

AI Boot Camp

Introduction to Pandas

Module 4: Day 1



Class Objectives

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

- 1 Describe the makeup of a DataFrame.
- 2 Import a CSV file using Pandas.
- 3 Utilize the heads, tails, and info functions.
- 4 Export a CSV file using Pandas.
- 5 Create DataFrame from lists/dictionaries.
- 6 Rename columns.



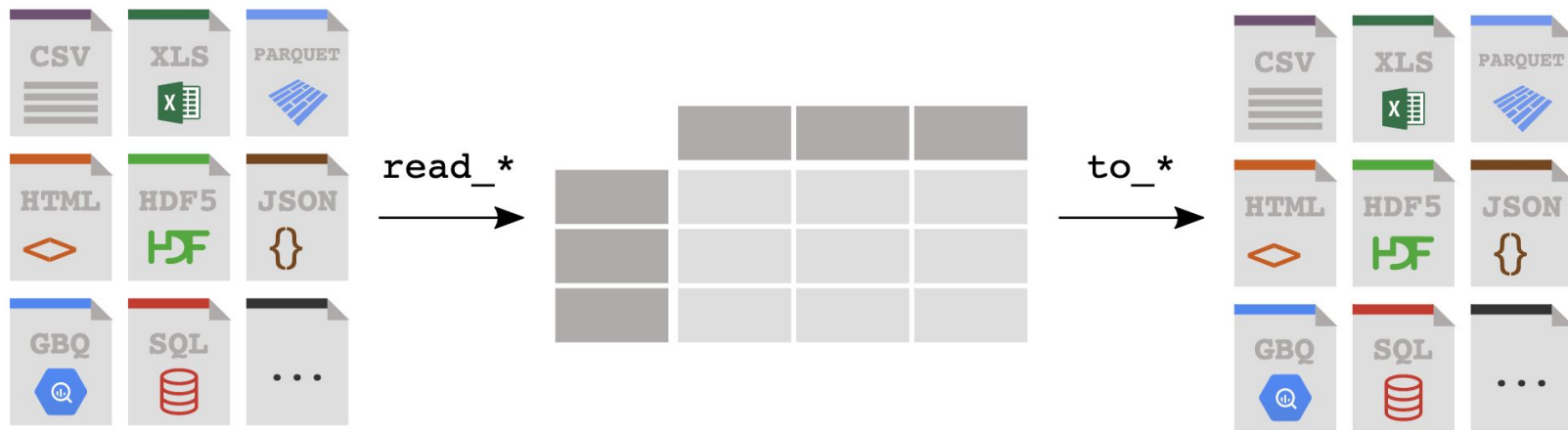
Instructor **Demonstration**

Introduction to Pandas

Pain points when using spreadsheets

- 1 Microsoft Office Excel is expensive.
- 2 Google Spreadsheets is not as advanced as Excel.
- 3 Difficulty in auditing complex cell formulas.
- 4 Slow workbooks for large files.
- 5 Automation and custom function creation is not inherent. Macros and VBA need to be learned.
- 6 Data is static in a spreadsheet, so it is difficult to use real time data.
- 7 What other pain points have you encountered with spreadsheets? Have you experienced any disasters while working with spreadsheets, such as a system crash?

Introduction to Pandas



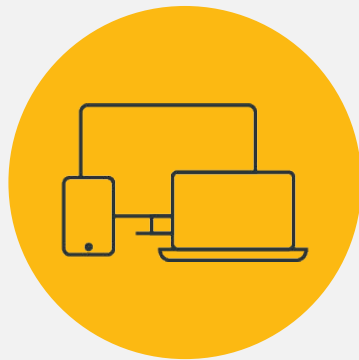
Introduction to Pandas

DataFrame

| | column names | | | | |
|-------|--------------|--|--|--|-----|
| index | | | | | |
| | | | | | row |
| | | | | | |
| | | | | | |
| | | | | | |
| | column | | | | |

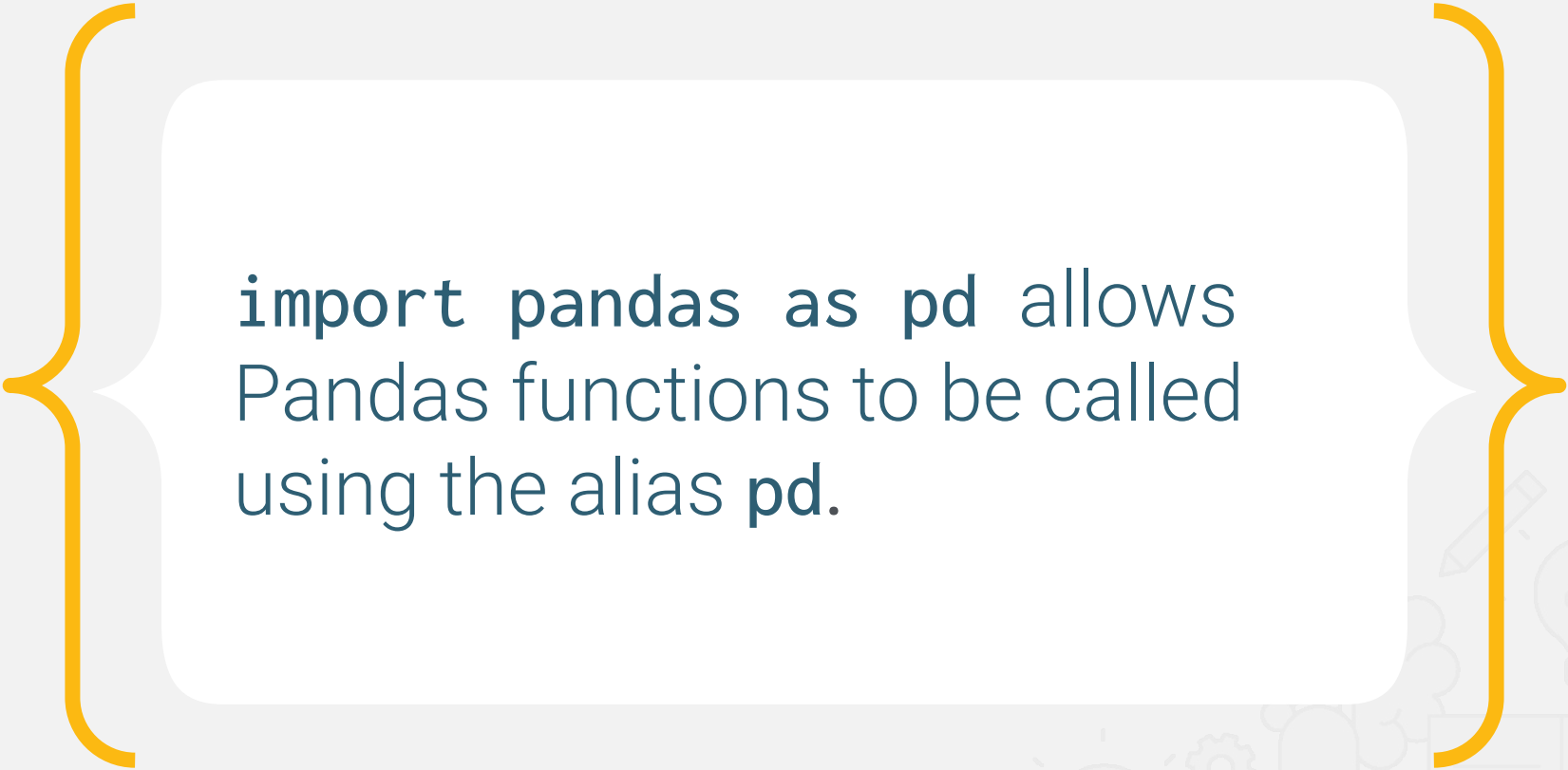
Series

| index | |
|-------|--|
| | |
| | |
| | |
| | |
| | |




Instructor **Demonstration**

Creating Series and DataFrames



`import pandas as pd` allows
Pandas functions to be called
using the alias `pd`.





Creating a Series

To create a Series, simply use the `pd.Series()` function and place a list within the parentheses as follows:

```
```python

We can create a Pandas Series
from a raw list

data_series = pd.Series(["UCLA",
"UC Berkeley", "UC Irvine",
"University of Central Florida",
"Rutgers University"])

data_series

```
```

```
0          UCLA
1      UC Berkeley
2      UC Irvine
3  University of Central Florida
4      Rutgers University
dtype: object
```



Creating a DataFrame

- There are different ways to create DataFrames from scratch. One way 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, as the following example shows:

```
```python

Convert a list of dictionaries into a DataFrame

states_dicts = [{"STATE": "New Jersey", "ABBREVIATION": "NJ"},
 {"STATE": "New York", "ABBREVIATION": "NY"}]

states_df = pd.DataFrame(states_dicts)

states_df

```
```

| | Abbreviation | State |
|---|--------------|------------|
| 0 | NJ | New Jersey |
| 1 | NY | New York |



Creating a DataFrame

- Another way to use the `pd.DataFrame()` function is to provide it with 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 as the following code shows:

```
# Convert a single dictionary
containing lists into a DataFrame

pharaoh_df = pd.DataFrame(

    {"Dynasty": ["Early Dynastic
Period", "Old Kingdom"],

    "Pharaoh": ["Thinis", "Memphis"]}

)

pharaoh_df
```

| | Dynasty | Pharaoh |
|---|-----------------------|---------|
| 0 | Early Dynastic Period | Thinis |
| 1 | Old Kingdom | Memphis |



Activity:

DataFrame Shop

Create a DataFrame for a frame shop. The DataFrame should contain three columns, "Frame", "Price", and "Sales", and have five rows of data stored within it.

Using an alternative method, create a DataFrame for an art gallery. The DataFrame should contain three columns, "Painting", "Price", and "Popularity", and have four rows of data stored within it.

Suggested Time:
10 Minutes





Time's up!
Let's review



Activity:

DataFrame Shop

```
# Create a DataFrame from a list of dictionaries.
```

```
painting_df = pd.DataFrame([
    {"Painting": "Mona Lisa (Knockoff)", "Price": 25,
     "Popularity": "Very Popular"},
    {"Painting": "Van Gogh (Knockoff)", "Price": 20,
     "Popularity": "Popular"},
    {"Painting": "Starving Artist", "Price": 10,
     "Popularity": "Average"},
    {"Painting": "Toddler Drawing", "Price": 1, "Popularity":
     "Not Popular"}
])
painting_df
```

| | Painting | Price | Popularity |
|---|----------------------|-------|--------------|
| 0 | Mona Lisa (Knockoff) | 25 | Very Popular |
| 1 | Van Gogh (Knockoff) | 20 | Popular |
| 2 | Starving Artist | 10 | Average |
| 3 | Toddler Drawing | 1 | Not Popular |

Suggested Time:
10 Minutes



Activity:

DataFrame Shop

```
# Create a DataFrame of frames using a dictionary of lists.
```

```
frame_df = pd.DataFrame({  
    "Frame": ["Ornate", "Classical", "Modern", "Wood",  
    "Cardboard"],  
    "Price": [15.00, 12.50, 10.00, 5.00, 1.00],  
    "Sales": [100, 200, 150, 300, "N/A"]  
})  
frame_df
```

| | Frame | Price | Sales |
|---|-----------|-------|-------|
| 0 | Ornate | 15.0 | 100 |
| 1 | Classical | 12.5 | 200 |
| 2 | Modern | 10.0 | 150 |
| 3 | Wood | 5.0 | 300 |
| 4 | Cardboard | 1.0 | N/A |

Suggested Time:

10 Minutes



Instructor **Demonstration**

Reading and Writing CSVs



CSV files: Comma separated value files.



CSV file use cases



Importing and exporting data



Data analysis



Data backups



Data transfer

Different Encoding Types for CSV files



UTF-8 (Unicode Transformation Format 8-bit):

This is the most common type of encoding, and can represent most characters and symbols.



UTF-16 (Unicode Transformation Format 16-bit):

This encoding type can represent a larger range of characters than UTF-8 but also requires more storage space.



ASCII (American Standard Code for Information Interchange):

ASCII encoding represents English characters and some symbols, but does not support other languages.



ISO-8859-1:

This encoding standard is used for Western European languages, and can also represent characters from the Latin alphabet.

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)
```



Instructor **Demonstration**

Column Manipulation

The Importance of Reordering Columns



Personal preference



Data analysis



Data visualization

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  
people_df.columns
```

```
Index(['last_name', 'company_name ', 'city', 'email'], 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 = people_df[["last_name", "company_name", "city", "email"]]  
organized_df.head()
```

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={"last_name": "Last Name", "company_name": "Company",  
"city": "City", "email": "Email"})  
renamed_df.head()
```

| | Last Name | Company | City | Email |
|---|------------|-------------------------|--------------------------------|--|
| 0 | Tomkiewicz | Alan D Rosenberg Cpa Pc | St. Stephens Ward | atomkiewicz@hotmail.com |
| 1 | Zigomalas | Cap Gemini America | Abbey Ward | evan.zigomalas@gmail.com |
| 2 | Andrade | Elliott, John W Esq | East Southbourne and Tuckton W | france.andrade@hotmail.com |
| 3 | Mcwalters | McMahon, Ben L | Hawerby cum Beesby | ulysses@hotmail.com |
| 4 | Veness | Champagne Room | Greets Green and Lyng Ward | tyisha.veness@hotmail.com |



Activity:

Import, Organize, Export

In this activity, you will take a premade DataFrame of donors to a non-profit organization and reorganize it so that it improves readability and analysis.

Suggested Time:

15 Minutes





Time's up!
Let's review



Activity:

Import, Organize, Export

Convert columns into the more readable versions using the following code:

```
# Rename columns for readability

donors_df_renamed =
donors_df.rename(columns={"donorName": "Donor",

"employerName": "Employer",

"zipcode": "Zip Code",

"usd": "Donation Amount"

})

donors_df_renamed.head()
```

Using double brackets, the new columns are reorganized and placed into another new variable.

```
# Organize the columns so that only the donor name,
amount, employer, and city are
displayed. The name and donation amount
should be first

donors_df_organized = donors_df_renamed[['Donor',
                                          'Donation
                                          Amount',
                                          'Employer',
                                          'City'
                                          ]]

donors_df_organized.head()
```

| | Donor | Donation Amount | Employer | City |
|---|----------------------------------|-----------------|------------------------------|------------------------|
| 0 | CAREY, JAMES | 500 | UNEMPLOYED | HOCKESSIN |
| 1 | OBICI, SILVANA | 250 | STONY BROOK | PORT JEFFERSON STATION |
| 2 | MAISLIN, KAREN | 250 | RETIRED | WILLIAMSVILLE |
| 3 | MCCLELLAND, CARTER AND STEPHANIE | 1000 | UNION SQUARE ADVISORS | NEW YORK |
| 4 | MCCLUSKEY, MARTHA | 250 | STATE UNIVERSITY OF NEW YORK | BUFFALO |

Suggested Time:
10 Minutes



Break

15 mins

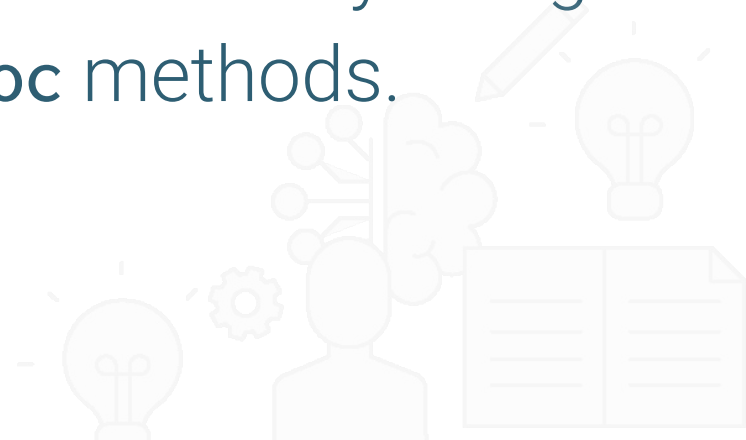


Instructor **Demonstration**

Indexing (`loc` and `iloc`)



Programmers can easily collect specific rows and columns of data from a DataFrame by using the **loc** and **iloc** methods.





loc returns data based on column names and row indexes.

Instead of using labels, **iloc** uses integer-based indexing for selection by position.



Exploring Data with loc and iloc

A new index can be specified using the `set_index` function. Note in the example below that "STREET NAME" becomes the index column after the `set_index` function is used:

```
# Set new index to STREET NAME
df = original_df.set_index("STREET NAME")
df.head()
```

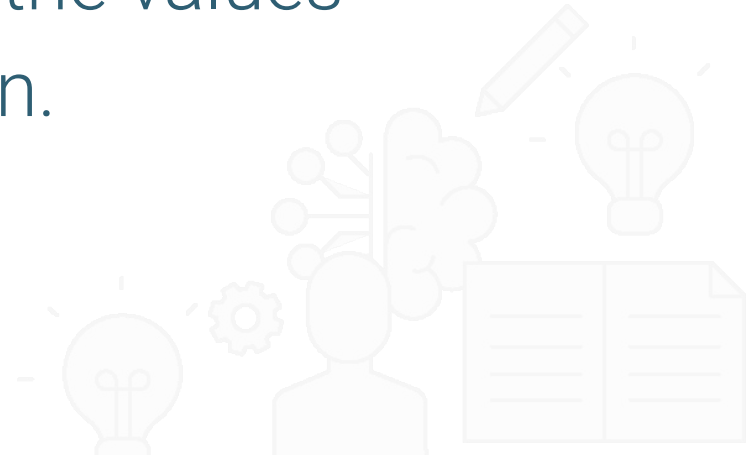
| STREET NAME | STREET NAME ID | STREET FULL NAME | POSTAL COMMUNITY | MUNICIPAL COMMUNITY |
|----------------|----------------|-------------------|------------------|---------------------|
| PRIVATE STREET | 1400342 | PRIVATE STREET | BATON ROUGE | BATON ROUGE |
| 4TH | 1 | Leanora Cashell | BATON ROUGE | BATON ROUGE |
| 11TH | 10 | Arley Medford | BATON ROUGE | BATON ROUGE |
| ADDINGTON | 100 | Rafaello Crawshaw | BATON ROUGE | BATON ROUGE |
| CHALFONT | 1000 | Karalee Hallaways | BATON ROUGE | PARISH |

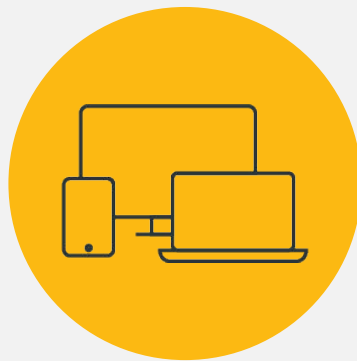
Exploring Data with `loc` and `iloc`

- Both `loc` and `iloc` use brackets that contain the desired rows, followed by a comma and the desired columns.
- For example, `loc["ADDINGTON", "STREET FULL NAME"]` or `iloc[3,1]`



loc and **iloc** can be used to conditionally filter rows of data based on the values within a column.





Instructor **Demonstration**

Indexing



Activity:

Good Movies

In this activity you will find the less popular movies (with fewer than 20k votes) that are highly rated (at a 7 or above) using `loc` and `iloc` on data from IMDB.

Suggested Time:

20 Minutes





Time's up!
Let's review



Activity:

Good Movies

```
# We only like good movies, so find those that scored over 7, and ignore the norm rating
good_movies_df = movie_file_df.loc[movie_file_df["IMDB"] > 7, [
    "FILM", "IMDB", "IMDB_user_vote_count"]]
good_movies_df.head()
```

| | FILM | IMDB | IMDB_user_vote_count |
|---|--------------------------------|------|----------------------|
| 0 | Avengers: Age of Ultron (2015) | 7.8 | 271107 |
| 1 | Cinderella (2015) | 7.1 | 65709 |
| 2 | Ant-Man (2015) | 7.8 | 103660 |
| 5 | The Water Diviner (2015) | 7.2 | 39373 |
| 8 | Shaun the Sheep Movie (2015) | 7.4 | 12227 |

```
# Find less popular movies--i.e., those with fewer than 20K votes
unknown_movies_df = good_movies_df.loc[good_movies_df["IMDB_user_vote_count"] < 20000, [
    "FILM", "IMDB", "IMDB_user_vote_count"]]
unknown_movies_df.head()
```

| | FILM | IMDB | IMDB_user_vote_count |
|----|-----------------------------------|------|----------------------|
| 8 | Shaun the Sheep Movie (2015) | 7.4 | 12227 |
| 9 | Love & Mercy (2015) | 7.8 | 5367 |
| 10 | Far From The Madding Crowd (2015) | 7.2 | 12129 |
| 20 | McFarland, USA (2015) | 7.5 | 13769 |
| 29 | The End of the Tour (2015) | 7.9 | 1320 |

Suggested Time:
20 Minutes



Activity:

Comic Books

You will now take a large CSV file containing comic books, read it into Jupyter Notebook using Pandas, clean up the columns, and then write their modified DataFrame to a new CSV file.

Suggested Time:
15 Minutes





Time's up!
Let's review



Recap

Today, you covered:

- 1 The basics of Pandas.
- 2 Creating DataFrames.
- 3 Manipulating columns within a DataFrame.
- 4 How to import and export CSV files.
- 5 Selecting data from DataFrames with the use of the `loc` and `iloc` functions.



Next

In the next lesson, we will start exploring data with Pandas, sorting values in a DataFrame, viewing summary statistics of a DataFrame, and filtering data within a DataFrame.



Questions?





The End