Data 115: Week 8 Data Cleaning in Pandas

Daryl DeFord

October 26, 2020

This document provides descriptions of the pandas functions for data cleaning and aggregation operations (like pivot tables), replicating our earlier work in Excel. Throughout this guide, I will assume that you have already imported the pandas package as pd and that you have loaded in a dataframe called df to process.

- **Missing Data:** When you load in a dataframe with missing entries, they are displayed (like when you use df.head()) as 'NaN' values, which stands for Not a Number. Their specific type comes from the numpy package: np.nan and they are used to represent locations in the dataframe where a row and column index suggest that data could exist but there isn't anything there.
  - You can find NaN entries using the .isnull() function - df['D_Votes'].isnull() returns a series of Booleans telling you whether the corresponding entry is missing or not. Since the output is Boolean, you can use it to access those rows directly: df[df['D_Votes'].isnull()] will return a new dataframe corresponding to the rows where the D_Votes value is missing.
  - To replace the missing entries with another value, you can use the .fillna() function, which takes an input of the value you would like to replace the missing entries with. Thus, df.fillna(-99) will replace the missing entries with -99 throughout the data frame.
  - The previous function is a special case of the .replace() function, which takes as input the value you want to replace and the value you want to replace it with. For example, df.replace(1,100) will replace every occurrence of 1 in the dataframe with 100. This function also takes lists as inputs for both entries, if you want to replace multiple values at once. For example, df.replace([1,0],[True,False]) will replace all entries of 1 with True and all entries of 0 with False.
- **Duplicates:** Detecting and removing duplicate data is a common task for data cleaning. In python the main approach makes a new Boolean column that records whether each row is a duplicate or not, which then allows you to investigate further.
  - To identify duplicated rows in a dataframe, we can use the .duplicated() function, which returns a Boolean column with True values for rows that are duplicates of other rows with lower indices in the series or dataframe. This operation can also be applied to individual columns if we want to find something like multiple entries corresponding to the same id #.
  - To see all rows that are duplicates (including the first occurrence of each duplicate), we can add the optional argument keep=False to the duplicated function.
  - To remove these rows from the dataframe, there is the special .drop_duplicates() function, which saves us from having to drop the rows individually (although you should always check or justify that the rows actually do need to be dropped).
  - Just like in excel, there is a .unique() function that returns all of the unique entries of a given column. To compute the number of unique elements we can use the len function, so something like: len(df['Name'].unique()) tells us how many unique names there are in our dataset.
- **String Processing:** In Python, strings are denoted with single or double quotes around the characters and can include letters, numbers, punctuation, and spaces. The functions below can either be applied to strings directly with something like "hello, world".upper() or to a variable representing a sting:
  my_string = "hello, world"
  my_string.upper()
  - **Basic Functions**
    - .upper() capitalizes all letters
    - .lower() puts all letters in lowercase
    - .title() capitalizes the first letter of each 'word'

    - .proper() capitalizes the first letter
    - .strip() trims the extra white spaces around a string
    - .startswith() returns True or False depending on whether or not the input string is at the beginning
    - .endswith() returns True or False depending on whether or not the input string is at the end
    - in checks whether a substring occurs in a larger string
    - .index() locates a substring and returns its position
    - .find() locates a substring and returns its position
    - .count() returns the number of occurences of a substring
    - .split() returns a list of substrings that were separated by the argument (default ',') in the original
  - **With Pandas:**
    - To apply a string function to an entire column (Series) of data, we use the .str syntax:
      df['Name'].str.title()
      would capitalize the first letter of each
    - These commands can be nested, like in the soda example we looked at:
      df["Soda"].str.upper().str.startswith("P").replace([True,False],["Pepsi","Coke"])
      the commands are executed from left to right, so this line first capitalizes all of the letters in the column, then makes a new column with Boolean values depending on whether the first letter of each entry is a 'P', and then finally replaces the Boolean values with Pepsi if it was a P or Coke if it wasn't.

- **Aggregation :** As we previously saw with Excel pivot tables, being able to group data by its categorical labels and then perform basic operations (like sum, max, or average) on the rows of the data that share those labels is a very powerful tool. In pandas, the basic function for performing this type of procedure is the .groupby() function, which also underlies the pandas implementation of the pivot table.
  - The .groupby() function takes as input a column label and aggregates all of the rows that share the same value in that label. This grouping can then be applied to compute functions across all rows in the same category. Many of the functions that we have looked at for analyzing individual columns can be applied to this grouping. For example,
    df.groupby('Brand')['Stars'].describe()
    returns the 5 number summary for the stars ratings of each brand in the ramen data set.
  - Multiple columns can be passed to groupby() to aggregate the data across several categorical variables at once. The rest of the function works the same way, selecting columns of numerical data to operate on and aggregation functions to apply. However, the same result can be achieved with the pivot table syntax described next.
  - To create a pivot table from a data frame, we use the pivot_table() function. This takes several arguments, corresponding to the spaces in the lower right of the Excel window for placing columns. The first entry is the numerical column to analyze (like the star rankings in the ramen example). Next, it takes the categorical columns to do the aggregation, which can either be placed along rows (index) or columns (columns). Finally, the function to apply is given with the aggfunc argument (the default is the mean). As an example:
    df.pivot_table("Stars",index="Country",columns="Style",aggfunc = max)
    will return a table with rows indexed by country and columns indexed by style, whose entries are the maximum Star rating for a ramen from that country, of that style.