Exploring Pandas

Data Boot Camp

Lesson 4.2

Class Objectives

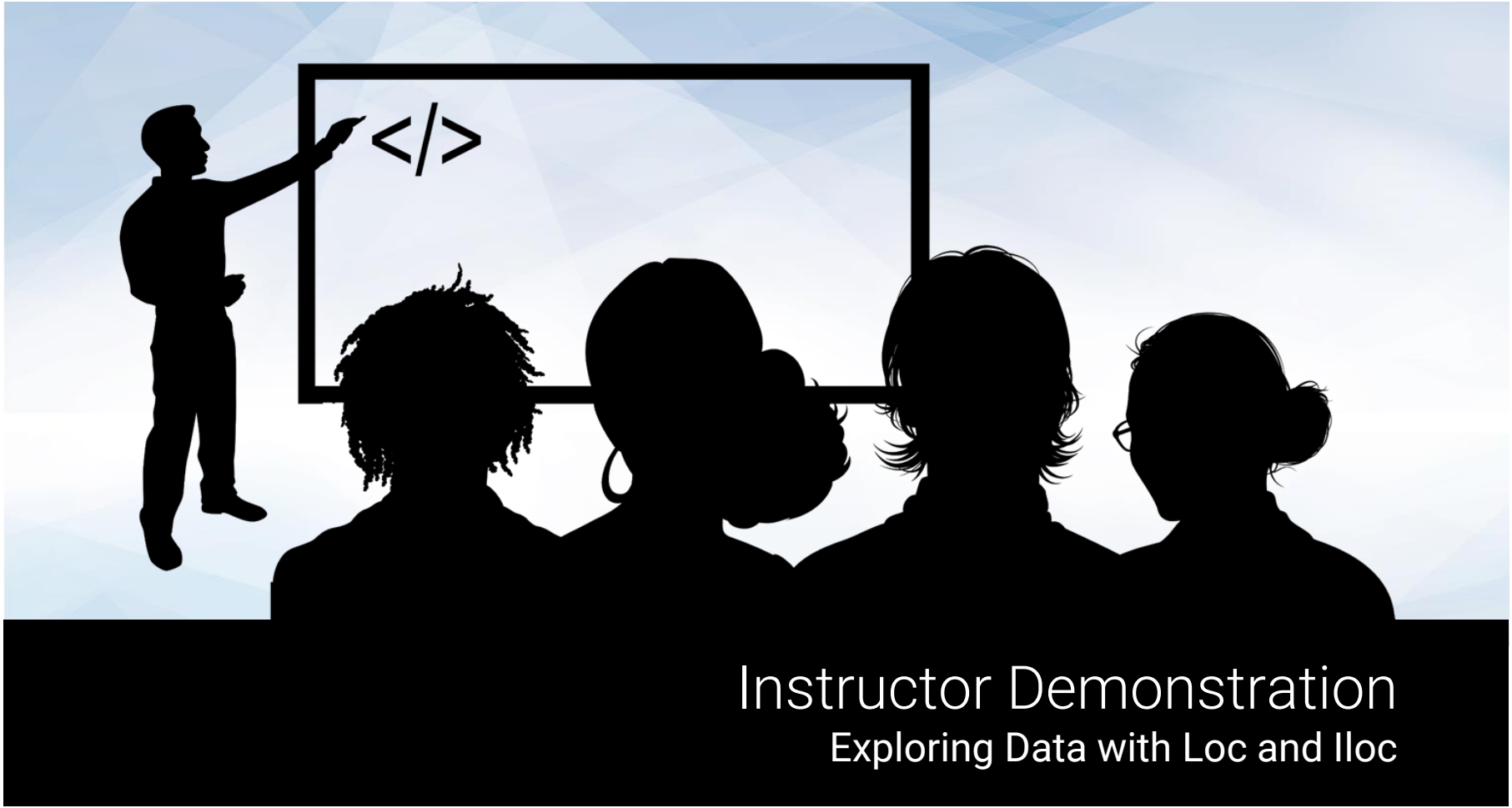
By the end of today's class you will be able to:

Understand how to navigate through DataFrames using Loc and Iloc.

Understand how to filter and slice Pandas DataFrames.

Understand how to create and access Pandas GroupBy objects.

Understand how to sort DataFrames.



Instructor Demonstration

Exploring Data with Loc and Iloc

Programers can easily collect specific rows/columns of data from a DataFrame using the `loc()` and `iloc()` methods.

Exploring Data With Loc and Iloc

- The `loc()` method returns data based upon an index of labels/strings.
- `loc()` is limited to string types and cannot be used on a numerical index. As an alternative solution you can use the `df.set_index()` function passing in the desired column header for the index.
- On the other hand the `iloc()` method instead of using labels, uses integer based indexing for selection by position.

```
In [4]: # Set new index to last_name
        df = original_df.set_index("last_name")
        df.head()
```

Out[4]:

	id	first_name	Phone Number	Time zone
last_name				
Richardson	1	Peter	7-(789)867-9023	Europe/Moscow
Berry	2	Janice	86-(614)973-1727	Asia/Harbin
Hudson	3	Andrea	86-(918)527-6371	Asia/Shanghai
Mcdonald	4	Arthur	420-(553)779-7783	Europe/Prague
Morales	5	Kathy	351-(720)541-2124	Europe/Lisbon

One of the most powerful aspects of Pandas

Exploring Data With Loc and Iloc

- Both `loc()` and `iloc()` methods use brackets that contain the desired rows, followed by a comma and then the columns desired.
- For example: `loc['Berry', 'Phone Number']` or `iloc[1,2]`

In [5]:
```
# Grab the data contained within the "Berry" row and the "Phone Number" column
berry_phone = df.loc["Berry", "Phone Number"]
print("Using Loc: " + berry_phone)

also_berry_phone = df.iloc[1, 2]
print("Using Iloc: " + also_berry_phone)
```

```
Using Loc: 86-(614)973-1727
Using Iloc: 86-(614)973-1727
```

Exploring Data With Loc and Iloc

- Both methods allow us to select a range of columns and rows by providing a list.
- We can also use a colon to tell Pandas to look for a range.

In [6]:
```
# Grab the first five rows of data and the columns from "id" to "Phone Number"
# The problem with using "last_name" as the index is that the values are not unique so duplicates
are returned
# If there are duplicates and loc[] is being used, Pandas will return an error
richardson_to_morales = df.loc[["Richardson", "Berry", "Hudson",
                                "Mcdonald", "Morales"], ["id", "first_name", "Phone Number"]]
print(richardson_to_morales)

print()

# Using iloc[] will not find duplicates since a numeric index is always unique
also_richardson_to_morales = df.iloc[0:4, 0:3]
print(also_richardson_to_morales)
```

```
            id first_name       Phone Number
last_name
Richardson   1      Peter    7-(789)867-9023
Richardson  25     Donald   62-(259)282-5871
Berry        2     Janice   86-(614)973-1727
Hudson       3     Andrea   86-(918)527-6371
Hudson       8    Frances   57-(752)864-4744
Hudson      90      Norma  351-(551)598-1822
Mcdonald     4     Arthur  420-(553)779-7783
Morales      5      Kathy  351-(720)541-2124

            id first_name       Phone Number
last_name
Richardson   1      Peter    7-(789)867-9023
Berry        2     Janice   86-(614)973-1727
Hudson       3     Andrea   86-(918)527-6371
Mcdonald     4     Arthur  420-(553)779-7783
```

Exploring Data With Loc and Iloc

By passing in a colon by itself, `loc()` and `iloc()` will select all rows or columns depending on where it is placed in relation to the comma.

In [7]:
```
# The following will select all rows for columns `first_name` and `Phone Number`
df.loc[:, ["first_name", "Phone Number"]].head()
```

Out[7]:

	first_name	Phone Number
last_name		
Richardson	Peter	7-(789)867-9023
Berry	Janice	86-(614)973-1727
Hudson	Andrea	86-(918)527-6371
Mcdonald	Arthur	420-(553)779-7783
Morales	Kathy	351-(720)541-2124

`loc()` and `iloc()` can be used to conditionally filter rows of data based upon the values within a column.

Exploring Data With Loc and Iloc

- Instead of passing a list of indices, we can use a logic statement!
- In case of multiple conditions that should be checked for, & and | may be added into logic test as representations of and and or.

In [9]:
```
# Loc and Iloc also allow for conditional statments to filter rows of data
# using Loc on the logic test above only returns rows where the result is True
only_billys = df.loc[df["first_name"] == "Billy", :]
print(only_billys)

print()

# Multiple conditions can be set to narrow down or widen the filter
only_billy_and_peter = df.loc[(df["first_name"] == "Billy") | (
    df["first_name"] == "Peter"), :]
print(only_billy_and_peter)
```

```
           id first_name      Phone Number       Time zone
last_name
Clark      20      Billy  62-(213)345-2549   Asia/Makassar
Andrews    23      Billy  86-(859)746-5367  Asia/Chongqing
Price      59      Billy  86-(878)547-7739   Asia/Shanghai

            id first_name      Phone Number       Time zone
last_name
Richardson   1      Peter   7-(789)867-9023   Europe/Moscow
Clark       20      Billy  62-(213)345-2549   Asia/Makassar
Andrews     23      Billy  86-(859)746-5367  Asia/Chongqing
Price       59      Billy  86-(878)547-7739   Asia/Shanghai
```

Activity: Good Movies

In this activity, you will create an application that looks through IMDB data in order to find only the best movies out there.

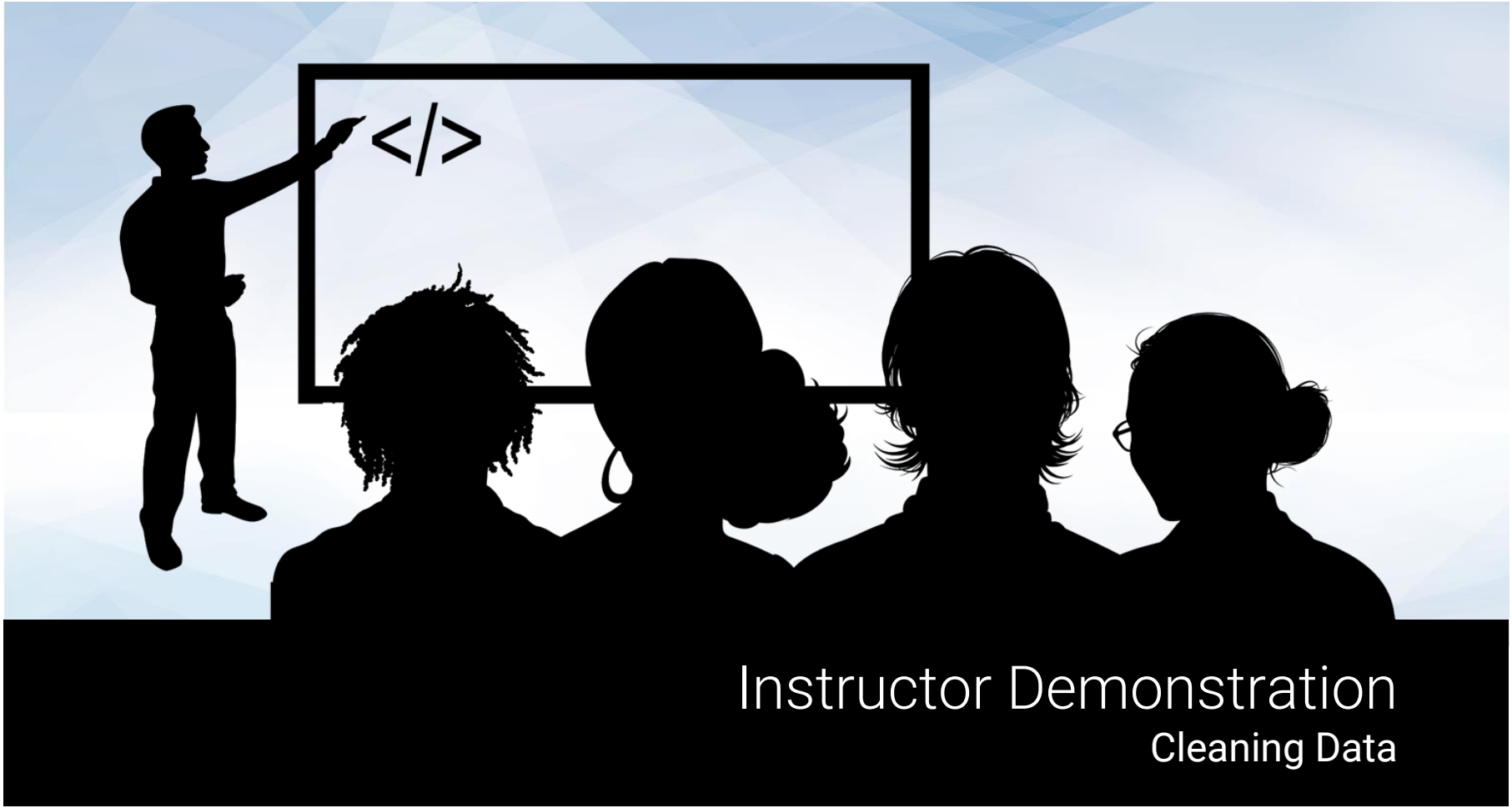
Suggested Time:
20 Minutes

Activity: Good Movies

Instructions:

- Use Pandas to load and display the CSV provided in Resources.
  List all the columns in the data set.
- We're only interested in IMDb data, so create a new table that takes the Film and all the columns relating to IMDB.
- Filter out only the good movies—i.e., any film with an IMDb score greater than or equal to seven and remove the norm ratings.
- Find less popular movies that you may not have heard about - i.e., anything with under 20K votes.
- Finally, export this file to a spreadsheet, excluding the index, so we can keep track of our future watchlist.

Time’s Up! Let’s Review.



Instructor Demonstration

Cleaning Data

When dealing with massive datasets it is almost inevitable that duplicate rows, inconsistent spelling and missing values will crop up.

Cleaning Data

`del <DataFrame>[<columns>]`

In [4]:
```
# Preview of the DataFrame
# Note that FIELD8 is likely a meaningless column
df.head()
```

Out[4]:

	LastName	FirstName	Employer	City	State	Zip	Amount	FIELD8
0	Aaron	Eugene	State Department	Dulles	VA	20189	500.0	NaN
1	Abadi	Barbara	Abadi & Co.	New York	NY	10021	200.0	NaN
2	Adamany	Anthony	Retired	Rockford	IL	61103	500.0	NaN
3	Adams	Lorraine	Self	New York	NY	10026	200.0	NaN
4	Adams	Marion	None	Exeter	NH	03833	100.0	NaN

In [5]:
```
# Delete extraneous column
del df['FIELD8']
df.head()
```

Out[5]:

	LastName	FirstName	Employer	City	State	Zip	Amount
0	Aaron	Eugene	State Department	Dulles	VA	20189	500.0
1	Abadi	Barbara	Abadi & Co.	New York	NY	10021	200.0
2	Adamany	Anthony	Retired	Rockford	IL	61103	500.0
3	Adams	Lorraine	Self	New York	NY	10026	200.0
4	Adams	Marion	None	Exeter	NH	03833	100.0

Cleaning Data

`count()`

`<DataFrame>.dropna(how='any')`

In [6]:
```
# Identify incomplete rows
df.count()
```

Out[6]:
```
LastName     1776
FirstName    1776
Employer     1743
City         1776
State        1776
Zip          1776
Amount       1776
dtype: int64
```

In [7]:
```
# Drop all rows with missing information
df = df.dropna(how='any')
```

In [8]:
```
# Verify dropped rows
df.count()
```

Out[8]:
```
LastName     1743
FirstName    1743
Employer     1743
City         1743
State        1743
Zip          1743
Amount       1743
dtype: int64
```

Cleaning Data

`value_counts()`

`replace()`

In [12]:
```
# Display an overview of the Employers column
df['Employer'].value_counts()
```

Out[12]:
```
None             249
Self             241
Retired          126
Self Employed     39
Self-Employed     34
```

In [13]:
```
# Clean up Employer category. Replace 'Self Employed' and 'Self' with 'Self-Employed'
df['Employer'] = df['Employer'].replace(
    {'Self Employed': 'Self-Employed', 'Self': 'Self-Employed'})
```

In [14]:
```
# Verify clean-up.
df['Employer'].value_counts()
```

Out[14]:
```
Self-Employed    314
None             249
Retired          126
Google             6
```

Activity: Sydney Crime

In this activity, you will take a crime dataset from Sydney and do your best to clean it up so that the DataFrame is consistent and no rows with missing data are present.

Suggested Time:
20 Minutes

Activity: Sydney Crime

Instructions:

- Read in the csv using Pandas and print out the DataFrame that is returned.
- Get a count of rows within the DataFrame in order to determine if there are any null values.
- Drop the rows which contain null values.
- Search through the ‘Offence type’ column and replace any similar values with one consistent value.
- Create a couple DataFrames that look into one Region only and print them to the screen.

Time’s Up! Let’s Review.

Activity:

Pandas Recap and Data Types

In this activity, we will recap what has been covered in Pandas up to this point.

Suggested Time:
15 Minutes

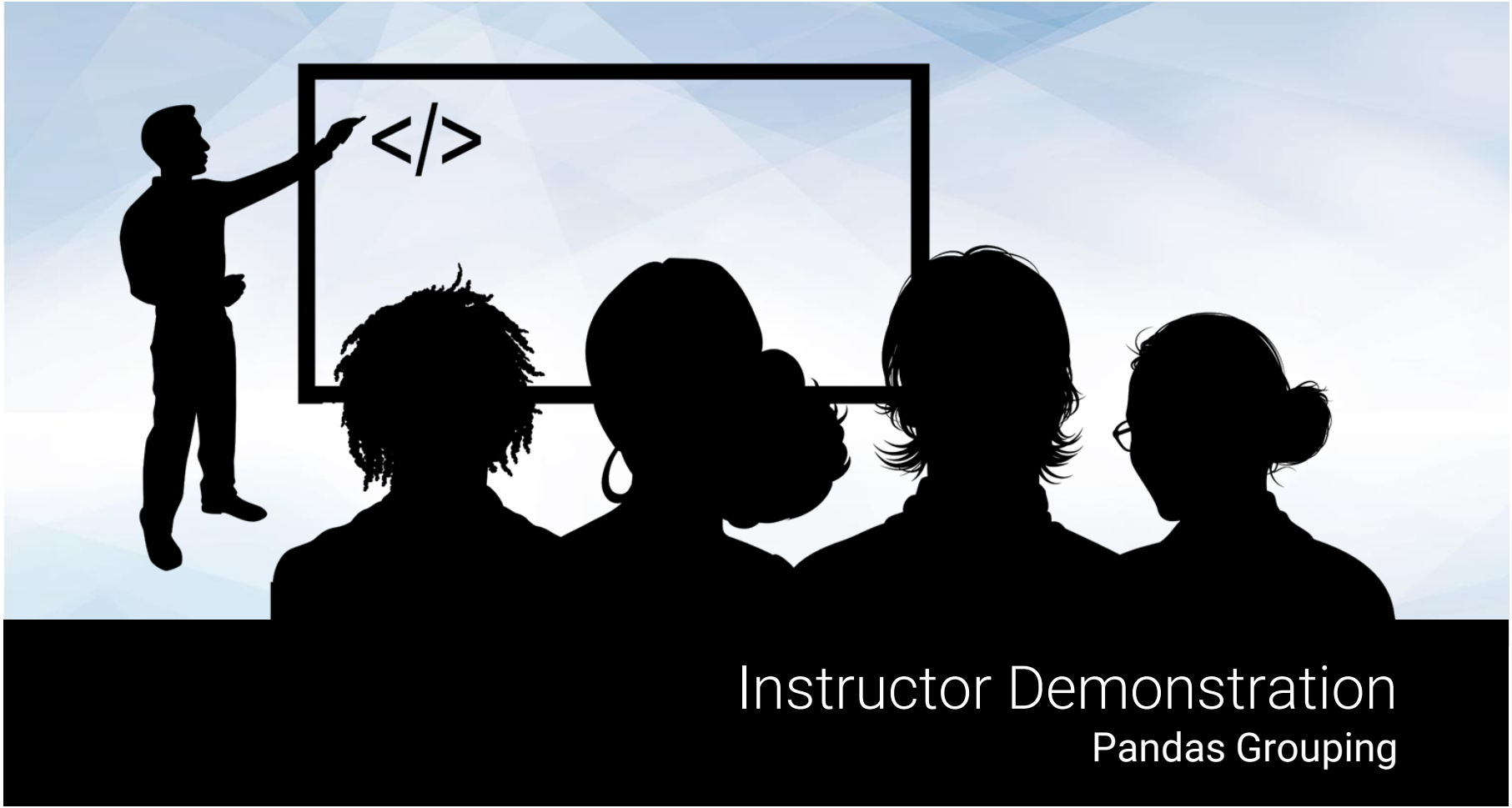
Activity: Pandas Recap and Data Types

Instructions	Hints
Open up ‘PandasRecap.ipynb’ under the ‘unsolved’ folder in your Jupyter Notebook.	A list of DataFrame's data types can be seen by accessing its `dtypes` property.
Go through the cells following the notes.	In order to change a non-numeric column to a numeric column, use the `df.astype(<datatype>)` method and pass in the desired datatype as the parameter. 

Countdown timer

15:00

(with alarm)



Instructor Demonstration

Pandas Grouping

`.groupby()` is a simpler method to filter data.

Pandas Grouping

In order to split the DataFrame into multiple groups and group by state the `df.groupby([<Columns>])` is used.

The `.groupby()` method returns a GroupBy object that can only be access by using a data function on it.

In [9]:
```
# Using GroupBy in order to separate the data into fields according to "state" values
grouped_usa_df = usa_ufo_df.groupby(['state'])

# The object returned is a "GroupBy" object and cannot be viewed normally...
print(grouped_usa_df)

# In order to be visualized, a data function must be used...
grouped_usa_df.count().head(10)
```

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x10cde6278>
```

Out[9]:

	datetime	city	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
state										
ak	311	311	311	311	311	311	311	311	311	311
al	629	629	629	629	629	629	629	629	629	629
ar	578	578	578	578	578	578	578	578	578	578
az	2362	2362	2362	2362	2362	2362	2362	2362	2362	2362
ca	8683	8683	8683	8683	8683	8683	8683	8683	8683	8683
co	1385	1385	1385	1385	1385	1385	1385	1385	1385	1385
ct	865	865	865	865	865	865	865	865	865	865
dc	7	7	7	7	7	7	7	7	7	7
de	165	165	165	165	165	165	165	165	165	165
fl	3754	3754	3754	3754	3754	3754	3754	3754	3754	3754

Pandas Grouping

The `pd.DataFrame()` method makes it possible to create new DataFrames using solely GroupBy data.

A DataFrame can also be created by selecting a single series from a GroupBy object and passing it in as the values for a specified column.

In [11]:
```
# Since "duration (seconds)" was converted to a numeric time, it can now be summed up per state
state_duration = grouped_usa_df["duration (seconds)"].sum()
state_duration.head()
```

Out[11]:
```
state
ak     1455863.00
al      900453.50
ar    66986144.50
az    15453494.60
ca    24865571.47
Name: duration (seconds), dtype: float64
```

In [12]:
```
# Creating a new DataFrame using both duration and count
state_summary_df = pd.DataFrame({"Number of Sightings": state_counts,
                                 "Total Visit Time": state_duration})
state_summary_df.head()
```

Out[12]:

	Number of Sightings	Total Visit Time
ak	311	1455863.00
al	629	900453.50
ar	578	66986144.50
az	2362	15453494.60
ca	8683	24865571.47

Pandas Grouping

It is possible to perform a `df.groupby()` method on multiple columns as well. This can be done by simply passing two or more column references into the list parameter.

In [13]:
```
# It is also possible to group a DataFrame by multiple columns
# This returns an object with multiple indexes, however, which can be harder to deal with
grouped_international_data = converted_ufo.groupby(['country', 'state'])

grouped_international_data.count().head(20)
```

Out[13]:

		datetime	city	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
country	state									
au	al	1	1	1	1	1	1	1	1	1
	dc	1	1	1	1	1	1	1	1	1
	nt	2	2	2	2	2	2	2	2	2
	oh	1	1	1	1	1	1	1	1	1
	sa	2	2	2	2	2	2	2	2	2
	wa	2	2	2	2	2	2	2	2	2
	yt	1	1	1	1	1	1	1	1	1
ca	ab	284	284	284	284	284	284	284	284	284
	bc	677	677	677	677	677	677	677	677	677
	mb	124	124	124	124	124	124	124	124	124
	nb	86	86	86	86	86	86	86	86	86
	nf	15	15	15	15	15	15	15	15	15
	ns	101	101	101	101	101	101	101	101	101
	nt	13	13	13	13	13	13	13	13	13
	on	1335	1335	1335	1335	1335	1335	1335	1335	1335
	pe	10	10	10	10	10	10	10	10	10
	pq	62	62	62	62	62	62	62	62	62
	qc	124	124	124	124	124	124	124	124	124
	sa	27	27	27	27	27	27	27	27	27
	sk	77	77	77	77	77	77	77	77	77

Pandas Grouping

A new DataFrame can be created from a `GroupBy` object.

In [14]:
```
# Converting a GroupBy object into a DataFrame
international_duration_df = pd.DataFrame(
    grouped_international_data["duration (seconds)"].sum())
international_duration_df.head(10)
```

Out[14]:

		duration (seconds)
country	state	
au	al	900.00
	dc	300.00
	nt	360.00
	oh	180.00
	sa	305.00
	wa	450.00
	yt	30.00
ca	ab	530994.00
	bc	641955.82
	mb	160132.00

Activity: Building a PokeDex

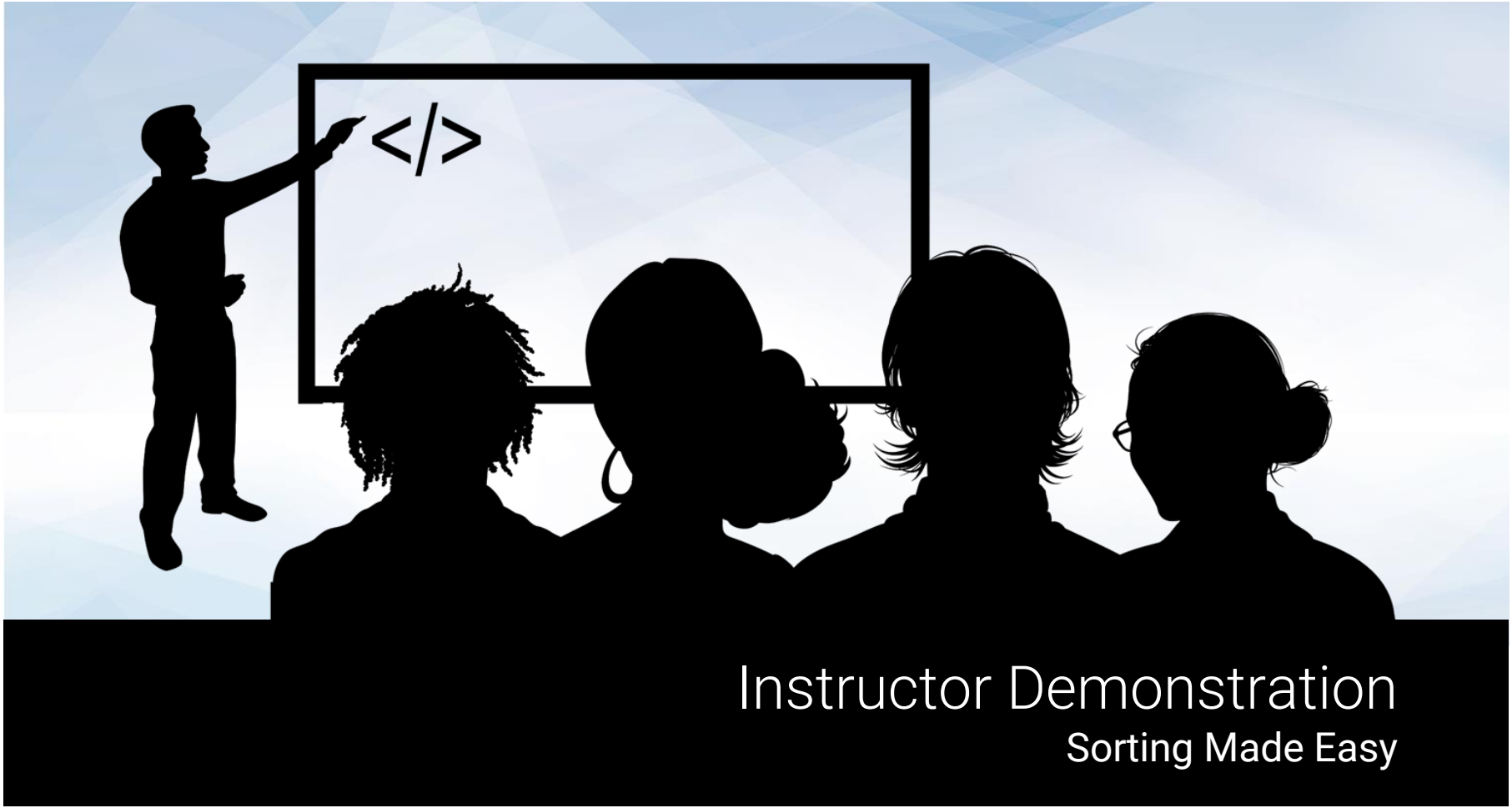
In this activity, you will create a DataFrame that visualises the average stats for each type of Pokemon from the popular video game series.

Suggested Time:
25 Minutes

Activity: Pandas Recap and Data Types

Instructions	Bonus
Read the Pokemon CSV file with Pandas.	Sort the table by strongest type and export the resulting table to a new CSV.
Create a new table by extracting the following columns: ‘Type 1’, ‘HP’, ‘Attack’, ‘Sp. Atk’, ‘Sp. Def’, and ‘Speed’.	
Find the average stats for each type of Pokemon.	
Create a new DataFrame out of the averages.	
Calculate the total power level of each type of Pokemon by summing all of the previous stats together and place the results into a new column.	

Time’s Up! Let’s Review.



Instructor Demonstration

Sorting Made Easy

Sorting Made Easy

In order to sort a DataFrame based upon the values within a column, simply use the `df.sort_values()` method and pass the column name to sort by in as a parameter.

The parameter of 'ascending' is always marked as True by default. This means that the `sort_values()` method will always sort from lowest to highest unless the parameter of `ascending=False` is passed into the `sort_values()` method as well.

In [3]:
```
# Sorting the DataFrame based on "Freedom" column
# Will sort from lowest to highest if no other parameter is passed
freedom_df = happiness_df.sort_values("Freedom")
freedom_df.head()
```

Out[3]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..L
139	Angola	140	3.795	3.951642	3.638358	0.858428	1.104412	0.049869
129	Sudan	130	4.139	4.345747	3.932253	0.659517	1.214009	0.290921
144	Haiti	145	3.603	3.734715	3.471285	0.368610	0.640450	0.277321
153	Burundi	154	2.905	3.074690	2.735310	0.091623	0.629794	0.151611
151	Syria	152	3.462	3.663669	3.260331	0.777153	0.396103	0.500533

In [4]:
```
# To sort from highest to lowest, ascending=False must be passed in
freedom_df = happiness_df.sort_values("Freedom", ascending=False)
freedom_df.head()
```

Out[4]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.4982
0	Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.7966
128	Cambodia	129	4.168	4.278518	4.057483	0.601765	1.006238	0.4297
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.8335
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.7925

Activity: Search for the Worst

In this activity, you will take a dataset composed of soccer player statics and will attempt to determine which players are the worst in the world at their particular position.

Suggested Time:
25 Minutes

Activity: Search for the Worst

Instructions:

- Read in the CSV file provided and print it to the screen.
- Print out a list of all of the values within the ‘Preferred Position’ column.
- Select a value from this list and create a new DataFrame that only includes players who prefer that position.
- Sort the DataFrame based upon a player's skill in that position.
- Reset the index for the DataFrame so that the index is in order.
- Print out the statistics for the worst player in a position to the screen.

Time’s Up! Let’s Review.


the End