FOR PRESENTER USE ONLY

Exploratory Data Analysis With Pandas

Overview

In this lesson, students will begin using Pandas for exploratory data analysis. This will include filtering and sorting data to generate insights.

Duration

120 minutes

Learning Objectives

In this lesson, students will:

- Use Pandas to read in a data set.
- Use DataFrame attributes and methods to investigate a data set's integrity.
- Apply filters and sorting to DataFrames.

FOR PRESENTER USE ONLY

Pre-Class Materials and Preparation

For remote classrooms: Virtual breakout rooms and Slack may be needed to facilitate the partner exercise and discussions. As you plan for your lesson:

- Consider how you'll create pairs for the partner exercise (randomly, or with pre-assigned partners).
- Determine how (if at all) exercise timing may need to be adjusted.
- For helpful tips, keep an eye out for the **For remote classrooms** tag in the speaker notes.
- Prepare screenshots and answers to exercises in advance so that they can be easily shared in Slack during your lecture.

FOR PRESENTER USE ONLY

Suggested Agenda

Time	Activity
0:00–0:50	**Welcome + Meet Pandas**
0:50–1:00	**Break**
1:00–1:50	**Filtering and Sorting Data**
1:50–2:00	**Wrapping Up, Q&A, and Exit Ticket Completion**

FOR PRESENTER USE ONLY

Jupyter Notebook

The exercises referred to in this lesson can be found in the Python Workbooks + Data folder.

Exploratory Analysis With Pandas

GENERAL ASSEMBLY

Our Learning Goals

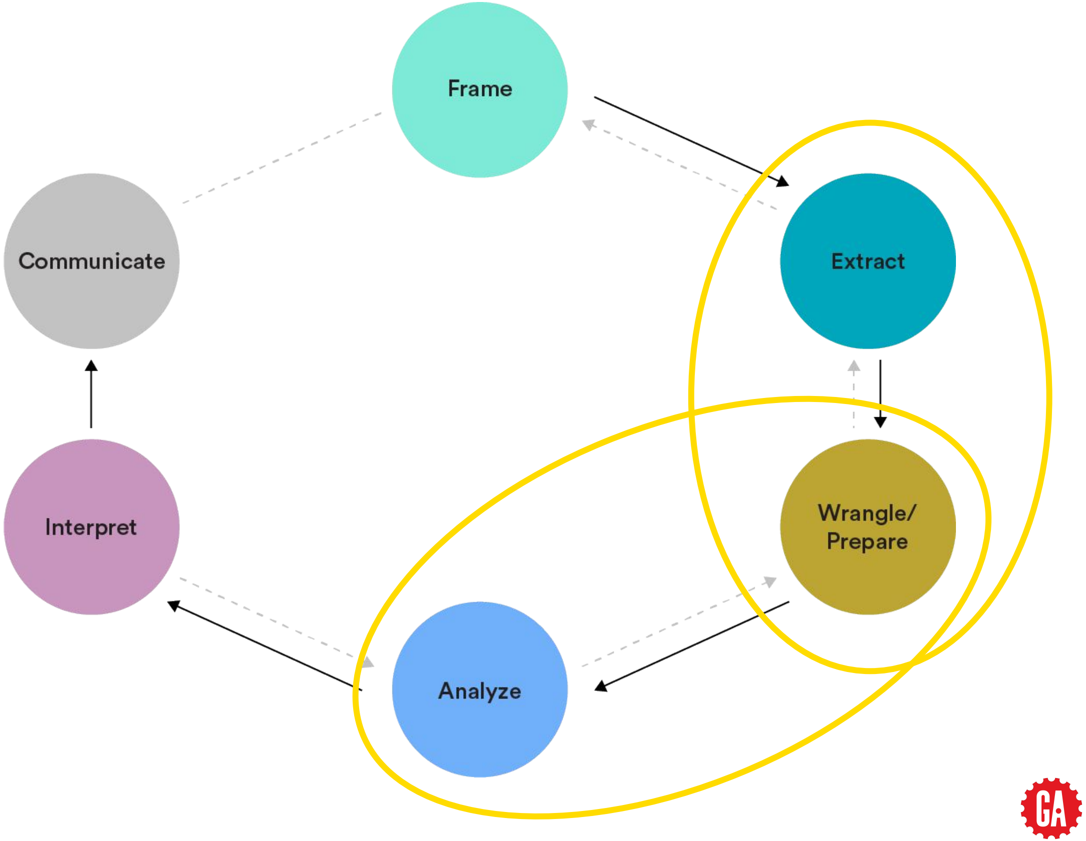
- Use Pandas to read in a data set.
- Use DataFrame attributes and methods to investigate a data set’s integrity.
- Apply filters and sorting to DataFrames.

The Data Analytics Workflow

Extract: Select and import relevant data.

Wrangle/Prepare: Clean and prepare relevant data.

Analyze: Structure, comprehend, and visualize data.



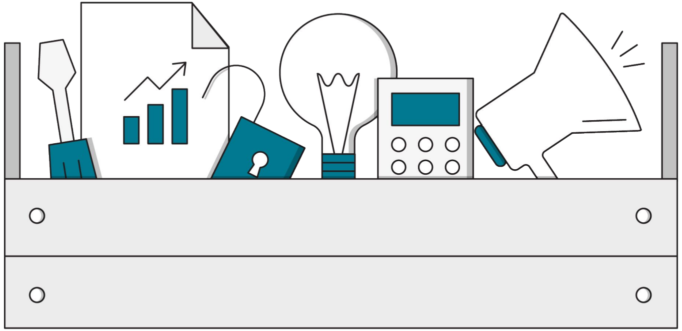
Exploratory Data Analysis With Pandas

Meet Pandas

What Is Pandas?

Pandas is the most prominent Python library for **exploratory data analysis (EDA)**. We use Pandas to investigate, wrangle, and clean our data.

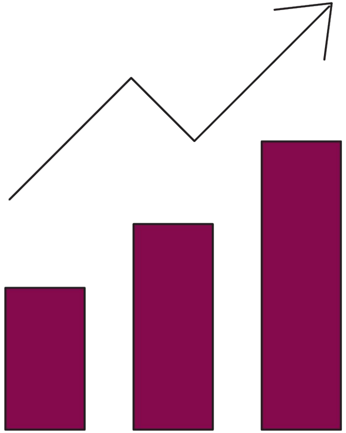
Pandas is a versatile toolbox that can be used for all of our data exploration needs. (Think Excel or Google Sheets, but much faster and with way more flexibility!)



Exploratory Data Analysis: Definition

In a nutshell, exploratory data analysis (EDA) means "**getting to know" a data set**. This can include:

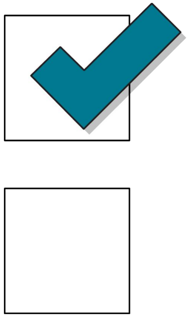
- **Checking data types** to make sure data is stored properly.
- **Calculating summaries for columns**, like the average, minimum, or maximum.
- Evaluating your data set for **missing data**.
- Identifying potential **trends or outliers**.
- **Basic visualization** of your data.



Exploratory Data Analysis Best Practices

At the very least, as part of EDA, you should determine:

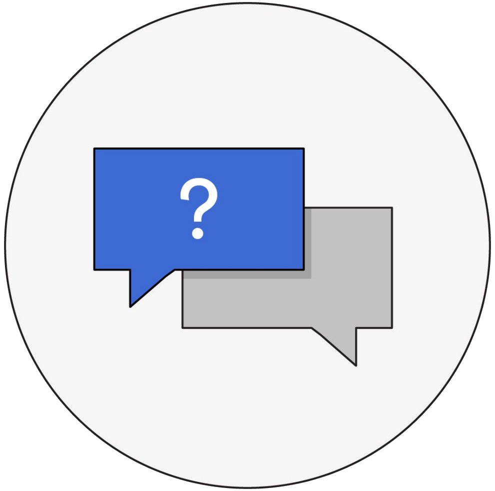
- **The number of rows** in the data set.
  - **What does each row represent?** Is each row a person, an observation, a time point?
- **The number of columns** in the data set.
  - **What does each column represent? How was that data collected?** *Try using a data dictionary — it can often directly answer these questions for you!*



From Questions to Hypotheses

Start by asking yourself…

- What **fields can I COMBINE** to find interesting insights?
- What **ACTIONS can someone take** as a result of my charts and analyses?



Reading a Data Set

Pandas makes it easy to import a data set using a method for reading .csv files.

```
import pandas as pd

data_frame = pd.read_csv(file_address,sep=delimiter_character)
```

Note: Not all .csv files use commas! **sep** defines the character used to separate values.

Series vs. DataFrames

Pandas relies on two key objects, each with their own methods and properties:

- A **Series**, accessed using single square brackets, is an attribute of the larger DataFrame object and can only access one column.
  - **data_frame['column_name']**

- A **DataFrame**, accessed using double square brackets, treats the output as its own smaller table and can include multiple columns.
  - **data_frame[['column_name']]**
  - **data_frame[['column_a', 'column_b']]**

Accessing and Modifying the Index

Much like lists, DataFrame rows also have an **index** to keep of them.

While the default index starts at zero *(remember that Python starts counting at zero!)*, there may already be a column in the data itself that we'd prefer to use as the index.

```
data_frame.index
```

```
data_frame.set_index(column_name, inplace=True)
```

Columns and Data Types

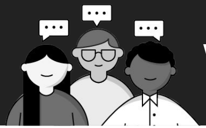
One of the first things we want to learn about a new table is what columns and data types we'll be working with in that table.

```
data_frame.columns # Prints all the column names.

data_frame.dtypes # Prints all the data types, but is hard to read!
```

```
# Easy-to-read DataFrame of the data types:

pd.DataFrame(data_frame.dtypes, columns=['DataType'])
```



Discussion:

Why Investigate Data Types?

Why would we be interested in the data types as one of our first questions?

What operations might we perform in response to the data types we see?

Renaming Columns

We can use the **`.rename()`** method to provide a dictionary of replacement names:

```
df.rename(columns={'OldName': 'NewName'}, inplace=True).head(3)
```

The **inplace** option determines whether we're creating a new DataFrame or modifying the original directly. **`inplace=True`** means we are overwriting the original!

Common Column Operations

	What This Method Does
`.describe()`	Provides summary attributes, including maximum, minimum, mean, and the 25%, 50%, and 75% quartile values (for numeric columns) or most frequent value (for categorical columns).
`.value_counts()`	Counts the number of occurrences of each value in the column.

`.unique()/`
`.nunique()`
	Provides a list of unique values or the number of unique values.

Guided Walk-Through:

8.1 Exploring the Superstore Data

15 minutes

We'll use the Superstore data set to practice exploring data with Pandas.

We will be looking at a single table from this database — the "Orders" table — to explore its properties.

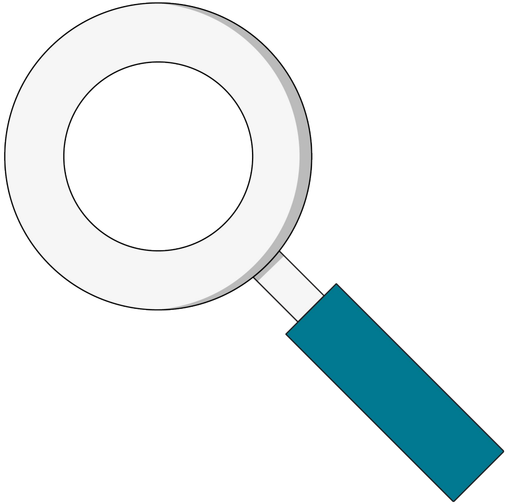
Partner Exercise:

8.2 Explore Another Table

20 minutes

Answer the questions in Section 8.2 by using the column operations methods.

You may want to start with some of the same exploratory methods we just used.



Exploratory Data Analysis With Pandas

Filtering and Sorting Data

Filtering on One Condition

Filtering and sorting are important steps that allow us to drill into subsets of our data.

To filter, we use a process called **Boolean filtering**, wherein we first define a Boolean mask and then use it to filter our DataFrame.

The Boolean Mask

We can create a whole series of **True** and **False** values by using a comparison statement with the column:

```
df['Column'] == 'ValueDesired'
```

Index	
1	False
2	False
3	False
4	False
5	True
6	True
7	True
8	False

The Boolean Mask (Cont.)

Once we have a Boolean series, we can then filter the entire data set for only those values with a **True** result:

```
df[df['Column'] == 'ValueDesired'].tail(3)
```

Index	Column	Price
317	ValueDesired	10.99
318	ValueDesired	5.00
319	ValueDesired	2.75

DataFrame Syntax Chaining

When filtering with this syntax, the result is a DataFrame.

This means that we can continue accessing other columns or using methods:

```
# Access the Price column of all results passing the filter:

d[df['Color'] == 'Purple']['Price']

# Gives numerical summaries based only on results passing the filter:

df[df['Color'] == 'Purple']['Price'].describe()
```

Filtering by Multiple Conditions

Adding more conditions uses the same syntax, even if it looks more complicated:

```
df[(df['Color'] == 'Silver') & (df['ListPrice'] > 350)]
```

```
df[(df['ListPrice'] < 3000) | (df['ZipCode'] == '95404')]
```

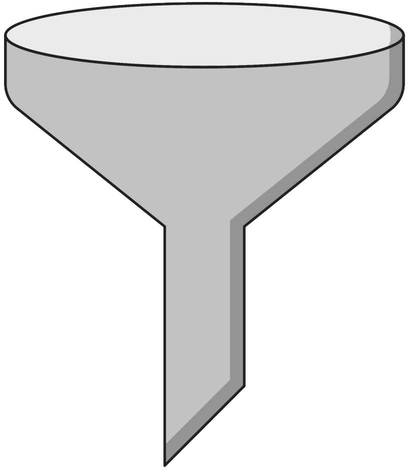
The parentheses here are very important! Leaving them out will usually trigger an error.

Solo Exercise:

8.3 Boolean Filtering

15 minutes

Use Boolean filtering to narrow down the DataFrame in Section 8.3 of your workbook.



Sorting

The magic of libraries like Pandas is that there’s a method for almost everything. In the case of sorting, we have a **`sort_values()`** method:

```
# For a Series object, no need to specify column: There's only one!

data_series.sort_values()

# For a DataFrame, it will sort by index unless given a column name.

data_frame.sort_values(by='column_name', descending=True)
```

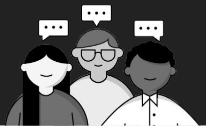
Accessing an Individual Row

Now that we can sort and filter, it's more likely that we might want to access a single row of data from a DataFrame. We can use the **`iloc`** property to use indexing syntax, like so:

```
data_frame.sort_values(by="points_scored").iloc[ [0] ]
```

Note the two brackets, which mean that the result will be a DataFrame. We could also simply use one bracket to instead return a Series with accessible properties:

```
data_frame.sort_values(by="points_scored").iloc[0]["player_name"]
```



Discussion:

Key Definitions Recap

While working with Pandas, it will be important to distinguish between Series and DataFrames objects.

What are the key properties of each type of object?

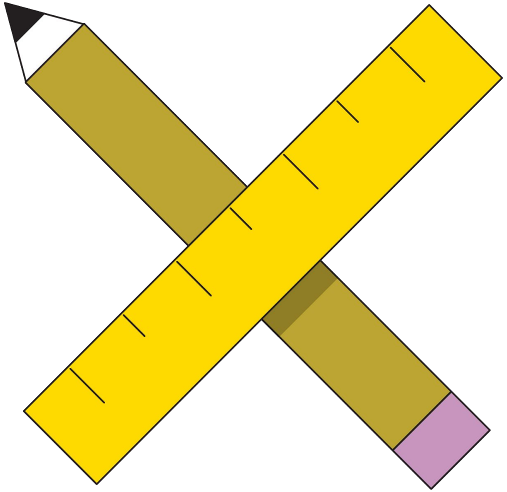
What are their differences and similarities?

Group Exercise:

8.4 Sorting and Filtering

25 minutes

Use sorting and filtering methods to explore the attributes of a data set and answer stakeholder questions.



Exploratory Data Analysis With Pandas

Wrapping Up

Recap

In today's class, we...

- Used Pandas to read in a data set.
- Used DataFrame attributes and methods to investigate a data set's integrity.
- Applied filters and sorting to DataFrames.

Looking Ahead

On your own:

- Work through the Python progress assessment on myGA (due at the end of the unit).
- Share your capstone project ideas with your instructor for review.
- Join someone else's project or invite others to join yours!

Next Class:

Data Visualization With Pandas

Don't Forget: Exit Tickets!