select a cell click anywhere in the box and a large blue box will appear arround it. To edit a cell clik on it twice and the surrounding box will turn green. As shown bellow.

Welcome to coding in Python! These workshops are designed to introduce you to the basics of coding in Python then get you to apply your new found coding skills to carry out some data analytics!
This workshop is written in a jupyter notebook. A Jupyter Notebook "is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text". A Jupyter notebook is made up of cells (the grey boxes). These cells can be text (known as markdown), code or visulisations/images to name a few.
To select a cell click anywhere in the box and a large blue box will appear arround it. To edit a cell clik on it twice and the surrounding box will turn green. As shown bellow.

During this workshop you will read through the jupyter notebok, running sections of code yourself to learn key data analysis skills. There will also be places to edit and write your own code within the notebook.

Instructions and tasks for you to complete are in purple

How to run a cell of code

To run code select the cell you want to run (the blue box will appear arround the selected cell) and the EITHER:

1) type CTRL + ENTER on your key board

OR

2) click Run on the jupter notebook the menu above

How do I know if I have run the cell?

If the cell has been run a number will appear in squared brackets by the cell e.g. `In [1]: or In [12]:`

The numbering refers to the order in which you have run the cells

An un-run cell has an empty set of squared brackets by the cell e.g. `In []:`

How do I add a new cell?

To add a new cell select a cell (so it is surrounded by a blue box) and type 'b' to add a cell bellow or 'a' to add a cell above.

To delete a cell select it and double tap 'd'.

Hello World

In coding tradition the first thing we are going to program is to print the words "Hello World".

The `print()` function prints the specified message to your screen

The speech marks `" "` arround Hello World let python know you are typing text (also known as a string).

Run the code below

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

Basic maths

Lets do some basic maths
Run the code below

In [ ]:
```
2+2
```

If we print 2+2 we get the same thing
Run the code below

In [ ]:
```
print(2+2)
```

If we `print("2+2")` however we get the characters 2 + 2 as the speach marks tell Python we are typing text and not numerical characters

Run the code below and see for yourself

In [ ]:
```
print("2+2")
```

We can also assign the sum 2+2 to a variable.

In this case we have named the variable answer. We can now type answer instead of 2+2 every time.

To assign 2+2 to the variable answer run the code below

In [ ]:
```
answer = 2+2
print(answer)
```

In [ ]:
```
# we can also add our created variable to a new equation
answer + 5
```

Importing Packages

Python has a lot of basic functionality built in e.g. the maths we have just done, but a lot of the time while doing data analysis you will require more than the basic funcitonality. This is whe you need to import packages. If you have seen anyone elses code before you will have seen people tend to install all their packages at the begining of their code.

To import a package use the statement `import` followed by the `package_name` .

Run the code below to import the pandas package.

In [ ]:
```
import pandas
```

The pandas package

The pandas package allows us to easily create and handle dataframes, simialr to excel. Data is put into an easily readable format of columns and rows.

It also allows us to read in data from a CSV file or excle file.

When we use the package we refer to it for example `pandas.read_csv()`

Note on abbreviating packages

To save us having to type out pandas everytime we can abbreviate pandas to pd using the code bellow (**Run the code below**).

This is often done for commonly used packages to save time and redce the likelihod of typos while coding.

So `pandas.read_csv()` becomes `pd.read_csv()` .

In [ ]:
```
import pandas as pd
```

Now let's install some of the packages we will need for the rest of the workshop

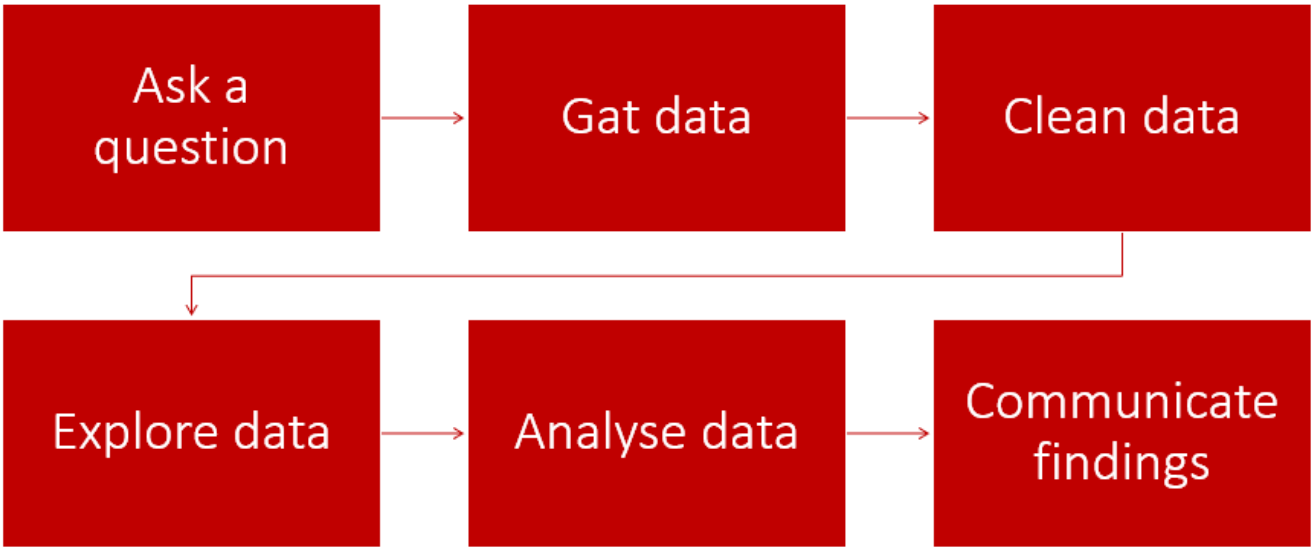
Run the code below

In [ ]:
```
# Import other required packages

# packages for visulising data
import matplotlib.pyplot as plt
import seaborn as sns
```

The data science process

In this introduction to python workshop we are going to follow the basic data science/ analysis process (Sarah introduced this process flow in our intro to data science talk earlier in the year). We will introduce you to some basic data manipualtion, data cleaning and visulisation skills in python this week. In the second workshop we will look at some methods of data analysis and more complex visulisations.



Reading in data

For this workshop will be using some data I have fabriacated about buying coffee (my favourite thing to do). Before we get started with any analysis we need to import the data.

The data is originally in a CSV (comma seperated variable) format. To read the data in will use the pandas package introduced above. Reading the data into a dataframe using the `pd.read_csv()` function.

The `pd.read_csv()` function requires a 'filepath' i.e. we need to tell python where to find the data. To do this we put the filepath in speechmarks within the brackets of the function. e.g.
`pd.read_csv("file_path")`

There is a quick note about file paths bellow if these are new to you
Run the code below

In [ ]:
```
coffee = pd.read_csv("Intro_to_Python/coffee_data.csv")
```

So what have we done in the above code?

We have told pandas to read in a csv file and provide the path to where the CSV file can be found; saved in a folder within my documents.

We have also told pandas to assign the dataframe the name 'coffee' It is useful to have logical dataframe names rather than df1, df2 etc so you know what you are referring to when you come back to your code.

Exploring the data

Data type

We can check that the coffee data has been stored in a dataframe by using the function `type()`
Run the code below

In [ ]:
```
type(coffee)
```

Look at the dataframe

So what does the coffee dataframe look like? By typing coffee we can see the dataframe in full
Run the code below

In [ ]:
```
coffee
```

With big dataframes it is easier to just look at the top few lines. To do this we type `dataframe_name.head()` which gives us the first 5 lines of the dataframe as well as the column headings.
Run the code below

In [ ]:
```
coffee.head()
```

Alternatively we can use tail to see the last 5 lines of the table. Do the same thing but replace `.head()` with `.tail()`
Try this in the cell bellow.

In [ ]:
```

```

This is what you should see

	coffee_name	coffee_type	coffee_shop	price	cake_deal	rating
19	pret_latte	latte	pret	3.1	True	3
20	uni_coffee_americano	americano	uni_coffee	2.1	False	2
21	uni_coffee_mocha	mocha	uni_coffee	2.4	True	3
22	uni_coffee_latte	latte	uni_coffee	2.4	True	2
23	uni_coffee_tea	tea	uni_coffee	0.8	False	1

Data Types

There are different types of data:

Objects: also known as strings or written characters/text in plain english e.g. Hello World

Intergers: Numeric characters e.g. 2

Floats: Numeric characters with decimals e.g. 2.34534

Datetime: Values that are either a date, time or both e.g. 2019-10-31 09:26:03.478039 (9:26 am on Halloween 2019)

Boolean: True or False data type

Learn more about python data types using this realpython online resource (https://realpython.com/python-data-types/)

We can check the data type of each column in a dataframe using the `.dtypes` funciton
Run the code below

In [ ]:
```
coffee.dtypes
```

We can also explore the different columns and get some basic summary statisitcs by using the function `.describe()`
This is only possible for columns with a numeric (interger or float) data type.
Run the code below

In [ ]:
```
coffee.describe()
```

Referring to a column in a dataframe

If we want to sunamrise a single column of the dataframe we can also do that.

There are several ways to refer to a column in a pandas dataframe.

The easiest way is by putting the name of the column in square barckets and speech marks `[" "]` after the name of the dataframe. e.g. `dataframe_name["column_name"].describe()`

In [ ]:
```
coffee['price'].describe()
```

We can also just get one of the metrics from the `describe()` function e.g. `.mean()`, `.quantile(0.75)`, `.max()`

Mean

In [ ]:
```
coffee['price'].mean()
```

Median

In [ ]:
```
coffee['price'].median()
```

Quantile

In [ ]:
```
# Get the 75% quantile
coffee['price'].quantile(0.75)
```

Write your own code below to get the 25% quantile

In [ ]:
```
# Get the 25% quantile

```

Maximum and Minimum

In [ ]:
```
# get the maximum coffee price
coffee['price'].max()
```

Write your own code below to get the cheapest coffee price

In [ ]:
```
# get the minimum coffee price

```

Looking at the result of `coffee['price'].max()` £20 seems a bit expensive for a coffee- lets have a look at all the coffee prices using the `print()` function.

In [ ]:
```
list(coffee['price'])
```

Data Cleaning

It seems that £20.00 probably has the decimal in the wrong place. £2.00 seems a much more sensible price for coffee. This is an example of a fairly common error, particualrly if the data has been entered by a human.

We have identified the first bit of data that needs cleaning!

There are several ways we can do this

1) Re-wrtie the list using `[]` with each variable seperated by a comma. We are going to assign the cleaned coffee variables to a new column called 'price_clean1' so we can compare columns later.
Run the code below

In [ ]:
```
# rewrite the coffee as a list with the correct value in place
# and use = to asign this to coffee['price_clean1']
coffee['price_clean1']= [2.4, 2.6, 2.5, 2.0, 2.5, 2.7, 2.75, 3.0, 2.9, 1.8, 3.0, 2.5, 3
# print to check
print(coffee['price_clean1'])
```

In [ ]:
```
# create another 2 duplicates of column price and name them price_clean2, price_clean3
coffee['price_clean2'] = coffee['price']
coffee['price_clean3'] = coffee['price']
```

Typing out the whole list of coffee prices is quite time consuming, we wouldn't be able to do it if we had 100s of rows of data. Instead we can identify which row the incorrect value is in and correct just that value.

In [ ]:
```
print(coffee['price'])
```

We can see that the £20.00 coffee is in row 15. Note that the row indexing starts at zero (not one) in python.

We can use the `dataframe_name.loc` attribute of pandas dataframes to access rows and/or columns of the dataframe we want to edit.

The general format of loc is shown by the code bellow. The first item in the squared brackets identifies the row(s) of interest and the second item (after the comma) identifies the column(s) of interest

```
dataframe_name.loc[row label,column_lable]
```

NOTE: to select all rows or all columns use a colon `:`

Run through the next few cells and make sure you understand how `.loc` is working to select the different columns and rows

In [ ]:
```
# select all rows and column price_clean2
coffee.loc[:,"price_clean2"]
```

In [ ]:
```
# select row 5 and all columns
coffee.iloc[[5],:]
```

In [ ]:
```
# select row 3-7 and columns 'price' and 'rating'
coffee.loc[[3,4,5,6,7],['price','rating']]
```

In [ ]:
```
# using iloc we can use the index of the columns as well.
# select rows 0-3 and columns 0-4
coffee.iloc[[0,1,2,3],[0,1,2,3,4]]
```

In [ ]:
```
# check the location of the incorrect data is row 15 in column price_clean2
coffee.loc[[15],"price_clean2"]
```

In [ ]:
```
# assign the incorrect data the correct value
coffee.loc[[15],"price_clean2"] =2.0
print(coffee['price_clean2'])
```

The above two methods are great if only a few values need cleaning or the dataset is small enough to easily find the location of the incorrect data. However for large datasets these methods are impracticle and applying a more general rule would be better.

We can hypothesis that any coffee over about £8 probably has had the price incorrectly entered and that the decimal point should be moved. Therefore we can find every row where the price is over £8 and divide it by 10 (effectively moving the decimal place back a digit).

Run the code below

Locate where coffee price is greater than or equal to £8: `coffee.loc[coffee['price']>=8]`

In [ ]:
```
# is coffee price greater than or equal to £8.
# gives a boolean (true/false) answer
coffee['price']>=8
```

In [ ]:
```
# use .loc to identify the row where coffee price is >= £8
coffee.loc[coffee['price']>=8]
```

Now back to the cleaning

Run the code bellow, using the comments to work out what each bit of the code is doing, ask if you have any questions

In [ ]:
```
# in rows where coffee price is >= £8,
# in column 'price_clean3' calcualte the cleaed price of coffee
# by dividing the original coffee['price'] by 10

coffee.loc[coffee['price']>=8, 'price_clean3'] = coffee['price']/10
# view column price_clean3
print(coffee['price_clean3'])
```

Let's check our cleaning

Lets check and compare our cleaning methods We can see the original price of £20.00 and then how each method has cleaned the data to £2.00
Run the code below

In [ ]:
```
# show row 15 of the datrframe and the price columns
coffee.loc[[15],['price','price_clean1','price_clean2','price_clean3']]
```

Visualising data

Often the best way to understand the data you have is to visualise it

Histograms

We will plot two histograms to see the distribution of data before and after cleaning.
The simplest way to do this is using the matplotlib `.hist()` function.
Run the two cells below

In [ ]:
```
coffee['price'].hist();
```

In [ ]:
```
coffee['price_clean1'].hist();
```

Scatterplots and correlations

Use the matplotlib pyplot `.scatter()` function to create a scatter plot.
Whilst a histogram is 1 dimensonal (takes one setof data), scatter plots plot two lots of data (often columns) against each other.

We will use the scatter plot to see if there is a correlation between price and rating, specifying price as the x-axis variable and rating as the y-axis variable
Run the code below

In [ ]:
```
coffee.plot.scatter(x='price_clean3',y='rating');
```

A quick note on graph customisation

There are lots of different ways to edit graphs to get them looking how you want

- `s=` : scalar i.e. point size
- `c=` : colour of the points, see the list of matplotlib colours below. colours need to be in speech marks e.g. `"red"`

- `alpha=` : How transparent the point is, 0-1 (completely transparent- solid colour)
- `marker=` : Defines how the point looks, there are a few examples below and more here (https://matplotlib.org/3.1.1/api/markers_api.html#module-matplotlib.markers)

marker	symbol	description
"."	●	point
","	·	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	Y	tri_down
"2"	⅄	tri_up
"3"	≺	tri_left
"4"	≻	tri_right
"8"	●	octagon
"s"	■	square

Play arround with the code below, changing different elements

In [ ]:
```
coffee.plot.scatter(x='price_clean3',y='rating', s=20, c='green', alpha=0.3);
```

Plot your own scatter plot of price against rating with pink triangular points, scalar = 75 and make the points semi-transparent

In [ ]:
```

```

Visulisations using Seaborn

The seaborn package can also be used to visulise data. Often producing cleaner and clearer looking plots along with statisitical graphics.

`sns.jointplot()` is used to plot two variable with seaborn. Notice it also prduces a histogram of the variable distibutions of each axis.

Like with matplotlib elements of the graph are editable

`kind=` : Can be “scatter” | “reg” | “resid” | “kde” | “hex”

Run the code bellow and then try changing the kind of graph plotted

In [ ]:
```
sns.jointplot(x='price_clean3',y='rating', kind ='scatter',data=coffee);
```

In [ ]:
```
# plot a kde
```

In [ ]:
```
# plot a reg plot
```

In [ ]:
```
# plot a kde
```

In [ ]:
```
# plot a residuals plot
# these are used in statistical testing don't worry if you don't recongise then (we are
```

Different editable parameters with Seaborn

To edit the seaborn plot, the parameters you specify are slightly different to matplotlib.

You will notice with seaborn that the `x` and `y` specify only the column name and `data` is used to specify the dataframe where the `x` & `y` columns can be found.

Other editable parameters:

- `color=` colour of the points, see the list of matplotlib colours. Colours need to be in speech marks e.g. `"red"`
  Note the American spelling, using 'colour' will produce an error
- `height=` Size of the figure (it will be square), enter a numerical value.
- `marker=` Same as matplotlib

Read more about the editable parameters in the seaborn jointplot documentation (https://seaborn.pydata.org/generated/seaborn.jointplot.html)

Play arround with the code below, changing different elements

In [ ]:
```
sns.jointplot(x='price_clean3',y='rating', kind ='scatter',data=coffee,\
              color='grey', height=8, marker='v');
```

Plotting categorical data

Bar chart

We are going to use seaborne.catplot; this is a visulisation which can handle categorical data.

We are going to plot the column 'coffee_shop;, as each row is a unique type of coffee by counting the number of times the coffee shop name occurs that tells use how many coffees they sell.

In [ ]:
```
sns.catplot(x='coffee_shop', data=coffee, kind='count');
```

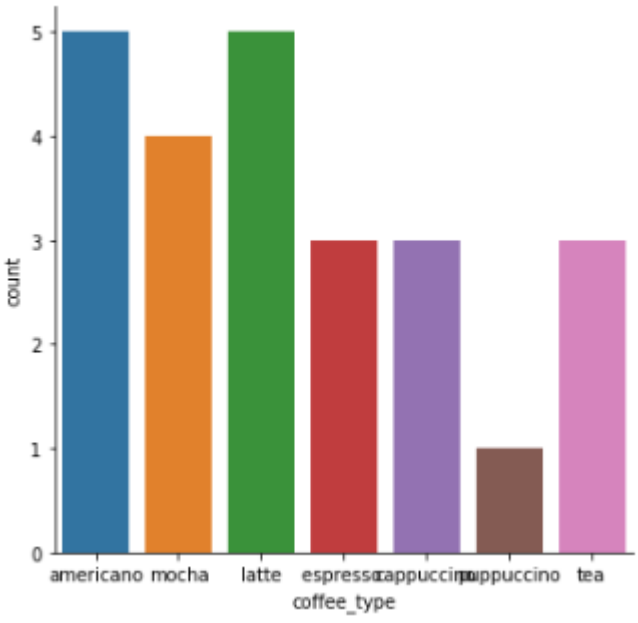
Now produce a count plot for column 'coffee_type'

Write your own code in the cell bellow

In [ ]:
```

```

This is what you should see



Box plot

Visualises the summary statisitcs of the data: mean/median and interquartile ranges

In [ ]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x='coffee_type', y='rating', data=coffee);
```

Violin plot

Also visualises the summary statisitcs of the data, but often prefered over a box plot as they also show the distribution of the data

In [ ]:
```
plt.figure(figsize=(10,5))
sns.violinplot(x='coffee_type', y='rating', data=coffee);
```

You can edit the violin plot to show a third variable, using the `hue` parameter.
Adding `hue='cake_deal'` will produce a violinplot for every coffee type with and without a cake deal

In [ ]:
```
# Visualise the diistirbution and summary stats. of coffee price for each coffe type, w

# define plot size
plt.figure(figsize=(10,5))
# plot
sns.violinplot(x='coffee_type', y='price_clean1', hue='cake_deal', data=coffee);
```

Categorical plot: the realtionship between a categorical and numerical variable

Plot the rating of each coffee by shop

In [ ]:
```
sns.catplot(x='coffee_shop', y='rating', data=coffee, height=6, dodge=True);
```

Plot the price of each coffee by type and colour the points based on the shop they were purchased from

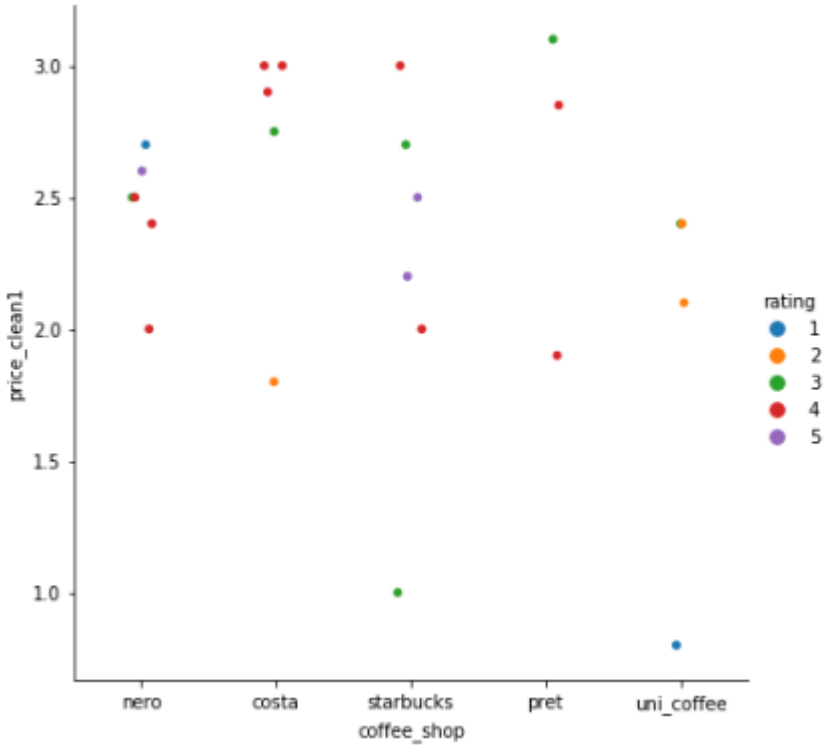
In [ ]:
```
sns.catplot(x='coffee_type', y='price_clean1', hue='coffee_shop', data=coffee, height=6
```

Plot the cleaned price of each coffee by shop and colour the points based on the rating they were given

In [ ]:
```

```

It should look like this



We will cover more complex visualisations next week. But if you are interseted try looking at the seaborn example gallery (https://seaborn.pydata.org/examples/index.html) and try experimenting plotting some of the visualisations with the coffee data.
You can also try reading in your own data (make sure its in a CSV format- this is easy to do for excel spreadsheets (https://www.ablebits.com/office-addins-blog/2014/04/24/convert-excel-csv/)) using `pd.read_csv()` and making your own visualistaions.

File Paths

In [ ]: