Math is important to Data Science because it allows us to speak about data sets in an objective manner. As humans there is an innate level of error to us. We can see a data set, and we can start to see and understand patterns in it. We assume that our findings are correct. Now we present our findings, and it turns out that we were wrong. Using numbers and facts can ensure that we can speak with authority, and that we are not analyzing data based on emotion, and feelings. We need quantitative data and results backed by that quantitative data. When looking at a data set, there are a few key values that we should always seek to evaluate. This assumes we are working with a clean data set.

<https://www.kaggle.com/andrewsundberg/college-basketball-dataset>

Dataset that we will be working with.

We are going to be creating a series of visualizations and a series of equations to help us see some patterns with data.

Let’s download and extract the csv files from the website.

Let’s plan out our file structure.

Create a new Folder that contains 2 subfolders. Give them appropriate names. If you can’t think of any, use the ones I provided. The file structure should look like this.

Lesson 2

->dataset

->cbb.csv

->cbb13.csv

->lab

->main\_program.py

Next, we are going to open up our main\_program.py.

Let’s populate it with some code.

Remember that python’s “path” is relative to where python is being executed in the terminal. What does that mean exactly? Pay strict attention to where python is being called in your terminal. What does that mean exactly? It means that where your TERMINAL IS determines how your path should be built. I opened up my terminal in the Lesson 2 folder. That means that when I execute python, the directory has to point with that fact in mind. I have to go from the Lesson 2 folder, into the Dataset Folder, and then I have to Select the cbb20.csv file. If I open my terminal from the lab folder for example, first I have to path OUT of the lab folder, then I have to path into the dataset folder, and then I have to select my cbb file. Try to always keep this in mind. In larger scale projects and labs it is better practice to use absolute pathing, using path.os, but for smaller projects like this one, we can just use relative paths. We just have to keep in mind where python is being called from in the terminal.

Let’s execute our code and see what our data looks like, from a high level. Let’s also ensure to look back at our data set on Kaggle.com. Whenever we use data, it’s relevant to get as much information on the data as possible. What each column represents, that RANGES of data, whether the data is categorical or quantitative, who the individuals in our data set are. For this specific data set, there are values that have already been calculated and input. This means we have to know what the values are and how they were calculated.

For example we have BARTHAG. What is BARTHAG? How is that calculated?

We also have ORB, ADJOE, and tons of other headers that don’t necessarily describe to us what they represent. It’s very important to make sure that we get as much information as possible as data scientists.

Let’s start to get some useful information from our Data. How many conferences are there in Division 1?

Let’s start by printing out data frame and seeing what information we can get from it, at a base level.

Next, let’s do a value counts on the Conference row, and store that SERIES into a variable.

Columns in a DataFrames are SERIES, not Dataframes.

Series are 1-Dimensional objects, with an index.

In the case of a value\_counts on a column, the index of the series are the unique values in a row, and the values in the column are the counts.

We have a series, but we want to plot the conference against the count, so it’d be useful to have a DataFrame instead.

To do this, we can call the reset\_index() method. What this does, is “shift” the index column so to speak. It takes the existing index, places it as the left most column, and instead creates a new index, by default, using numerical values.

<https://pandas.pydata.org/docs/reference/api/pandas.Series.reset_index.html>

The reset\_index() method is quite powerful, so here is a link to read more about it. Note that this method works only on series though, not dataframes.

Now that we have our New Dataframe, let’s plot that dataframe as a bar graph.

Perfect, now we have our visuals. We can see the high and the low.

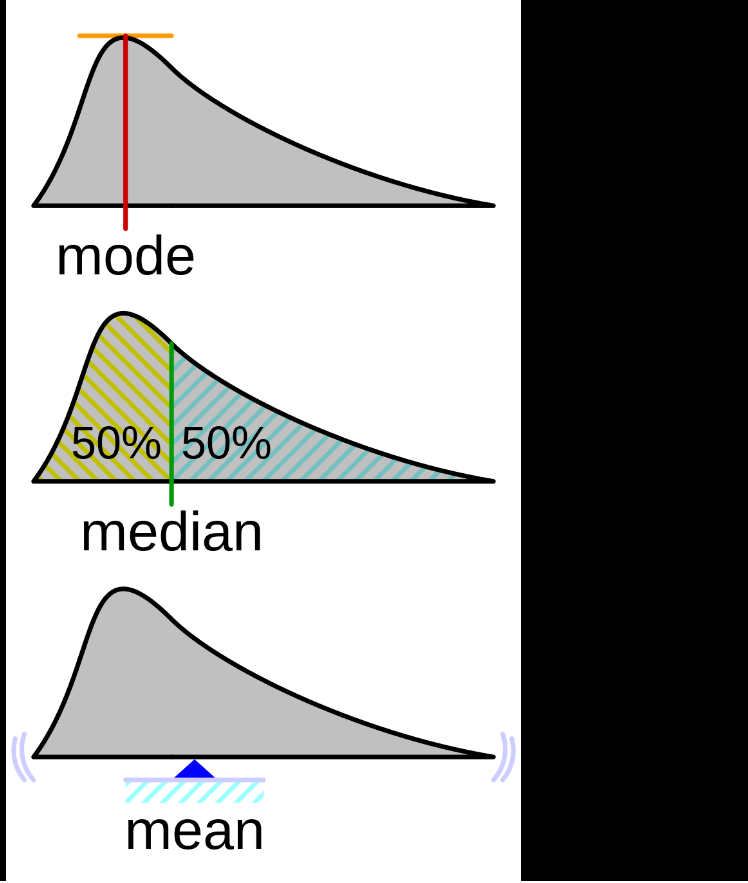
We can start to make some guesses, but we don’t have anything tangible.

We want to speak with authority, and we want to speak with facts.

Let’s get some valuable information from our DataFrame.

Let’s go over the Mode.

The mode is the value that appears most often in a data set. It is possible for a data set to not contain a mode. This is fine. It is also possible for a data set to contain more than one mode.



Visually, this is what the mode looks like. It is the peak of the probability density function.

I could go over how to iterate through the data frame, and how we could do a count of each, but that is considered bad design. We should avoid hard iteration of the dataframe. What do I mean by “hard” iteration. I mean to say that we should avoid at all cost iterating through each row and or column by implementing the iterrows and the itertuple methods. Pandas has many many MANY built in methods, and start your search there first.

<https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas>

Mode, is fairly easy. It’s a count. We already know how to count. Just run value\_counts on the specific row, and then bam. The first results(or results) should give you your mode.

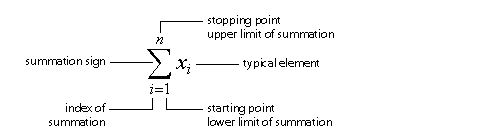
Next, let’s try to get the median. Using our visual, we can see what the median represents. The median is the midpoint. The value at which 50% of the distribution lies on the left, and 50% of the distribution lies on the right. To

To get the median, we order the data set in ascending order. If the data set contains an odd amount of values it should be the middle value. If the dataset contains an even amount of values, it isn’t too much more difficult to get the median. It’s the middle two most values, added up, and then divided by 2. Sometimes, the median does not appear in the dataset itself. That is fine!

Lastly, we have the mean. Let’s stick away from using the “average” value. The reason being is that the last 2 values are averages as well. Let’s use the mean. It is the total of all the values of the dataset, divided by the amount of values in a dataset.



You might see this fancy notation. This might look scary, but let us clear it up



This is saying, the mean is equal to 1/n \* the summation of the ai … an.

If you are interested in learning mathematic notation, here is a good starting point.

<https://en.wikipedia.org/wiki/Glossary_of_mathematical_symbols>

While not necessary, it is helpful for future reference. Now we have our three key values.

Let’s get these values for the subset of our dataframe.

conf\_df.describe() let’s us easily see our mean. This is useful. If we want to get the value to use for later we can do, df[‘column’].mean().

This is useful if we need to use this for mathematical operations later.

The same can be done for .mode() and .median().

Now we have our 3 measures of central tendency. Very helpful data that we can use to calculate further values.