Python Pandas

By: Renee Mui & Victoria Gao

What is Python Pandas?

- Python pandas is a software library that is used to manipulate data tables and series.
- Pandas is useful for
  - Cleaning data: detecting and removing missing data
  - Inserting and deleting rows or columns of data
  - Selecting part of a dataset to analyze with statistics or visualize with graphs

Installing and Importing

To install:

- Make sure you have Python 3.7.1 and above
- Run **`pip install pandas`** in your Terminal

To import:

- Include **`import pandas as pd`** at the top of your python file

Getting Data from Outside Source

Data is stored in SERIES or DATAFRAMES.

- Series are like one-dimensional arrays that can have non-numerical indexes.
  - **pd.Series(data, index, dtype)**
    - data: array that you want series to be built off of
    - index: array containing the index to use for the series
    - dtype: data type for the series
- Data frames store tabular data.
  - **pd.DataFrame(data, index, columns, dtype)**
    - columns: array containing the column labels for the frame
  - To make a data frame from a csv file: **pd.read_csv(r<path to file>)**
    - The **r** is meant to address any special characters in the file path.

Viewing Data

Both series and data frames can be previewed with the head() and tail() methods.

- **series/df.head(n)**
  - Returns the first n rows of the series/data frame
- **series/df.tail(n)**
  - Returns the last n rows of the series/data frame

Notes

- The default value of n is 5 for both methods.
- Negative numbers (-n) can be used to show all but the first/last n rows.

Cleaning Up Null Values

- To check for null data, use the **isnull()** method
  - **<series/df>.isnull()** and **pd.isnull(<series/df>)** both work
  - Returns a boolean array, with `true` for missing values and `false` for non-missing values
  - **notnull()** does the opposite

- To delete rows with null data, use the **dropna()** method
  - **series/df.dropna()**
  - Returns the new series or dataframe

- To fill up NA values, use the **fillna()** method
  - **series/df.fillna(value)**
  - value can also be a dictionary mapping an index/column to its own value to use
  - Returns the new series or dataframe

Cleaning Up Data

- To replace values with other values, use the **replace()** method
  - **series/df.replace(to_replace, value)**
  - to_replace: values/regex or list of values/regex to be replaced
  - value: new value/list of new values to replace old ones
  - A dictionary specifying which values to replace with what can be used in place of to_replace and value.
  - Returns new series/df

- To rename rows/columns, use the **rename()** method
  - **series/df.rename(index/columns=<value>)**
  - index/columns indicate which axis' names are being changed
  - value can be either a dictionary mapping old names to new ones or a function
  - Returns new series/df

Analyzing Data

To get statistics on a series or dataframe:

- **series/df.mean()**
- **series/df.max()** or **series/df.min()**
- **series/df.count()** -- returns the number of non-null values
- And so on…

For data frame specific information:

- **df.shape** returns a tuple with the number of rows and columns
- **df.info()** prints information about the index, data types, and memory usage

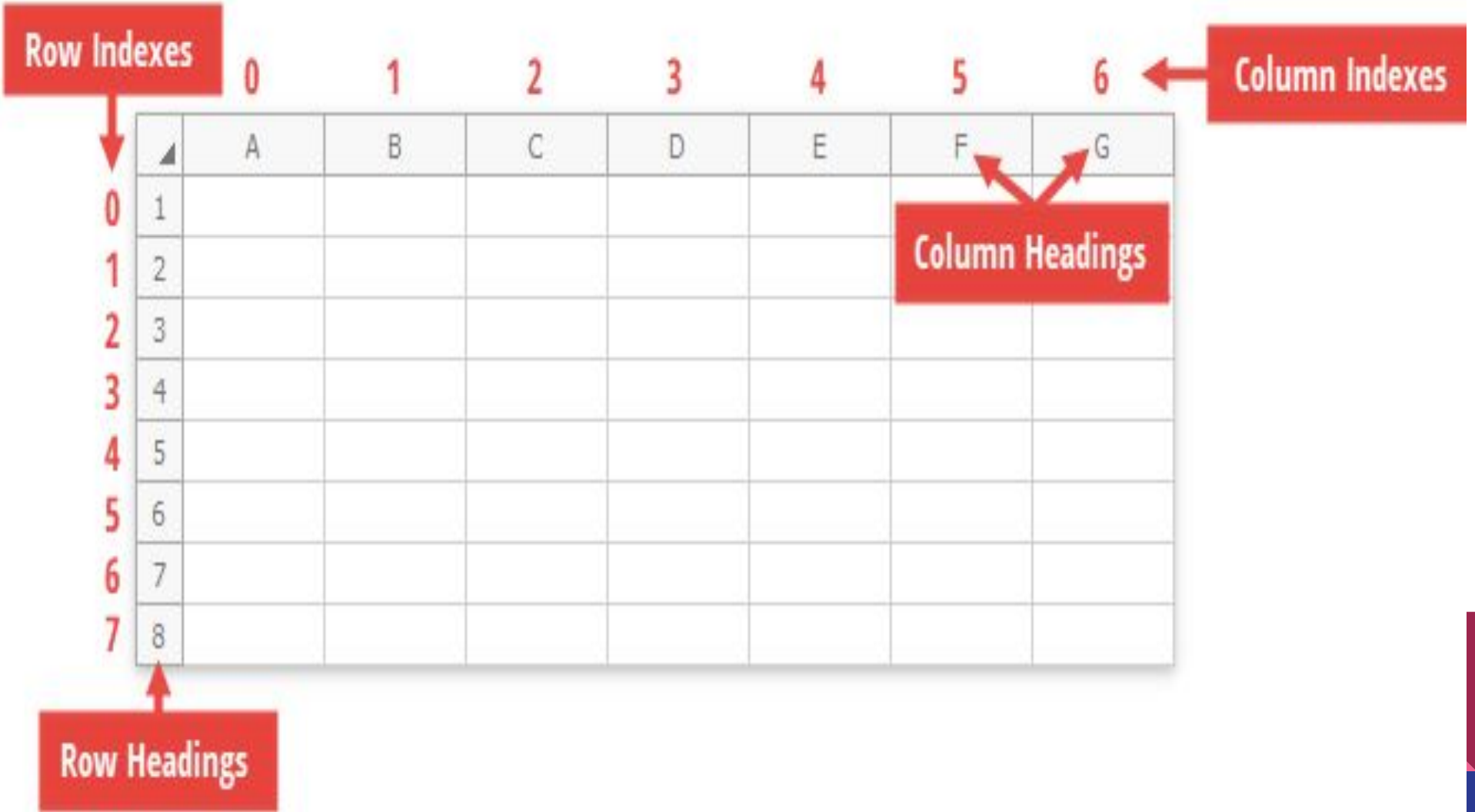
Selecting Data with .iloc[ ]

If one input is passed into iloc[ ] method, a row is returned

- **df.iloc[<integer>]** returns the row at index <integer>
- **df.iloc[[<integer_1>, <integer_2>, …]]** returns the rows at the indices specified in the array of integers [<integer_1>, <integer_2>, …]
- **df.iloc[<start>:<stop>]** returns the rows from index <start> to <stop>, excluding <stop>. <start> and <stop> are integers.

When there are two inputs, the first input specifies the row(s) and the second input specifies the column(s) in the dataframe.

- **df.iloc[<integer_1>, <integer_2>]** returns the value at row index <integer_1> and column index <integer_2>


Row Indexes
0
1
2
3
4
5
6
Column Indexes
A
B
C
D
E
F
G
1
2
3
4
5
6
7
8
Column Headings
Row Headings

Manipulating Columns

To select a column in a dataframe:

- **df[<column label>]**

To insert a column in a dataframe:

- **df.insert(loc, column, value)**
  - loc: integer that is between 0 and number of columns in the dataframe; column index
  - column: string that is the column label
  - value: string/integer/array that contains the data values in the column

To delete a column in a dataframe:

- **df.drop(columns)**
  - columns: string or array containing column labels in the dataframe

Filtering

- We can filter rows in a dataframe by placing boolean expressions inside square brackets [ ] after a dataframe name.
- Ex: movies[movies["Rating"] > 6] returns data for movies with a rating that is greater than 6

Grouping and Aggregating Data

- **df.groupby(‘<column_label>’)** divides the data into categories.
- **df.groupby(‘<column_label>’).groups** is a dictionary whose keys are the unique categories.
- After defining a GroupBy object, we can use aggregation to perform arithmetic operations on a column for each category.
- Ex: If we divided a dataframe containing movie data based on genre, we can perform operations like mean and max on the Ratings column to find the average and highest ratings for each movie genre.

df.groupby(‘<category>’).agg({‘column’:[<operation>]})

Data Visualization

- To better visualize the trends in data, we can make graphs with **.plot()** method
- **df.plot(kind = “<graph_type>”)** where **<graph_type>** can be
  - “bar”: vertical bar plot
  - “barh”: horizontal bar plot
  - “line”: line graph
  - “hist”: histogram
  - “pie”: pie chart
  - “box”: boxplot
  - “scatter”: scatterplot
- Let **graph = df.plot(kind = “<graph_type>”)**
  - **graph.set_xlabel("<x-label>")** adds an x-label to the plot
  - **graph.set_ylabel("<y-label>")** adds a y-label to the plot

Real-world Applications

- After cleaning, inserting, deleting, filtering, and selecting parts of a large dataset with Python Pandas, users can further analyze the data with other Python packages such as Matplotlib and NumPy.
- Prediction of Stocks: Create models from large, organized datasets that can predict behavior and price of stocks given independent variables like date and time, interest rates, inflation, etc.
- Classification of Cancer Samples: Use machine learning algorithms to find trends in large, organized genetic and patient health datasets that can be used to classify the stage and subtype of cancer a patient has.

Thank You

To learn more about Python Pandas, visit
https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html