4.3 Lesson Summary - Merging and Data Clean Project

The data you're analyzing will often be spread across different sources, be formatted suboptimally, or require additional organization to be useful. For these reasons it is very important to be able to combine your data together, organizing it in a logical manner while formatting it to make it more understandable.

Concept: It is not uncommon for the data you're analyzing to come from different sources or to be broken into multiple parts. Combining data into a meaningful whole is an important part of data analytics. Pandas provides the ability to **merge** DataFrames using the *pd.merge()* method. Combining two DataFrames into a new DataFrame can look like the following:

*merged\_df = pd.merge(first\_to\_merge\_df, second\_to\_merge\_df, on="merge\_on\_me")*

* Activity: 01-Ins\_Merging, 02-Stu\_Cryptocurrency

Concept: Pandas merge method performs an **inner join** by default. An inner join only returns data that has matching values. Rows in the DataFrames you are joining that do not match your specified *on* condition will be ignored. The total number of rows from a DataFrame created by an inner join is equal to the number of matching rows in the DataFrames that were joined. An **outer join** combines the rows of DataFrames regardless of if they match or not. The total number of rows in a DataFrame created using an outer join will be the number of matching rows in the joined DataFrames plus the unmatched rows in each joined DataFrame. A **left join** will keep all of the rows in the left DataFrame and only the matching rows in the right DataFrame while a **right join** will keep all of the rows in the right DataFrame and only the matching rows in the left DataFrame. An inner join is the default type of merge, to perform an outer, left, or right join add a *how* parameter set to *"outer"*, *"left"*, or *"right"*, for example:

*merged\_df = pd.merge(first\_to\_merge\_df, second\_to\_merge\_df, on="merge\_on\_me", how="outer")*

* Activity: 01-Ins\_Merging

Concept: Organizing data in **groups** can make analyzing it simpler and more intuitive. Pandas provides a "**binning**" function to apply groups to a DataFrame. If you specified the range of your data in a *bins* list and descriptions for your bins in a *labels* list you could categorize your data into groups using the following code:

*bins = [0, 10, 20, 30]*

*group\_names = ["Group1", "Group2", "Group3", "Group4"]*

*df["New Data Column"] = pd.cut(df["Value To Bin"], bins, labels=group\_names, include\_lowest=True)*

* Activity: 03-Ins\_Binning, 04-Stu\_TedTalks

Concept: Raw data is not always formatted in the most legible style. Pandas provides a *map* method to format columns of your DataFrame to make them easier to read. If you wanted to format numbers to look like prices you could use the following code:

*df["data\_to\_format"] = df["data\_to\_format"].map("${:.2f}".format)*

* Activity: 05-Ins\_Mapping

Concept: Cleaning up your data is an important first step to analyzing it. It is imperative to be able to Google, research, and learn how to better use Pandas, Python, and other data analytics tools. Being comfortable cleaning your data and mastering new data analytic techniques requires significant practice.

* Activity: 06-Stu\_CleaningKickstarter, 07-Ins\_IntroToBugfixing, 08-Stu\_BugfixingBonanza