Intro to Pandas

By Kasaboushi

What is Pandas?

Pandas is a python library intended to be used for data science, and it helps users perform data-related operations.

Pandas makes it easy to generate tables and charts, and is very good for manipulating time-based data.

This course will serve as a brief introduction to Pandas as a module.

What is Pandas?

With Pandas, you can:

- Generate tables and lists from dictionaries and files
- Display,organize, and search for data
- Easily calculate attributes of datasets such as mean, median and mode
- Output datasets as graphs with matplotlib
- And more…

Github Codespaces

- This course will make use of Github Codespaces to deliver lessons.
- Codespaces are cloud-based containers that can be shared and attached to repositories.
- Codespaces can be specialized specifically for individual projects and repositories.
- They also simplify sharing projects.
- To troubleshoot, it will be good to have knowledge of codespaces, but is not strictly required.
- Problems are located in the folder labeled “Problems”.

Getting Started

If you are using the attached codespace, the steps below have already been done.

For your own projects, you will have to install Python, then install Pandas.

To install Python, go to python.org/downloads/ and download the appropriate installer.

From there, the easiest way to install Pandas is through pip, a built-in tool that allows one to easily install files from online repositories.

- Type `pip install pandas` in a command line or in a terminal that can run code. Pandas should then automatically install itself.
- pip can be used for most libraries.

Lastly, import pandas into your python script: `import pandas` .

- Pandas is often imported as `pd` to save time, so consider `import pandas as pd`

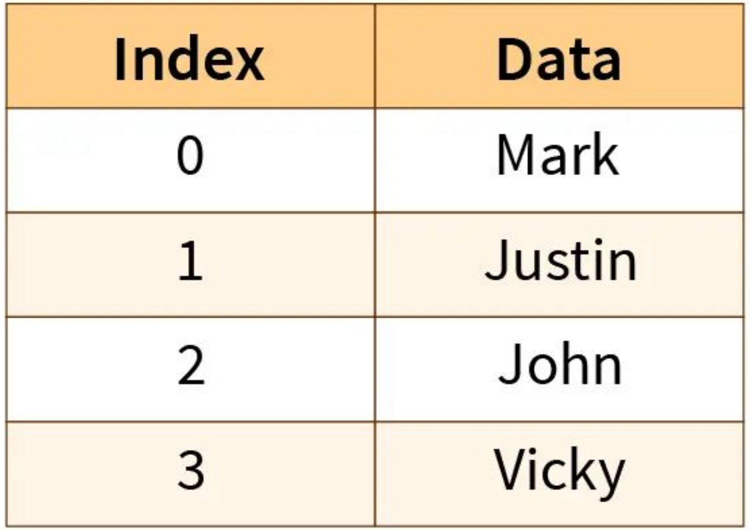
We have provided the file `testing.py` for you to test commands and code in the home directory of the codespace.

Data Structures: Series

- Pandas has two primary types of data structure that can be used in most situations: **Series** and **DataFrame**.
- A **Series** can be created from lists, dictionaries or other data inputs. They are essentially lists with labeled columns.

`pd.Series(data)` is the command to generate a series, in which data is some form of existing data that is inputted.

The example to the right shows a Series of strings and their corresponding indexes.



Index	Data
0	Mark
1	Justin
2	John
3	Vicky

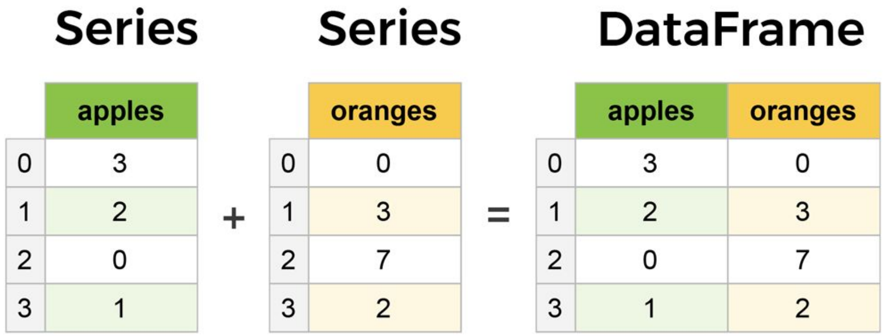
Data Structures: DataFrames

- **DataFrames** are similar to tables: They hav rows and columns, and store data in each "cell" like a table.
- They can also be thought of as multiple Series grouped together.
- DataFrames are made from dictionaries or arrays with `pd.DataFrame(data)`.

DataFrames can be made from files, such as a CSV file. Try the statement below in testing.py.

```
df = pd.read_csv("test.csv")
```

Then, print using the command `df.head()`.



Series

	apples
0	3
1	2
2	0
3	1

+

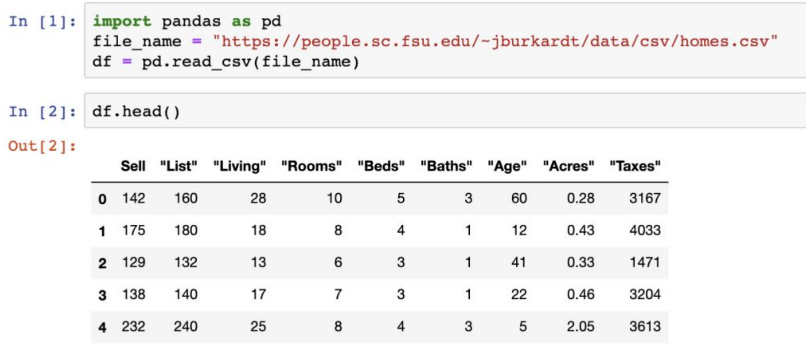
Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2



```
In [1]: import pandas as pd
        file_name = "https://people.sc.fsu.edu/~jburkardt/data/csv/homes.csv"
        df = pd.read_csv(file_name)

In [2]: df.head()

Out[2]:
```

	Sell	"List"	"Living"	"Rooms"	"Beds"	"Baths"	"Age"	"Acres"	"Taxes"
0	142	160	28	10	5	3	60	0.28	3167
1	175	180	18	8	4	1	12	0.43	4033
2	129	132	13	6	3	1	41	0.33	1471
3	138	140	17	7	3	1	22	0.46	3204
4	232	240	25	8	4	3	5	2.05	3613

Data Structures: Rows and Columns

- Both Series and DataFrames use a system of rows and columns as identifying characteristics for individual data points.
- Built in functions allow one to access data based on row and column. Columns and rows both have labels that can be edited.
- Custom labels can be set by adding an index argument.
  - `pd.DataFrame(data, index = [“one”,”two”,”three”])`
- Then, `df.loc[index]` can be used to return all rows under that index.

Given DataFrame `df`:

`df.loc[0]` returns everything in the 0th row

`df.loc[0,1]` returns everything in the 0th and 1st rows

`df.head()` will display the first five rows, and `df.tail()` will display the last five.

`df.index` and `df.columns` will display rows and columns respectively.

Selecting Specific Items

DataFrames work similarly to arrays and lists in that you can pass in arguments to get specific items. Specifically, they are similar to 2D arrays since they use both arrays and columns.

`df[label]`, where label is a string, will return every value stored under that column label. Passing in a slice, such as in `df[0:1]` will return every value between the rows, inclusive.

`df.loc[labels]` will return rows under the given row labels. Passing in a row argument will select a specific row.

Progress Check: Try completing `problems1.py`

Selecting Items with Criteria

Putting conditionals in selections can allow one to filter the selected items.

- `df[df["A"] > 0]` will select only rows with an A value greater than 0. This can be applied to the entire DataFrame.
- `isin()` can be applied as a selection condition.
- `df.loc[row, column]` will select specific items.

Try selecting items from the sample dataset and printing them.

Try completing `problems2.py`

Data Cleaning

Data Cleaning is the practice of fixing incorrect elements in a data set, and it is one of the primary functions of Pandas.

Potential errors that can be fixed by cleaning include:

- Empty cells
- Data in the wrong format
- Incorrect Values
- Duplicate entries

Data Cleaning is an important process for general data science.

`df.info()` will show information about a dataframe, including the number of non-null values.

Empty Cells

Empty cells can cause analysis issues. There are multiple ways to deal with empty cells.

- One way to deal with empty cells is to simply drop the row altogether. This can be done with the command `df.dropna()`, which drops all rows with null values. `dropna()` returns a new DataFrame unless the inplace parameter is set to true. If true, the original will be edited.
- Empty cells can be replaced with a value through `fillna(value, inplace = True)`
- Specified columns can be replaced by specifying column names such as in `df.fillna({"Column":, inplace=True})`
- Some people use mean, median, or mode to replace values:

```
x = df["Calories"].mean()

df.fillna({"Calories": x}, inplace=True)
```

Wrong Data/Format

- Improperly formatted data can make analysis impossible. You can either remove rows or convert the rows into the proper format.
- If working with date and time data, the `to_datetime()` can convert to the proper format.
  - `df['Date'] = pd.to_datetime(df['Date'], format='mixed')`
- However, if not working with time data, simply remove the row with `dropna()`
- If data is simply wrong, it can be edited with: `df.loc[row,column] = value.` This will set the data point to the value.
  - With large data sets, a looping approach is necessary. You can set parameters for replacement and use a for-each loop to go through the entire set.
- Lastly, `df.drop(row, inplace = True)` can remove rows entirely.

Duplicate Cleaning

The `duplicated()` method looks for duplicate rows in the data set and returns a boolean for each row.

`df.drop_duplicates(inplace = True)` will drop all duplicates.

Try the problems in `cleaning.py`.

Next, we will lastly demonstrate how to make graphs and plots.

Plots

Pandas uses `plot()` to make diagrams in conjunction with `matplotlib`. Install `matplotlib` and then import it. For simplicity, we will be importing `matplotlib` as `plt`.

```
df.plot()

plt.show()
```

This is the basic format of the plot generation.

Arguments can be added to the plot function as `df.plot(kind,x,y). X` and `y` change axis names, and `kind` specifies the type of graph (scatter for a scatterplot, hist for histogram, etc.).

Complete `plots.py.`

Analysis

If someone wants to use pandas in statistics, there are some key basic functions that statisticians will use – namely, mean, median, and mode.

`df.mean()`, `df.median()`, and `df.mode()` are the functions required. You can add the `numeric_only = True` argument to only target columns with numeric data.

The `df.std()` method will output the standard deviation of each column. You can also include axis arguments to output the standard deviation of each row.

- `dataframe.std(axis, skipna, ddof, numeric_only)`

Complete `analysisProblems.py.`

Thank You

This concludes the introductory Pandas course. Thank you for taking it, and I hope this course gave you some basic knowledge about using Pandas for data science.

There are plenty of resources online regardings Pandas, so if you want to continue learning, consider searching for more tutorials or work on a project.