Unit 4-3 Selecting and Joining

* Extracting Information by Column
  + To start, let's walk through how to extract information by column and select columns directly by name.
  + We can use pd.DataFrame() on an array of arrays to create a Pandas-style DataFrame. We'll use this DataFrame for almost all of the examples in this lesson:
    - import numpy as np
    - import pandas as pd
    - df = pd.DataFrame(np.array([[.25,10,600], [-.9,40,200], [.4,70,800], [.8,50,300]]), columns=["A","B","C"])
    - print (df)
    - Out[]:
    - A B C
    - 0 0.25 10.0 600.0
    - 1 -0.90 40.0 200.0
    - 2 0.40 70.0 800.0
    - 3 0.80 50.0 300.0
  + One of the simplest ways to extract information from a DataFrame is to pass a list of the column names you want:
    - df[["A","C"]] # selecting columns A and C
    - Out[]:
    - A C
    - 0 0.25 600.0
    - 1 -0.90 200.0
    - 2 0.40 800.0
    - 3 0.80 300.0
  + Selecting columns from a Pandas DataFrame is similar to working with key-value pairs in dictionaries. If you think of the column name as the key and the column content as the value, you'll see that the process is identical:
    - dict[key]
    - #or
    - dataframe[column]
  + Just like dictionaries, if we wanted to select several columns/keys simultaneously, we would need to pass a list:
    - dict[[key1, key2, key3]]
    - #or
    - dataframe[[col1, col2, col3]]
  + Column names can also be selected via attribute. This method, while quicker, can cause issues depending on how your columns are labeled.
    - df.A
    - out[]
    - 0 0.25
    - 1 -0.90
    - 2 0.40
    - 3 0.80
    - Name: A, dtype: float64
* Extracting Information by Row Index or Label
  + Selecting rows directly via index number or name.
    - The most straightforward way to select rows of information is by passing a DataFrame as a range.
    - df[0:3] # selects rows 0, 1 and 2
    - Out[]:
    - A B C
    - 0 0.25 10.0 600.0
    - 1 -0.90 40.0 200.0
    - 2 0.40 70.0 800.0
  + With this method, trying to extract a single row using an index value won't work.
    - df[1]
    - out[]
    - KeyError: 1
  + Instead, you'll need to use the .iloc (index location) attribute.
    - df.iloc[1]
    - out[]
    - A -0.9
    - B 40.0
    - C 200.0
    - Name: 1, dtype: float64
  + iloc can also take a range of index values. We'll cover this more in depth later in the lesson.
* Extracting a Portion of a DataFrame
  + Often when we work with a DataFrame, we don't know the exact location of our desired data. Not a problem! We simply implement conditional statements to extract them based on specific criteria. (This is where being able to code and create you own functions really comes into play.)
  + **To select DataFrame rows and columns using conditions and square brackets ([]), complete the following steps:**
    - 1) Decide on a condition that will result in a row or column being included in the subset.
    - 2) Implement the condition as a Python Boolean condition.
    - 3) Use the condition inside the square bracket to subset the DataFrame square.
* Boolean Conditions
  + Boolean conditions must return True in order for each row or column to be extracted from the DataFrame.
    - print (df) # Example df
    - Out[]:
    - A B C
    - 0 0.25 10.0 600.0
    - 1 -0.90 40.0 200.0
    - 2 0.40 70.0 800.0
    - 3 0.80 50.0 300.0
    - df['A'] > 0 # True, if row is selected to be in the new data set.
    - Out[]:
    - 0 True
    - 1 False
    - 2 True
    - 3 True
  + As we test a condition, the resulting output will be given as a Boolean.
* Using Boolean Conditions to Subset a DataFrame
  + To extract a subset, include the Boolean condition inside the square brackets of a DataFrame.
  + For example, the following output includes the contents of all rows in which the condition we're testing is True:
    - print (df)
    - Out[]:
    - A B C
    - 0 0.25 10.0 600.0
    - 1 -0.90 40.0 200.0
    - 2 0.40 70.0 800.0
    - 3 0.80 50.0 300.0
    - df[df['A'] > 0]
    - Out[]:
    - A B C
    - 0 0.25 10.0 600.0
    - 2 0.40 70.0 800.0
    - 3 0.80 50.0 300.0
* Replacing a DataFrame With Its Subset
  + In our previous examples, we extracted a subset of a DataFrame, but the DataFrame itself remained unchanged.
  + However, in some cases, we want a DataFrame to become its subset. For example, we may want to remove unnecessary columns or rows by only retaining the subset of a conditional check.
  + We can do this by reassigning the original variable (or a new variable) to the subset statement.
    - df = df[["A","C"]] # Selecting columns A and C and replacing the contents of df.
    - df
    - Out[]:
    - A C
    - 0 0.25 600.0
    - 1 -0.90 200.0
    - 2 0.40 800.0
    - 3 0.80 300.0
* Using Method Parameters to Extract Subsets
  + Many Pandas methods accept the inplace=True parameter to replace a DataFrame's contents with a subset of altered values.
  + Without the inplace=True parameter being passed, df.sort\_values() will return a sorted DataFrame as an output. However, the contents of the variable df will remain unchanged.
    - df
    - Out[]:
    - A C
    - 0 0.25 600.0
    - 1 -0.90 200.0
    - 2 0.40 800.0
    - 3 0.80 300.0
    - df.sort\_values(by="A", inplace=True)
    - print(df)
    - Out[]:
    - A C
    - 1 -0.90 200.0
    - 0 0.25 600.0
    - 2 0.40 800.0
    - 3 0.80 300.0
* One of the most common steps in data cleaning involves removing unknown, null, NA, or NaN values.
  + We've created a DataFrame with NaN values (using np.nan) to demonstrate the df.dropna() function:
    - print (df2)
    - out[]
    - A B C
    - 0 NaN NaN 6.0
    - 1 -0.9 4.0 2.0
    - 2 0.4 NaN 0.0
    - 3 0.8 5.0 3.0
    - df2.dropna(inplace=True)
    - print (df2)
    - out[]
    - A B C
    - 1 -0.9 4.0 2.0
    - 3 0.8 5.0 3.0
* Selecting Individual Cells or Cell Ranges
  + We've discussed the use of iloc, but there are other methods of selecting individual cells within a DataFrame.
    - iloc works on positions in the index (and therefore only takes integers).
    - loc works on labels in the index.
* Taking Advantage of the Index
  + When we locate cells with iloc, we're using indexing to find our desired cell(s).
  + A DataFrame always has an index. This can be an index of the automatically assigned 1, 2, 3, ... index numbers. Or, it can be an index you assign, such as a datetime column from your DataFrame.
  + An index is the fastest method of identifying a row. Index functions use a dictionary to identify the rows and columns specified by index values in iloc. This method of locating cell values is much faster than searching a column to find the right value.
* Indexing: Selecting Cells or Cell Ranges Using the Index
  + Let's look at an example of selecting individual calls using iloc:
  + The iloc method selects cells using this syntax:
  + dataframe\_name.iloc[row selection, column selection]
  + To find the cells in the 0th row of the df DataFrame, we can use this iloc statement:
    - df.iloc[0,] # Locates the cells in row 0.
    - df.iloc[0] # Locates the cells in row 0 (an all-column search is implied).
  + Similarly, these statements locate different cells within the DataFrame:
    - df.iloc[:5] # Locates rows 0–4 in df.
    - df.iloc[:5,] # Locates rows 0–4 in df.
    - df.iloc[:5,2:6] # Locates cells that only appear in both rows 0–4 and columns 2–5.
* Splitting DataFrames by Column
  + To split a DataFrame, we simply need to extract a subset from the DataFrame and assign it to a new variable. We then extract another subset containing the other values and assign it to a second variable.
  + Splitting a DataFrame at column 6:
    - new\_dataframe1 = original\_df.iloc[:, :6]
    - new\_dataframe2 = original\_df.iloc[:, 6:]
  + Splitting a DataFrame at row 5:
    - new\_dataframe1 = original\_df.iloc[:5,:]
    - new\_dataframe2 = original\_df.iloc[5:,:]
* Adding Rows and Columns in Pandas
  + Pandas provides several ways of inserting additional columns and rows.
  + You can:
    - Add new columns by assigning values to them using =.
    - Add new columns or rows using .append().
    - Add new columns or rows using .concat().
* Assigning Columns Using the Assignment Operator
  + Let's create a new column by assigning values using =.
  + Say we have a DataFrame with the columns Continent, Country, and Pageviews:
    - df
    - Out[]:
    - Continent Country Pageviews
    - 0 Europe UK 100000
    - 1 Europe DE 20000
    - 2 Africa Kenya 40000
    - 3 Africa Morocco 20000
    - 4 Africa Chad 10000
  + We'll create a new column and assign it the values of a list, just like adding a new key-value pair to a dictionary:
    - df["Unique\_visitors"] = [45000, 8000, 15000, 7000, 3000]
    - df
    - Out[]:
    - Continent Country Pageviews Unique\_visitors
    - 0 Europe UK 100000 45000
    - 1 Europe DE 20000 8000
    - 2 Africa Kenya 40000 15000
    - 3 Africa Morocco 20000 7000
    - 4 Africa Chad 10000 3000
* Using .append() to Combine Two DataFrames
  + We'll now combine two DataFrames so that the rows of the second DataFrame are added to the first. Columns not included in the first DataFrame will be added as new columns:
    - df1
    - Out[]:
    - Continent Country Pageviews
    - 0 Europe UK 100000
    - 1 Europe DE 20000
    - 2 Africa Kenya 40000
    - 3 Africa Morocco 20000
    - 4 Africa Chad 10000
    - df2
    - Out[]:
    - Continent Country Pageviews
    - 0 Asia China 110000
    - 1 Asia Japan 80000
    - 2 Asia India 40000
  + Add the rows of df2 to df1.
    - df1.append(df2)
    - Out[]:
    - Continent Country Pageviews
    - 0 Europe UK 100000
    - 1 Europe DE 20000
    - 2 Africa Kenya 40000
    - 3 Africa Morocco 20000
    - 4 Africa Chad 10000
    - 0 Asia China 110000
    - 1 Asia Japan 80000
    - 2 Asia India 40000
  + Assuming the columns are the same, the .append() method will "stack" rows on top of each other.
* Adding Columns From One DataFrame to Another
  + The .append() method was effective for adding rows from one DataFrame to another, but it only works with rows. We cannot use .append() to add columns from one DataFrame to another. Instead, we'll use the more flexible Pandas function .concat().
  + Let's try adding the columns from one DataFrame to another using pd.concat:
    - df1
    - Out[]:
    - Continent Country Pageviews
    - 0 Europe UK 100000
    - 1 Europe DE 20000
    - 2 Africa Kenya 40000
    - 3 Africa Morocco 20000
    - 4 Africa Chad 10000
    - df2
    - Out[]:
    - Unique\_visitors Num\_Moderators
    - 0 45000 18
    - 1 8000 12
    - 2 15000 11
    - 3 7000 13
    - 4 3000 15
  + Use .concat() to add the columns from df2 on the right of df1. Rows are aligned by index number, so it's crucial that, when you concatenate, your index value is a unique identifier:
    - pd.concat([df1, df2], axis = 1)
    - Out[]:
    - Continent Country Pageviews Unique\_visitors Num\_Moderators
    - 0 Europe UK 100000 45000 18
    - 1 Europe DE 20000 8000 12
    - 2 Africa Kenya 40000 15000 11
    - 3 Africa Morocco 20000 7000 13
    - 4 Africa Chad 10000 3000 15
  + Note: We set the axis parameter to 1 to specify that we want to concatenate on the column axis.
* DataFrame JOINs and Merges
  + Pandas' capabilities can often be accessed through several functions and methods. We see this with the .join() and .merge() methods.
  + We can use .join() to perform index JOINs or single-column JOINs.
  + The .join() method is less flexible than .merge(), as it only joins on the indices of the two DataFrames. But, if .join() will do the job, it'll be our first choice because it's simpler to achieve the correct settings for single-column JOINs.
* Joining DataFrames on Indices
  + Join two DataFrames by setting indices as equal to the columns to be joined:
    - df1
    - Out[]:
    - Continent Country Pageviews
    - 0 Europe UK 100000
    - 1 Europe DE 20000
    - 2 Africa Kenya 40000
    - 3 Africa Morocco 20000
    - 4 Africa Chad 10000
    - capitals
    - Out[]:
    - Country Capital
    - 0 UK London
    - 1 DE Germany
    - 2 Kenya Nairobi
    - 3 Chad N'Djamena
    - 4 Italy Rome
  + Join on Country and the indices:
    - df1.join(capitals.set\_index('Country'), on='Country')
    - Out[]:
    - Continent Country Pageviews Capital
    - 0 Europe UK 100000 London
    - 1 Europe DE 20000 Germany
    - 2 Africa Kenya 40000 Nairobi
    - 3 Africa Morocco 20000 NaN
    - 4 Africa Chad 10000 N'Djamena
  + As Country should be a unique value, we can set it as the index for the capitals DataFrame and use this index to match with the Country column in df1, thus joining the two DataFrames on that column.