**Working with and plotting data in python**

Now you’ve got python loaded and can hopefully run code, we can start looking at some data. The tutorials from Kaggle are a great (and much better) introduction to python: <https://www.kaggle.com/learn> . Here I will demonstrate how to read in and explore data in the pharmaceutical space.

The most common way to deal with data in python is with a library called “pandas”. A library is a toolkit that you can load into python to do specific tasks. Libraries exist for almost anything, and sometimes need to be installed before you can use them. Luckily anaconda has pre-installed almost all libraries we could need for doing machine learning and working with data. For example – most machine learning code is found in a library called “sklean”.

To begin, I have prepared an example of a dataset of mucoadhesive polymers, found in the excel file “example\_data\_mucoadhesive\_polymers.xlsx”.

The code to perform all the work in this document is in the notebook file “Working\_with\_and\_plotting\_data.ipynb”.

**Reading in data with pandas**

To begin, we need to load the pandas library so that python knows to “prepare” the code. Essentially python only loads a core set of functions when opened, and you have to tell it to load extras, in order to preserve memory. Libraries are easy to load:

A screenshot of a computer

Description automatically generated

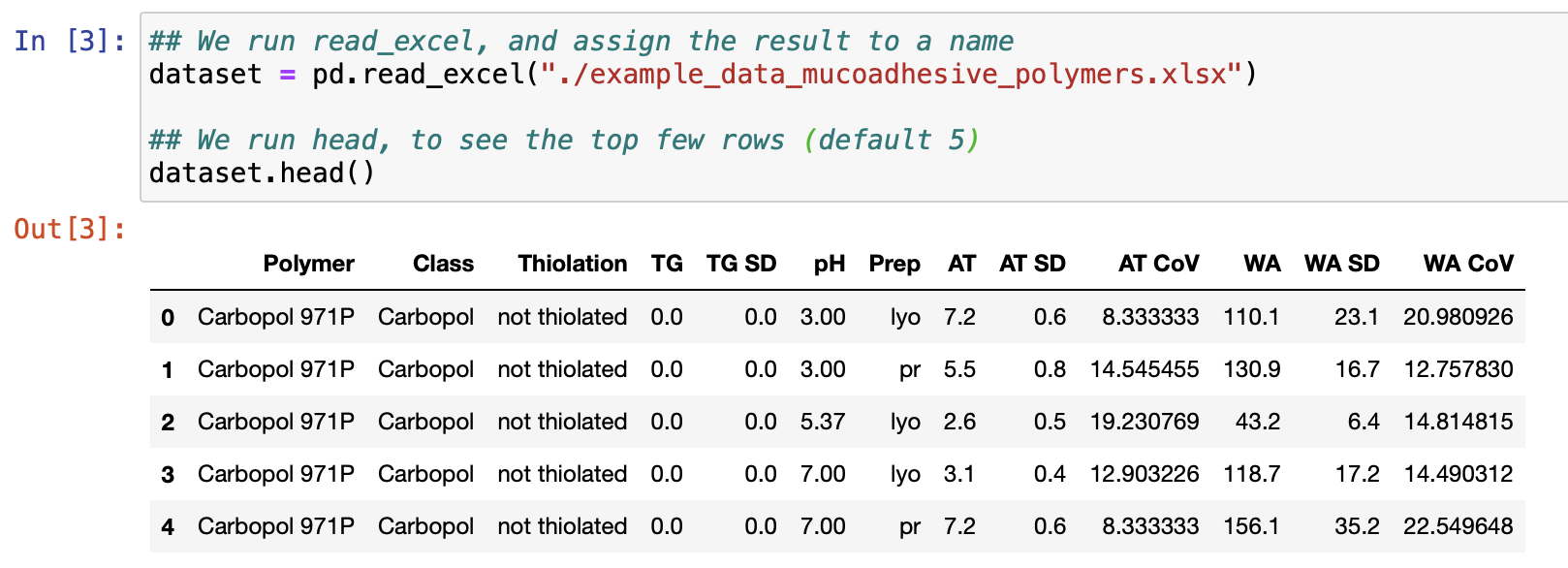
The “import” command imports pandas, but generally we might want to “call” the library in the future. To enable this, we assign a name to pandas. The generally accepted shorthand for pandas is “pd”:

A white rectangular sign with green text

Description automatically generated

Now, every time we type or reference “pd”, python knows we are talking about pandas.

Next we need to read in our data. Because our data is currently in an excel file, there is a function (see the Kaggle tutorials for more specifics about what a function is) to do this. By running the “read\_excel” function, pandas can extract the data from an excel spreadsheet, and by then running “head”, pandas will show us the top few rows of the spreadsheet:



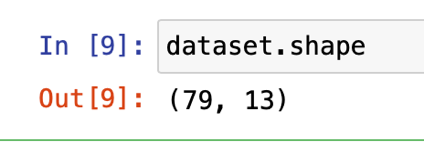
Pandas has read in our file and turned it into what is called a “DataFrame”. DataFrames are a pandas object that stores the data. DataFrames are easily manipulable in pandas/python, and are the object on which most machine learning methods run.

We can look at the column and row names in the DataFrame:

A screenshot of a computer

Description automatically generated

Get properties like the shape:



And a description of the general metrics for columns:

A screenshot of a data sheet

Description automatically generated

We can look at columns by just passing the column name to pandas:A screenshot of a computer

Description automatically generated

Similarly, for rows we use iloc:

A screenshot of a computer

Description automatically generated

Note that python starts from 0 rather than 1 in almost all cases.

You can do almost any manipulation you want in pandas – it is extremely each to manipulate, merge, transpose, calculate new columns etc. The documentation is great (<https://pandas.pydata.org/docs/user_guide/index.html>) and googling (or maybe chat-GTP) will almost always bring up some single line code someone has produced to do what you want.

Having very briefly introduced pandas, we are going to go over how to produce some very simple plots, and I will demonstrate some simple tips to making plots look nicer in python.

**Introduction to plotting**

There are many ways to plot graphs in python – the most common and easy to use (debatable) is to use a library called **matplotlib**. Matplotlib is a library originally based on plotting in a similar way to matlab (ew).

The general way to plot in python/matplotlib is:

-Load the data into a pandas dataframe

-Format the data in the way the plotting function expects

-Pass the data to the function to plot a graph

-(Spend a lot of time messing about trying to make the graph look nice)

The simplest way to plot data is just to call the “plot” function and pass it data:

A screen shot of a graph

Description automatically generated

Here we have essentially given the “plot” function of matplotlib a column, and asked it to plot it. The default plot will just be a line. As we give the function only a single column it is assuming the x values are the index (i.e the row number), and the y value is the value of column “TG” in that row.

Obviously this isn’t super informative but it gives you an idea of how easy it is to produce plots. Below I will show a couple of examples of types of plots applied to our data, but for more please see the python graph library, which has reproducible examples of almost anything you could want to plot:

<https://python-graph-gallery.com>

**Pie charts**

Possibly one of the simplest ways to view certain data types. In our dataset there are mucoadhesive polymers formed from many different types of base polymer. Here we can visualise the proportions of the different bases.

A screenshot of a computer

Description automatically generated

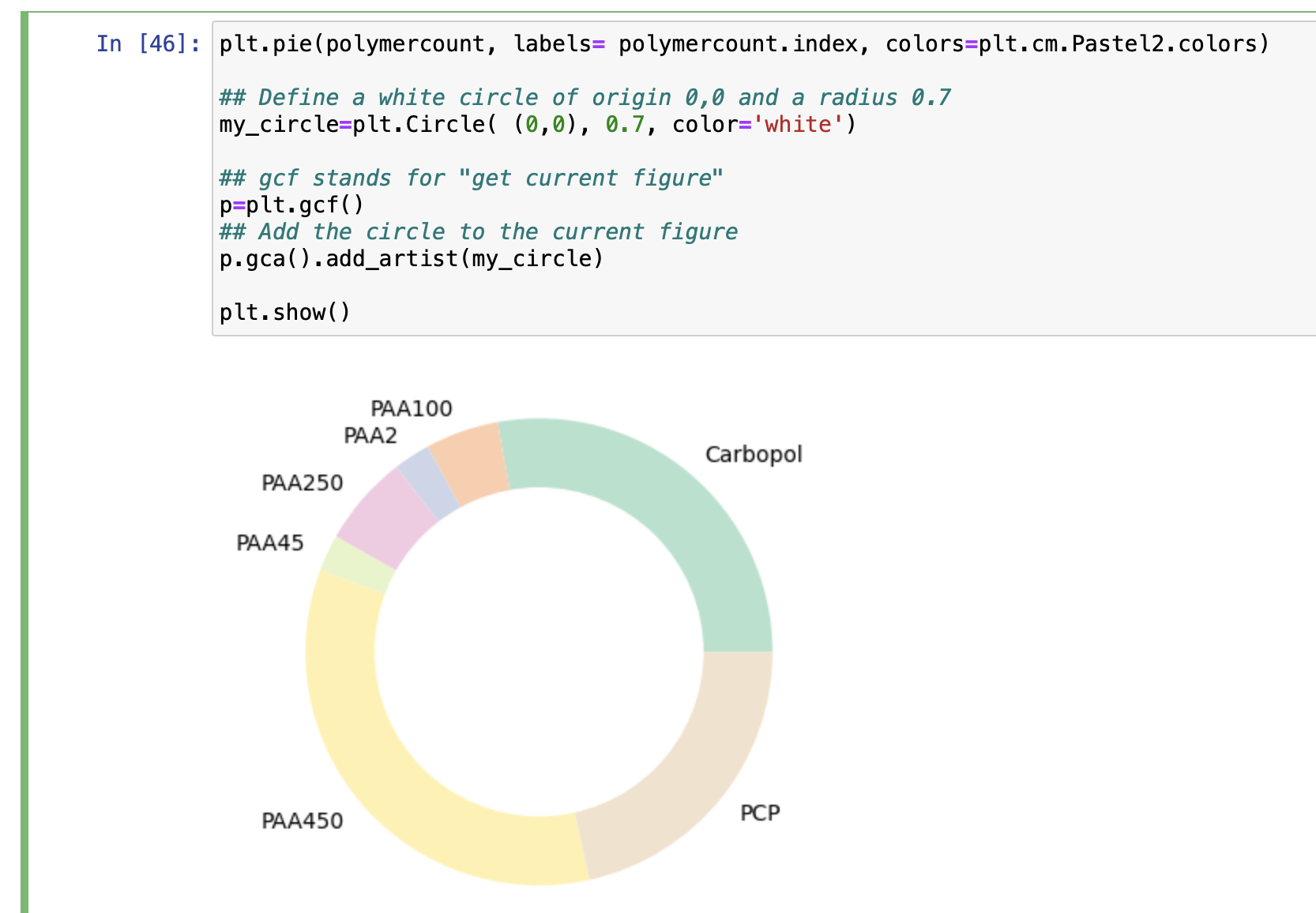
You can change the colours by either passing the plotting function a list of colours, or by specifying one of matplotlib colour palettes:

A screenshot of a graph

Description automatically generated

Finally – many people hate pie charts: <https://www.data-to-viz.com/caveat/pie.html>

They have some problems. Generally you should use a bar chart instead, but you can also make a pie chart more readable with a simple addition:



**Bar charts**

Bar charts are generally fairly simple in matplotlib. As an example, imagine we want to know the mean number of Thiol groups for each base polymer. We first need to group the data into base polymers, and then average the Thiol group columns:

A screenshot of a computer

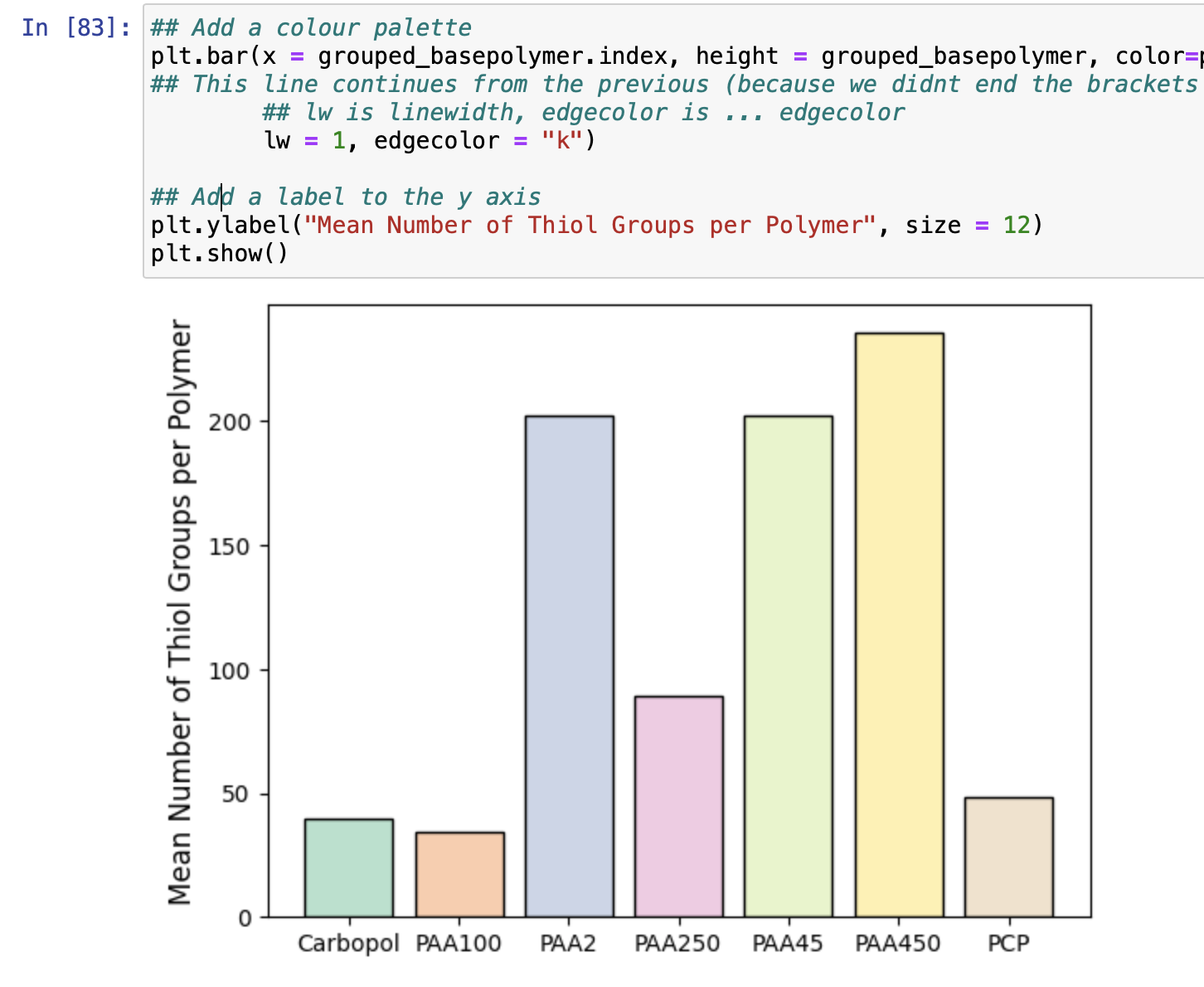
Description automatically generated

We can then pass this on to the bar plotting function:

A graph with blue bars

Description automatically generated

We can then make it look a bit nicer:



There are pretty much unlimited ways you can now change/adjust this plot, but this gives you a starting point.

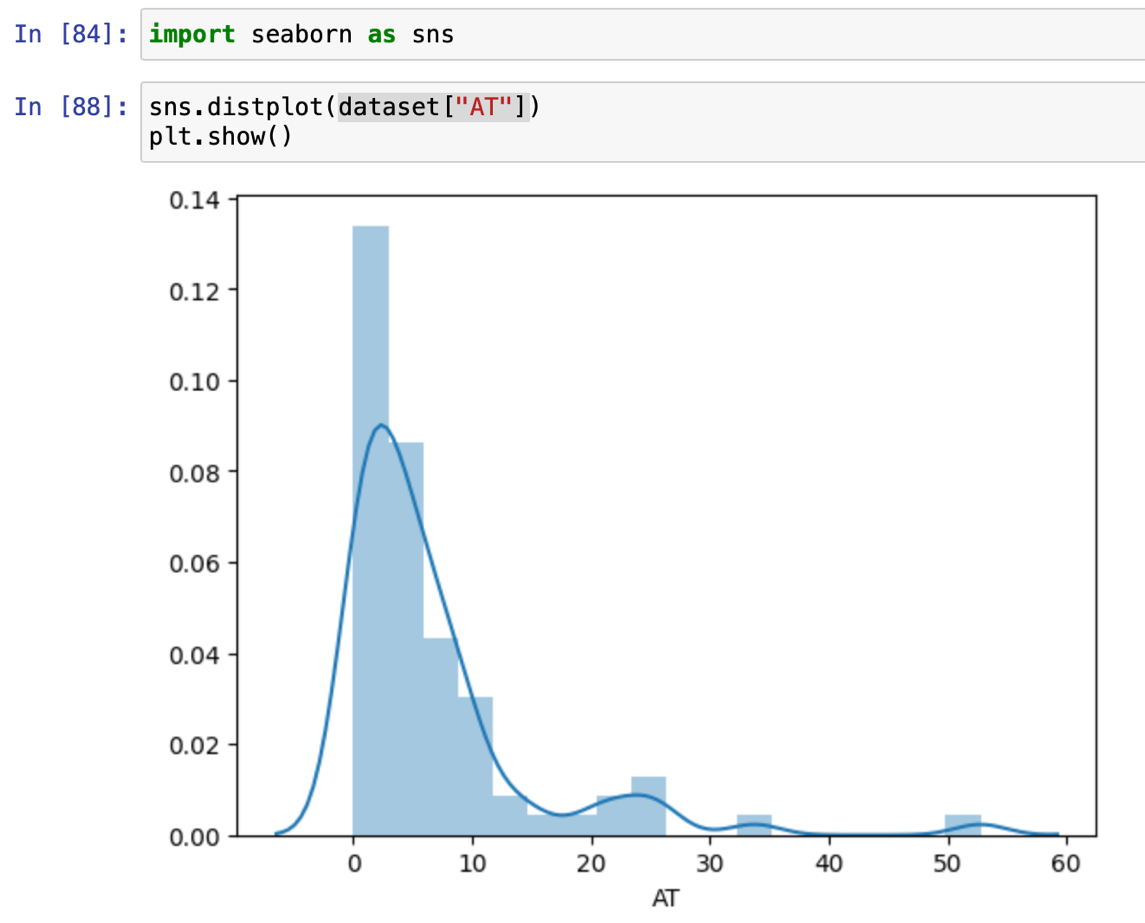
**Distribution plots**

Sometimes it easier and more useful to view the distribution of some data as a histogram. Imagine we want to see the distribution of Adhesion Times (AT) for our polymers:

A graph with blue squares

Description automatically generated

But a nicer way to look at distributions is using a “kernel density estimate” (KDE), which plots the distribution as a line. Yet another plotting library called **seaborn** can do this very easily:



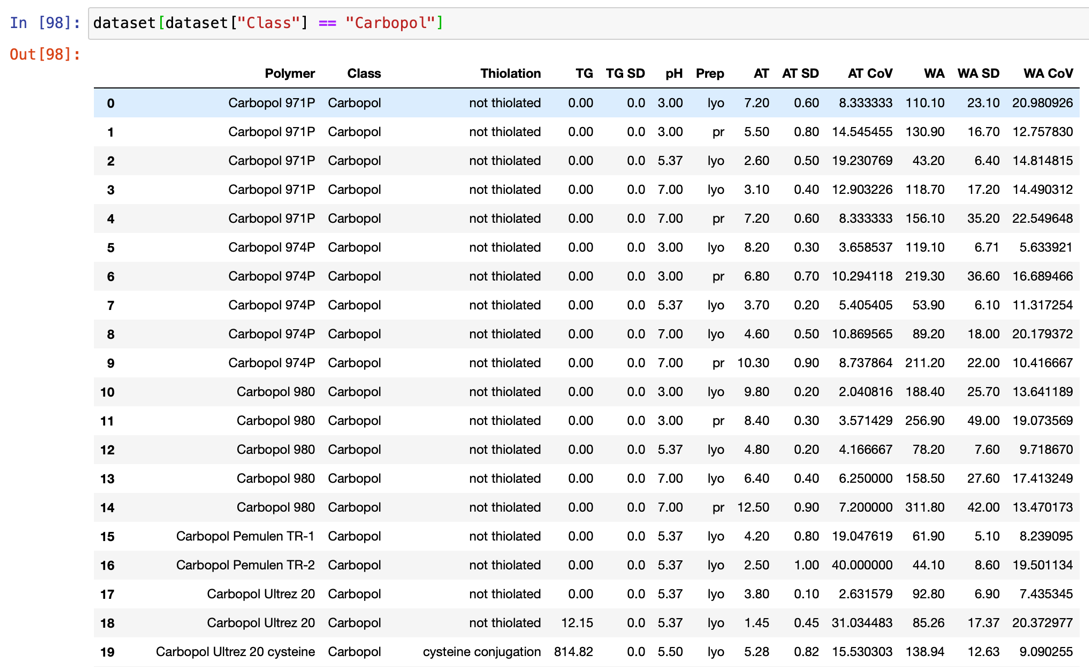
Say we wanted to plot the distribution of adhesion time (AT) for two different base polymers. This is possible through subsetting our dataframe.

To subset our dataframe we can first select only rows where the “Class” is eg. Carbopol:

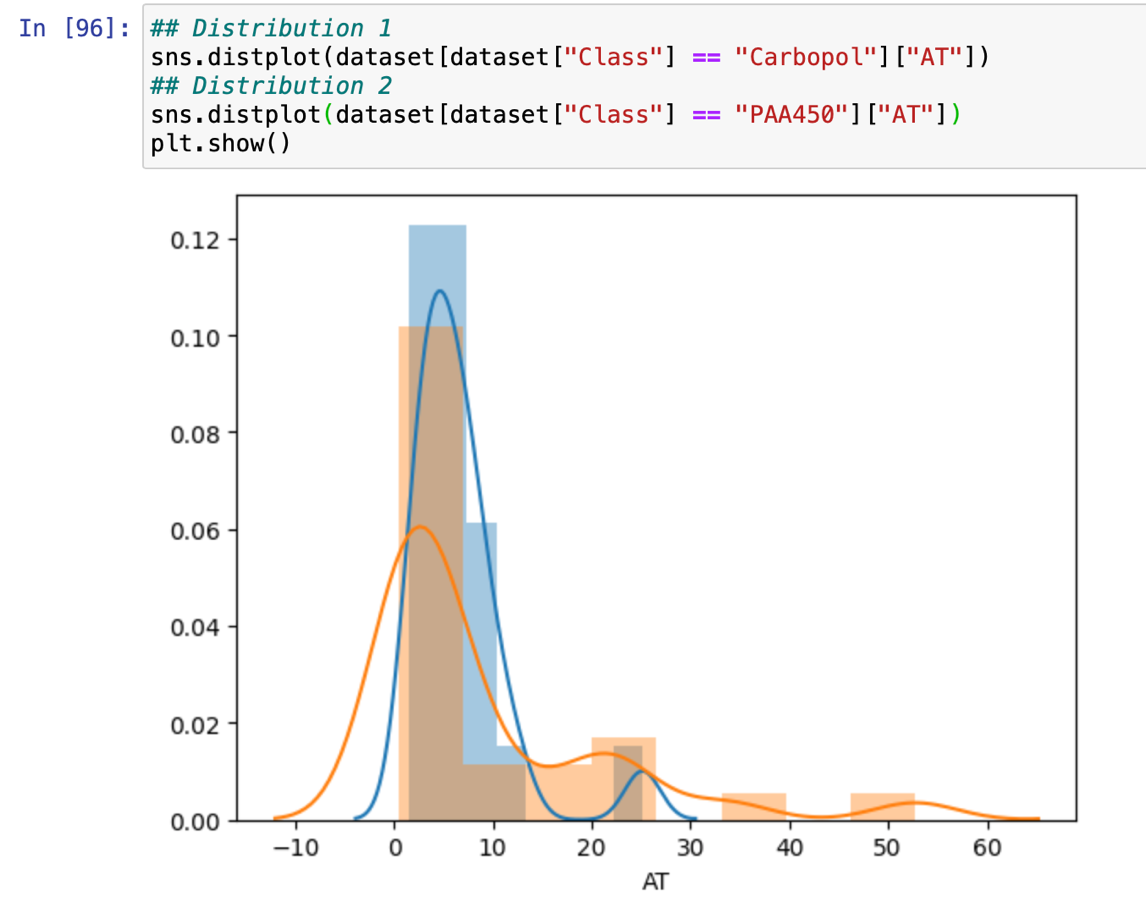
A screenshot of a computer

Description automatically generated

Notice that this just returns a list of True/False – ie. if the row contains Carbopol in the “Class” column. To actually subset the dataframe we need to tell pandas to select rows where the above query is True:

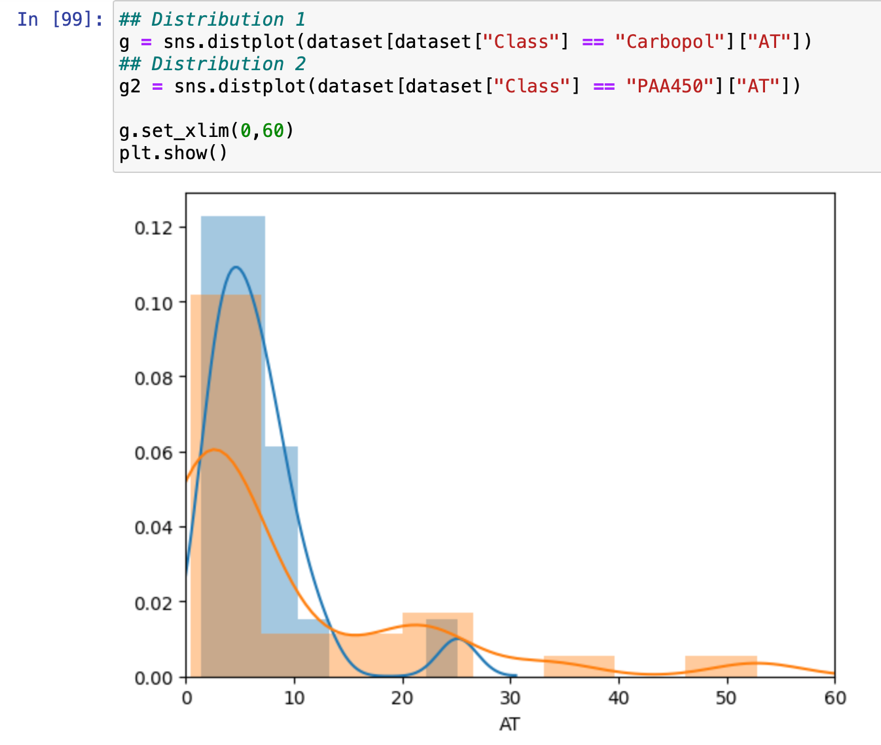


This has returned a new dataframe for only the rows containing “Carbopol” in the column “Class”. We can now use this with our plotting function to plot two distribution plots over each other:



See how we can just overlay multiple plots and matplotlib/seaborn will just add them on top of each other. (Notice also how distributions with a strict “cutoff” (like this one cannot go below 0) are not always suitable for a KDE – as a significant part of the distribution is impossible. We can adjust this however by:

1. Giving each graph a name (g and g2)
2. Altering the limits of the axes



**Scatter plots**

Scatter plots are great for displaying the relationships between two values. In our case lets look at the relationship between the Adhesion Time (AT) and Number of Thiol Groups (TG) colums:

A screen shot of a graph

Description automatically generated

**Boxplots**

Finally a more complex example – lets say we want to plot the distribution of thiol groups separated into base polymer. The best way to do this is with a boxplot.

To generate a boxplot we first need to format our data in a way that the plotting function can read. Seaborn needs the data in a “melted” format:

A screenshot of a computer

Description automatically generated

Which we can then pass to the boxplot function: A screenshot of a computer screen

Description automatically generated

We can now try and clean this up a little:



Hopefully this is enough to get you started with some plotting. For more examples see the python graph gallery: <https://python-graph-gallery.com> or google for things.