**Week 4 summary notes**

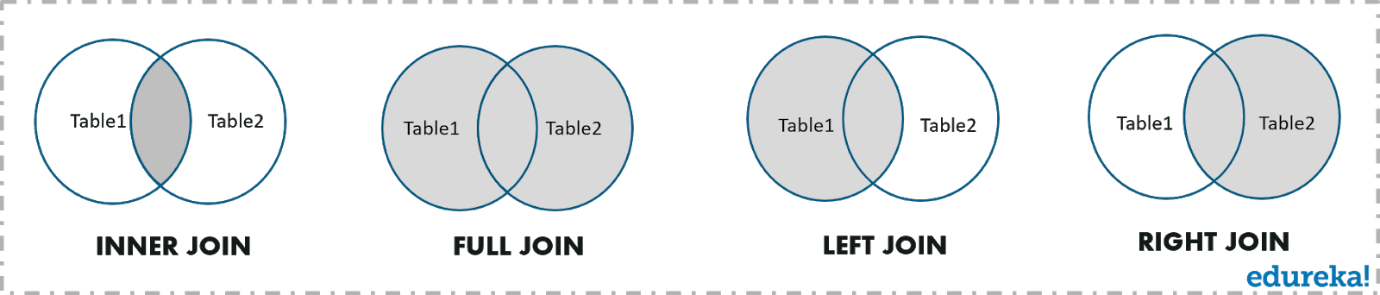
**Pandas: Lesson 3**

**Merging dataframes**

Often data sets are distributed across multiple tables (more on this when we get into SQL). Sometimes we may need data from two or more tables, in this case we need to merge two or more tables into a single unified table. This is part of the data wrangling process. The four main types of joins are:

* Inner join - Select records that have matching values in both tables.
* Outer join (aka Full join) - Selects all records that match either left or right table records.
* Left join - returns all records from the left table and all records that match from the right.
* Right join - returns all records from the right table and all records that match from the left.

The left table is referred to as the first table referred to in the function, the second table will be the right table.

****

We can join two (or more) tables in python using the pd.merge()method from pandas. Pandas will default to use inner join if not specified as a parameter to the function. The most basic use of pd.merge is:

pd.merge(table\_a, table\_b, on=”joining\_column”, how=“join\_type”)

However, there are plenty more optional parameters that can be included in the function. Please look up the Pandas documentation for further detail. Joining tables will become clearer once we encounter SQL later on in the course.

**Reference:** [SQL JOIN, JOIN Syntax, JOIN Differences, 3 tables - with Examples](https://www.dofactory.com/sql/join)

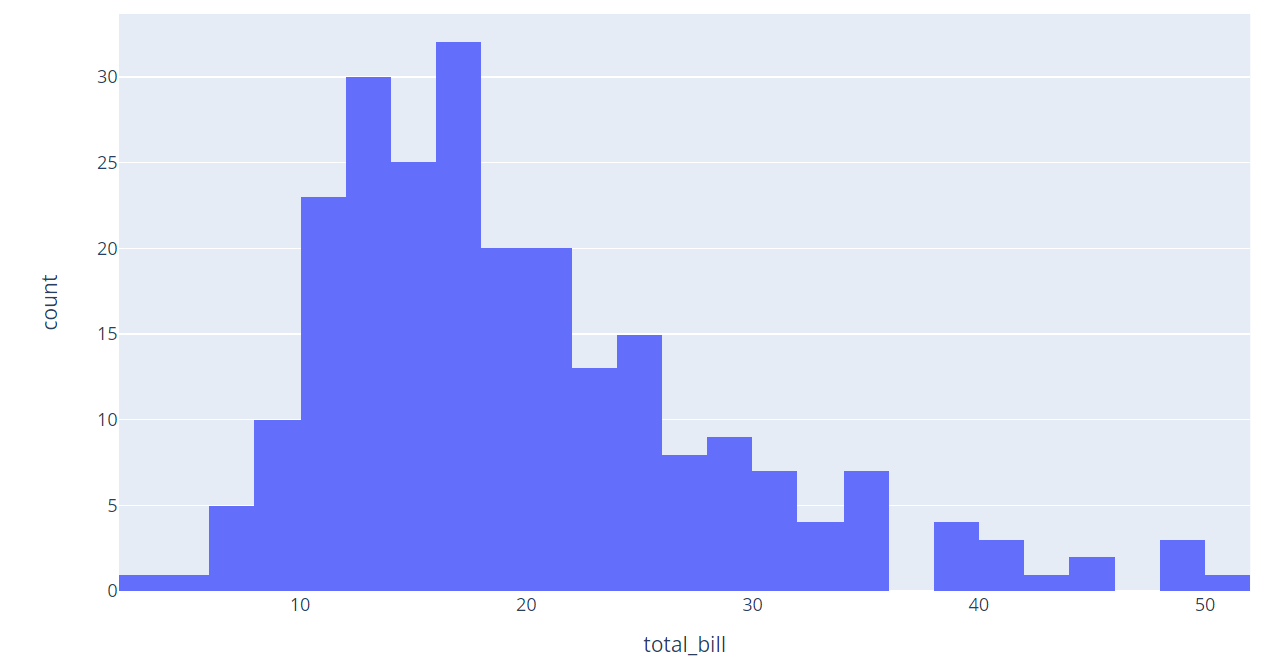
**Reference:** [How to merge DataFrames in pandas (video)](https://www.dataschool.io/merging-pandas-dataframes/)

There are other ways to combine tables (i.e. dataframes) for those interested: [Merge, join, concatenate and compare — pandas 1.1.5 documentation](https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html)

**Binning**

*Why do we need to bin?*

For example, you may have data on customers’ electricity use. Perhaps you were tasked to find out on average how much customers spend on their electricity per year? One option to present and analyse the data would be through a histogram where we bin, i.e. define a range of values and count the values that fall within that range. This could look like this:



Binning group customers values into bins, where a bin is a range of numbers. For instance, that chart above bins values in groups of 5 (i.e. 0-4, 5-9, 10-14 etc.)

**Reference:** [Histograms review (article)](https://www.khanacademy.org/math/statistics-probability/displaying-describing-data/quantitative-data-graphs/a/histograms-review)

**Reference:** [Pandas cut() Function Examples](https://www.journaldev.com/33394/pandas-cut-function-examples)

**Mapping**

Mapping is used to format and style data. By using the .map() and .format()commands, we can style data to present it in a different format (similar to how we set formats to cells in Excel and VBA). Styling data is very important because co-workers and colleagues may refer to your data - it's your job to make it as easy as possible for someone else to understand.

We can apply the map() function to a series in a pandas data frame:

df[“column\_to\_format”].map(“{style}”.format)

The “{style}” is used to determine how values will be adjusted. Please see [Style Pandas Dataframe Like a Master | by Eyal Trabelsi](https://towardsdatascience.com/style-pandas-dataframe-like-a-master-6b02bf6468b0) for more details (with additional content on how to physically format dataframes - not necessary for the course but noteworthy).

Be wary when formatting data - it will convert numeric data to strings meaning that you can no longer perform operations on the data since the data type has changed from numeric to string. Best use of formatting data is just before you want to output the data.

**Reference:** <https://pbpython.com/styling-pandas.html>

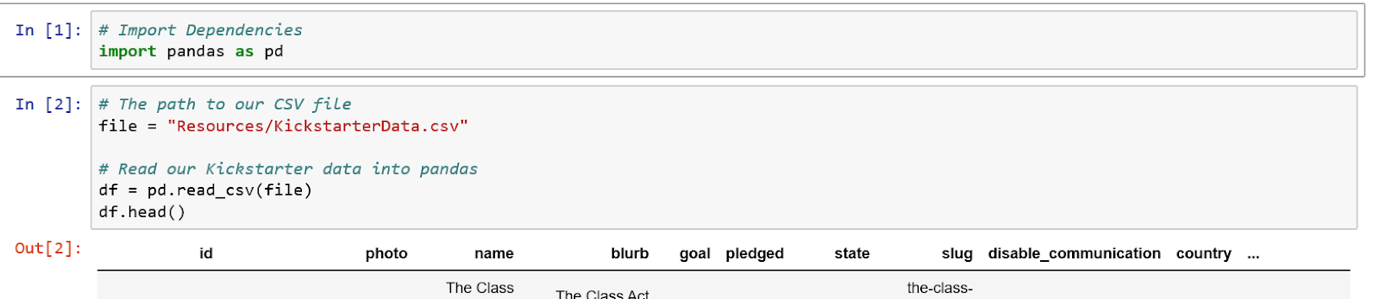
**DEBUGGING YOUR CODE**

**Tips and tricks**

1. Build the code block by block, step by step. For example, instead of writing everything down in one cell as below:



Instead, break it down into separate cells (i.e. one line per cell). This will help you to find which line of code is causing the error.



1. Print out the variables everytime you change it or manipulate it. This way you can keep track every step if the code is working according to the way you need.
2. If you do encounter an error. **Read the error message!**

When reading the error:

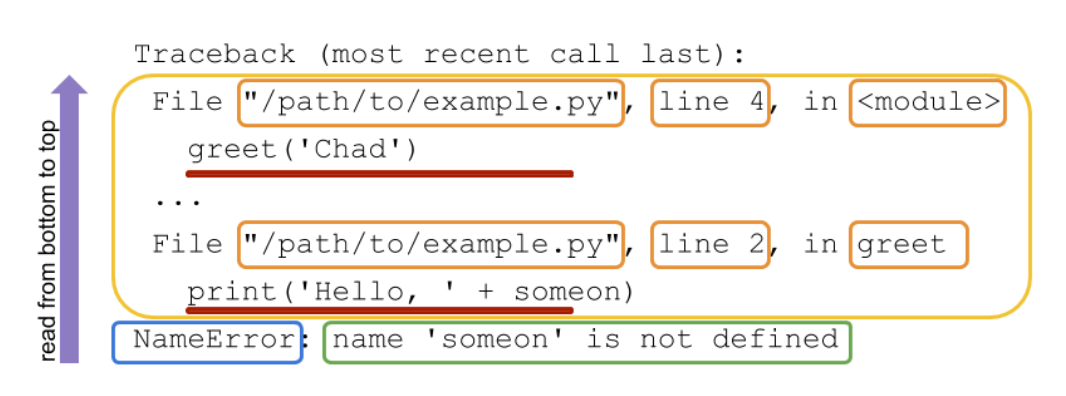
• The message will tell you which function the error was in, and which function(s)

called this. That helps you trace the path to the error.

• It will tell you what type of error it was. You can cut-and-paste key parts of the

error text into Google to help decipher it.

BUT HOW DO I READ THE ERROR MESSAGE?



Refer to this link below for a step by step guide:

[Understanding the Python Traceback – Real Python](https://realpython.com/python-traceback/)

These are the most common errors:

* Spelling (the spelling needs to be correct, as long as it is consistent).
* Capitalisation (Python is case-sensitive – you have been warned).
* Check that you are using “==” for comparison and “=” for assignment.
* Check you haven’t omitted operators, e.g., you should replace 5x with 5 \* x.

**Reference:** [Beginner Guide to Exception Handling in Python](https://towardsdatascience.com/exception-handling-in-python-85f49801b131)

[8. Errors and Exceptions — Python 3.9.1 documentation](https://docs.python.org/3/tutorial/errors.html)