5.3 Lesson Summary - Introduction to Statistics

Performing statistical analysis often reveals meaningful information about your data. Summary statistics like mean, median, mode, variance, and standard deviation describe patterns in your data. We have previously explored these statistical concepts and now we will learn how they are calculated in Python. To generate these statistics, we must use Python libraries we are already familiar with like Pandas and NumPy and we must utilize new libraries like SciPy. These statistical values can be used to enhance data visualization.

Concept: NumPy's mean and median methods allow you to get the average and median of a group of numbers in Python. For example:

*import numpy as np*

*np.mean(my\_numbers)*

*np.median(my\_numbers)*

* Activity: 01-Ins\_Summary\_Statistics

Concept: The Python library **SciPy** offers a number of useful functions to perform data analytics. The stats package in SciPy provides a number of methods to perform statistical analysis. The **mode** method can be used to find the most common number in a series. For example:

*import scipy.stats as sts*

*mode\_scipy = sts.mode(my\_numbers)*

* Activity: 01-Ins\_Summary\_Statistics
* Suppl link: <https://www.scipy.org/>

Concept: It is useful to be able to describe how spread out or compact your data is. **Standard Deviation** and **Variance** describe the spread of your data. Variance is the average of all of the squared differences from the mean. Standard deviation is the square root of the variance. In practice standard deviation describes where you will likely find most of your data and what qualifies as an outlier. NumPy provides the *var* and *std* methods to calculate the variance and standard deviation. For example:

*data\_variation = np.var(my\_numbers, ddof = 0)*

*data\_standard\_dev = np.std(my\_numbers, ddof = 0)*

* Activity: 01-Ins\_Summary\_Statistics

Concept: Describing how your data is distributed can help you better understand it. If your data is **normally distributed** this means it is spread out symmetrically. A graph of a normal distribution is referred to as a bell curve. Normal distributions often indicate results produced by random chance. Being able to conclude that your data is either the result of random chance or a pattern is a very important distinction. Scipy's stats package provides a *normaltest* method that we can use to calculate if our data is normally distributed. The *normaltest* method returns a chi-squared statistic, and the associated p-value. A p-value greater than 0.05 indicates that your data is normally distributed. In Python your code would look like the following:

*chi\_squared\_and\_p\_val = sts.normaltest(my\_numbers.sample(50))*

* Activity: 01-Ins\_Summary\_Statistics

Concept: A **z-score** is the number of standard deviations a given value is away from the mean of a data set. Because standard deviation describes where most of the data is, a data values' z-score indicates if the value is typical or an outlier. The higher the z-score the more of an outlier the data is. To calculate the z-score's for a dataset you can use SciPy's *zscore* function. For example:

*my\_zscores = sts.zscore(my\_numbers)*

* Activity: 01-Ins\_Summary\_Statistics

Concept: **Quantiles** allow us to divide data into specific regions. **Quartiles** divide our data up into four equal sized groups. Pandas DataFrame offers a *quantile* method to group data. For example:

*quartiles = my\_numbers\_df.quantile([.25, .5, .75])*

* Activity: 02-Ins\_Quartiles\_and\_Outliers, 03-Stu\_Summary\_Stats\_Python

Concept: **Standard Error**, like standard deviation, describes how spread out the data is. Where standard deviation deals with the entire population Standard Error only deals with a sampling of that population. SciPy provides a *sem* method for calculating the Standard Error of a sample of data. For example:

*from scipy.stats import sem*

*standard\_error = sem(sample\_data)*

* Activity: 04-Ins\_Standard\_Error, 05-Par\_Standard\_Error
* Suppl link: <https://www.statisticshowto.com/what-is-the-standard-error-of-a-sample/>

Concept: The Pearson Correlation Coefficient is a statistic that indicates the degree of correlation between two variables. It will be a value between +1 and -1 with +1 meaning a positive linear correlation. SciPy's stats library provides a *pearsonr* method to derive this coefficient, for example:

*correlation = sts.pearsonr(my\_numbers1, my\_numbers2)*

* Activity: 06-Ins\_Correlation\_Conundrum, 07-Stu\_Correlation\_Conquerors

Concept: A **linear regression** models the behavior of the data by generating a line to predict the relationship between two variables. SciPy's *linregress* method provides values that can be used to generate the equation for the line of our linear regression. For example:

*from scipy.stats import linregress*

*(slope, intercept, rvalue, pvalue, stderr) = linregress(x\_values, y\_values)*

* Activity: 08-Ins\_Fits\_and\_Regression, 09-Stu\_Fits\_and\_Regression