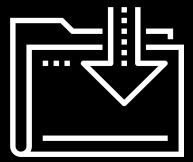
Introduction to Pandas and Jupyter

Data Boot Camp

Lesson 4.1



Class Objectives

By the end of today's class, you will be able to:

Serve Jupyter Notebook files from local directories and connect to their development environment.

Create Pandas DataFrames from scratch.

Run functions on Pandas DataFrames.

Read and write DataFrames to and from CSV files by using Pandas.

Instructor Demonstration

Introduction to Jupyter Notebook

Introduction to Jupyter Notebook

Before diving into Pandas, let's learn about Jupyter Notebook.

Jupyter Notebook is an open-source application that allows its users to create documents that contain live code, equations, visualizations, and explanatory text.

In other words, Jupyter Notebook combines a text editor, the console, and a markdown file into one application.

Connecting Jupyter Notebook to a Virtual Environment

Follow these steps:

- Create a Python file with Jupyter Notebook. Set the kernel as ‘PythonData’.
- Setting the kernel for Jupyter projects is important because these kernels let the program know which libraries are available for use.
- Only those libraries loaded into the selected development environment can be used in a Jupyter Notebook project.

Introduction to Jupyter Notebook

Before diving into Pandas, let's learn about Jupyter Notebook.

- Understanding the structure of Jupyter Notebook files is key to navigation.
- Each cell contains Python code that can be run independently by placing the cursor inside a cell and pressing Shift + Enter.
- Jupyter notebooks allow users to experiment with code directly and save it for later.
- The values in Jupyter notebooks are stored based on what lines of code were run last.

Activity: Comic Book Remix

In this activity, you will create a Jupyter notebook that performs the same functions as the Comic Book activity from the previous unit.

Suggested Time:

15 minutes

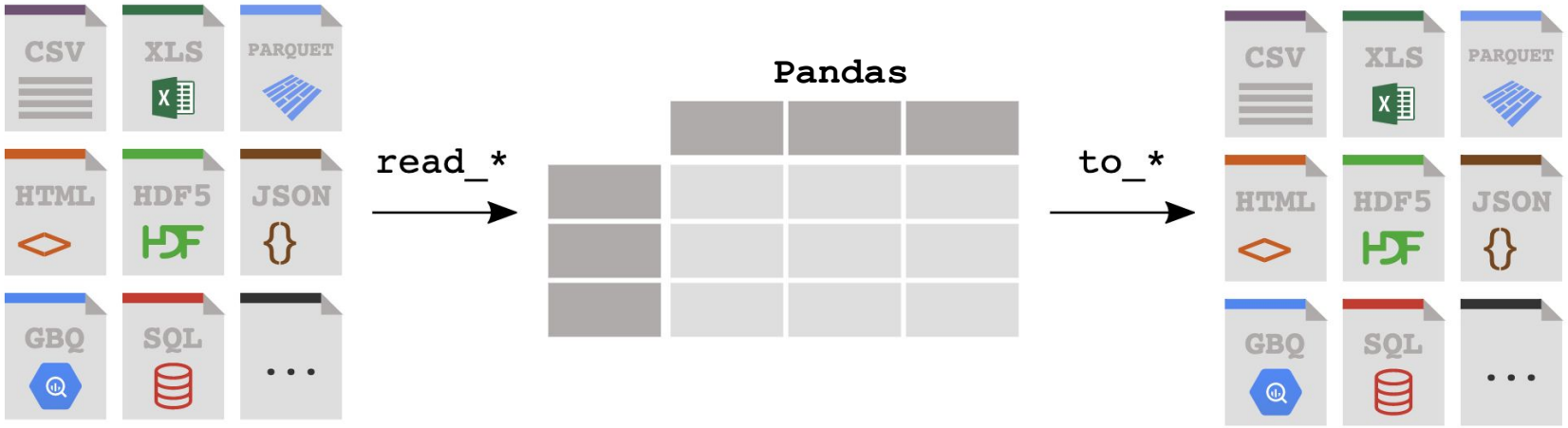
Time's Up! Let's Review.

Instructor Demonstration

Introduction to Pandas

Introduction to Pandas

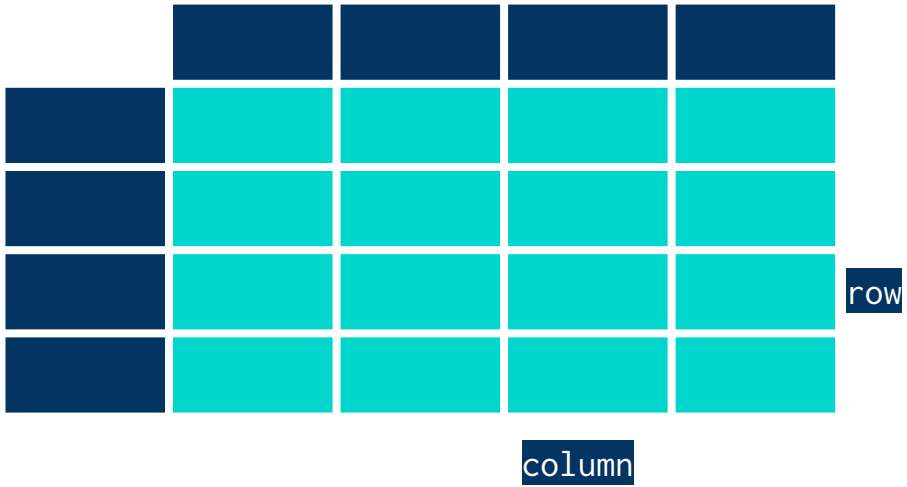
Modifying large datasets in Python can be challenging. Thankfully, the Pandas library is extremely powerful when it comes to visualizing, analyzing, and altering large datasets.



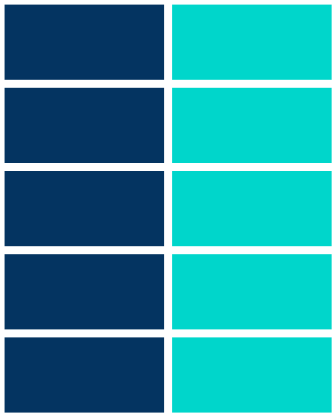
Introduction to Pandas

Although Python alone is stuck using lists, tuples, and dictionaries, Pandas lets Python programmers work with Series and DataFrames. A **DataFrame** is a table with rows and columns. Each column in a DataFrame is a **Series**.

DataFrames



Series



Instructor Demonstration

DataFrame Creation

DataFrame Creation

01

Import the Pandas library by using `import pandas as pd`.

This method of import allows Pandas functions/methods to be called by using the variable `pd`.

02

To create a Series, run the `pd.Series()` function and place a list within the parentheses. The index for the values in the Series will be the numeric index of the initial list.

DataFrame Creation

03

One way (of many) to create DataFrames from scratch is to use the `pd.DataFrame()` function and provide it with a list of dictionaries. Each dictionary will represent a new row where the keys become column headers and the values are placed inside the table.

04

Another way to use the `pd.DataFrame()` function is to provide a dictionary of lists. The keys of the dictionary will be the column headers and the listed values will be placed into their respective rows.

Activity: DataFrame Shop

In this activity, you will create DataFrames from scratch by using the two methods just discussed.

Suggested Time:

15 minutes

Time's Up! Let's Review.

Instructor Demonstration

DataFrame Functions

Built-In Pandas Function: `head()`

The `head()` method is helpful because it allows the programmer to view a minified version of a much larger table; then, they can make informed changes without searching through the entire dataset.

```
# Use Pandas to the read data.
data_file_df = pd.read_csv(data_file)
data_file_df.head()
```

	id	Full Name	Gender	Amount	Car
0	1	Minnnie Rean	male	15484.5	Jeep
1	2	Ursa Torricella	female	13443.3	Saturn
2	3	Joyann Pirolini	male	9095.6	Ram
3	4	Sharl Ridsdell	female	11871.6	Dodge
4	5	Laurence Jovasevic	male	13459.8	Chrysler

Built-In Pandas Function: `describe()`

The `describe()` method prints out a DataFrame containing some analysis of the table and its columns. It also indicates some of the other data functions can be performed on a DataFrame or Series.

```
# Display a statistical overview of the DataFrame.
data_file_df.describe()
```

	id	Amount
count	1000.000000	1000.000000
mean	500.500000	9988.738100
std	288.819436	5783.375372
min	1.000000	15.300000
25%	250.750000	5043.150000
50%	500.500000	9899.500000
75%	750.250000	15044.225000
max	1000.000000	19927.900000

Working with a Single Column

Most data functions can also be performed on a Series by referencing a single column within the whole DataFrame.

Similar to referencing a key within a dictionary, take the DataFrame and follow it with brackets that contain the desired column's header.

```
# Reference a single column within a DataFrame.
data_file_df["Amount"].head()
```

```
0    15484.5
1    13443.3
2     9095.6
3    11871.6
4    13459.8
Name: Amount, dtype: float64
```

Working with Multiple Columns

Multiple columns can be referenced by placing all of the column headers desired within a pair of double brackets. If two sets of brackets are not used, then Pandas will return an error.

```
# Reference multiple columns within a DataFrame.
data_file_df[["Amount", "Gender"]].head()
```

	Amount	Gender
0	15484.5	male
1	13443.3	female
2	9095.6	male
3	11871.6	female
4	13459.8	male

A Few Aggregating Functions

`.mean()`	computes the mean
`.sum()`	adds the values

```
# The mean method averages the series
average = data_file_df["Amount"].mean()
average
```

```
9988.738099999993
```

```
# The sum method adds every entry in the series
total = data_file_df["Amount"].sum()
total
```

```
9988738.100000001
```

Built-In Pandas Function: `unique()`

In some situations, it's helpful to list all of the unique values stored within a column. This is precisely what the `unique()` function does: it looks into a Series and returns all of the different values contained in it.

```
# The unique method shows every element only once
unique = data_file_df["Car"].unique()
unique
```

```
array(['Jeep', 'Saturn', 'Ram', 'Dodge', 'Chrysler', 'Cadillac',
       'Pontiac', 'Nissan', 'Lexus', 'Volkswagen', 'Suzuki', 'Kia',
       'Mercury', 'Audi', 'Bugatti', 'BMW', 'Mazda', 'GMC', 'Ford',
       'Mercedes-Benz', 'Land Rover', 'Chevrolet', 'Toyota', 'Honda',
       'Subaru', 'Oldsmobile', 'MINI', 'Lincoln', 'Mitsubishi', 'Isuzu',
       'Infiniti', 'Eagle', 'Saab', 'Buick', 'Volvo', 'Lotus', 'Maserati',
       'Jensen', 'Hyundai', 'Maybach', 'Corbin', 'Acura', 'Ferrari',
       'Plymouth', 'Studebaker', 'Jaguar', 'Rolls-Royce', 'Aston Martin',
       'Merkur', 'Citroën', 'Daewoo', 'Tesla', 'Porsche', 'Scion', 'Geo',
       'Hummer', 'Lamborghini', 'Fiat', 'Bentley', 'Peugeot', 'Austin',
       'Spyker'], dtype=object)
```

Built-In Pandas Function: `value_counts()`

Another method with similar functionality is `value_counts()`, which not only returns a list of all unique values within a Series, but also counts how many times a value appears.

```
# The value_counts method counts unique values in a column
count = data_file_df["Gender"].value_counts()
count
```

```
male          455
female        446
non-binary     99
Name: Gender, dtype: int64
```

Beyond Pandas Visualization Power

Calculations can also be performed on columns and then added into the DataFrame as a new column. This is done by referencing the DataFrame, placing the desired column header within brackets, and then setting it equal to a Series.

```
# Calculations can also be performed on Series and added into DataFrames as new columns
thousands_of_dollars = data_file_df["Amount"]/1000
data_file_df["Thousands of Dollars"] = thousands_of_dollars

data_file_df.head()
```

	id	Full Name	Gender	Amount	Car	Thousands of Dollars
0	1	Minnnie Rean	male	15484.5	Jeep	15.4845
1	2	Ursa Torricella	female	13443.3	Saturn	13.4433
2	3	Joyann Pirolini	male	9095.6	Ram	9.0956
3	4	Sharl Ridsdell	female	11871.6	Dodge	11.8716
4	5	Laurence Jovasevic	male	13459.8	Chrysler	13.4598

Activity: Training Grounds

In this activity, you will now take a large DataFrame containing 200 rows, analyze it with data functions, and then add a new column into it.

Suggested Time:

15 minutes

Time's Up! Let's Review.

Instructor Demonstration

Column Manipulation

Modifying Columns

Column manipulation

An easy way to modify the names or placement of columns is to use the `rename()` function and double brackets.

To collect a list of all the columns contained within a DataFrame, use the `'df.columns'` call, and an object containing the column headers will be printed to the screen.

```
# Collect a list of all columns within the DataFrame
training_df.columns
```

```
Index(['Name', 'Trainer', 'Weight', 'Membership(Days)'], dtype='object')
```

Modifying Columns

Column manipulation

To reorder the columns, create a reference to the DataFrame followed by two brackets with the column headers placed in the desired order.

It's also possible to remove columns simply by **not** creating a reference to them. This will, in essence, drop them from the newly created DataFrame.

```
# Reorganize the columns using double brackets
organized_df = training_df[["Name","Trainer","Weight","Membership(Days)"]]
organized_df.head()
```

	Name	Trainer	Weight	Membership(Days)
0	Gino Walker	Bettyann Savory	128	52
1	Hiedi Wasser	Mariah Barberio	180	70
2	Kerrie Wetzel	Gordon Perrine	193	148
3	Elizabeth Sackett	Pa Dargan	177	124
4	Jack Mitten	Blanch Victoria	237	186

Modifying Columns

Column manipulation

To rename the columns within a DataFrame, use the `df.rename()` method and place `columns={}` within the parentheses.

Inside the dictionary, the keys should be references to the current columns, and the values should be the desired column names.

```
# Use .rename(columns={}) to rename columns
renamed_df = organized_df.rename(columns={"Membership(Days)":"Membership in Days",
                                          "Weight":"Weight in Pounds"})
renamed_df.head()
```

	Name	Trainer	Weight in Pounds	Membership in Days
0	Gino Walker	Bettyann Savory	128	52
1	Hiedi Wasser	Mariah Barberio	180	70
2	Kerrie Wetzel	Gordon Perrine	193	148
3	Elizabeth Sackett	Pa Dargan	177	124
4	Jack Mitten	Blanch Victoria	237	186

Activity: Hey Arnold!

In this activity, you will take a premade DataFrame of *Hey Arnold!* characters and reorganize it so that it's easier to understand.

Suggested Time:

10 minutes

Time's Up! Let's Review.


Break

Instructor Demonstration

Reading and Writing CSV Files

Reading and Writing CSV Files

A CSV file's path is created and passed into the `pd.read_csv()` method, with the returned DataFrame stored within a variable.

```
# Read our data file with the Pandas library
# Not every CSV requires an encoding, but be aware this can come up
file_one_df = pd.read_csv(file_one, encoding="ISO-8859-1")
```

```
# Show the first five rows.
file_one_df.head()
```

	id	full_name	email	gender
0	1	Jacquenette Nesterov	jnesterov0@squarespace.com	female
1	2	Leanora Cashell	lcashell1@blogger.com	male
2	3	Arley Medford	amedford2@newyorker.com	male
3	4	Rafaello Crawshaw	rcrawshaw3@multiply.com	male
4	5	Karalee Hallaways	khallaways4@uol.com.br	non-binary

Reading and Writing CSV Files

It's just as easy to write to a CSV file as it is to read from one.

Simply use the `df.to_csv()` method, and pass the path to the desired output file. By using the index and header parameters, programmers can also choose whether they would like the index or header for the table to be passed as well.

```
# Export file as a CSV, without the Pandas index, but with the header
file_one_df.to_csv("Output/fileOne.csv", index=False, header=True)
```

Activity: Comic Books Part 1

In this activity, you will take a large CSV file containing comic books, read it into Jupyter Notebook by using Pandas, clean up the columns, and then write a modified DataFrame to a new CSV file.

Suggested Time:

20 minutes

Time's Up! Let's Review.

Activity: Comic Books Part 2

In this activity, you will take the modified version of the comic book DataFrame and create a new summary DataFrame based on that dataset, using some of Pandas' built-in data functions.

Suggested Time:

20 minutes

Time's Up! Let's Review.

Questions?