Welcome to Class

Sunday, September 29, 2019 8:38 PM

Overview

Today's lesson takes dives deeply into Pandas and covers some of the library's more complex functions - like iloc, loc, and grouping - whilst solidifying the concepts from the last class

Class Objectives

- Understanding how to navigate through DataFrames using Loc and Iloc
- Understanding how to filter and slice Pandas DataFrames
- Understanding how to create and access Pandas GroupBy objects
- Understand how to sort DataFrames

Reading and Writing CSV

Sunday, September 29, 2019 12:38 PM

Open pandas_reading_files.ipynb

Up until this point, the class has had to manually create DataFrames using the **`pd.DataFrame()`** method. There is a far more effective means by which to create large DataFrames; importing CSV files.

- Create a reference to the CSV file's path and pass it in into the **`pd.read_csv()`** method, making certain to store the returned DataFrame within a variable.
    - *In most cases it is not important to use or define the encoding of the base CSV file but if the encoding is different than UTF-8, then it may become necessary so that the CSV is translated correctly.*

In [1]:
```
# Dependencies
import pandas as pd
```

In [2]:
```
# Store filepath in a variable
file_one = "C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.1\\Data\\DataOne.csv"
```

In [3]:
```
# 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")
```

- To write to a CSV file, we can use the **`df.to_csv()`** method, passing the path to the desired output file.
    - By using the `index` and `header` parameters, programmers can also manipulate whether they would like the index or header for the table to be passed as well.

In [8]:
```
# Export file as a CSV, without the Pandas index, but with the header
file_one_df.to_csv("C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.1\\Output\\fileOne.csv", index=False, header=True)
```

Exercise: GoodReads - Part 1**

Overview

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

Files

GoodReads.ipynb
Books.csv

Instructions

- Read in the Books CSV
- Remove unnecessary columns from the Data frame, only these columns should remain
    - Isbn
    - Originial_publication_year
    - Original_title
    - Authors
    - Ratings_1
    - Ratings_2
    - Ratings_3
    - Ratings_4
    - Ratings_5
- Rename the following columns
    - ISBN
    - Publication Year
    - Original Title
    - Authors
    - One Star Reviews
    - Two Star Reviews
    - Three Star Reviews
    - Four Star Reviews
    - Five Star Reviews
- Write the data frame into a new csv file

Hints

The base CSV file uses UTF-8 encoding. Trying to read in the file using some other kind of encoding could lead to strange characters appearing within the dataset

Review: GoodReads - Part 1

Keys

- There are a lot of columns that are being modified within this code, so it is essential to make sure that all references are made accurately so as to avoid any potential errors.
- The initial CSV file is encoded using UTF-8 and should be read using this encoding as well so that ensure there are no strange characters hidden within the dataset

In [1]:
```
# Import Dependencies
import pandas as pd
```

In [2]:
```
# Make a reference to the books.csv file path
csv_path = "C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.1\\Data\\books.csv"

# Import the books.csv file as a DataFrame
books_df = pd.read_csv(csv_path, encoding="utf-8")
books_df.head()
```

Out[2]:

	id	book_id	best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	...	ratings_count	work_
0	1	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0	The Hunger Games	...	4780653	
1	2	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	...	4602479	
2	3	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	2005.0	Twilight	...	3866839	
3	4	2657	2657	3275794	487	61120081	9.780061e+12	Harper Lee	1960.0	To Kill a Mockingbird	...	3198671	
4	5	4671	4671	245494	1356	743273567	9.780743e+12	F. Scott Fitzgerald	1925.0	The Great Gatsby	...	2683664	

5 rows × 23 columns

- Removing unnecessary columns from the Data frame

In [3]:
```
# Remove unecessary columns from the DataFrame and save the new DataFrame
# Only keep: "isbn", "original_publication_year", "original_title", "authors",
# "ratings_1", "ratings_2", "ratings_3", "ratings_4", "ratings_5"
reduced_df = books_df[["isbn", "original_publication_year", "original_title", "authors",
                       "ratings_1", "ratings_2", "ratings_3", "ratings_4", "ratings_5"]]
reduced_df.head()
```

Out[3]:

	isbn	original_publication_year	original_title	authors	ratings_1	ratings_2	ratings_3	ratings_4	ratings_5
0	439023483	2008.0	The Hunger Games	Suzanne Collins	66715	127936	560092	1481305	2706317
1	439554934	1997.0	Harry Potter and the Philosopher's Stone	J.K. Rowling, Mary GrandPré	75504	101676	455024	1156318	3011543
2	316015849	2005.0	Twilight	Stephenie Meyer	456191	436802	793319	875073	1355439
3	61120081	1960.0	To Kill a Mockingbird	Harper Lee	60427	117415	446835	1001952	1714267
4	743273567	1925.0	The Great Gatsby	F. Scott Fitzgerald	86236	197621	606158	936012	947718

- Rename the following columns

In [4]:
```
# Rename the headers to be more explanatory
renamed_df = reduced_df.rename(columns={"isbn": "ISBN",
                                        "original_title": "Original Title",
                                        "original_publication_year": "Publication Year",
                                        "authors": "Authors",
                                        "ratings_1": "One Star Reviews",
                                        "ratings_2": "Two Star Reviews",
                                        "ratings_3": "Three Star Reviews",
                                        "ratings_4": "Four Star Reviews",
                                        "ratings_5": "Five Star Reviews", })
renamed_df.head()
```

Out[4]:

	ISBN	Publication Year	Original Title	Authors	One Star Reviews	Two Star Reviews	Three Star Reviews	Four Star Reviews	Five Star Reviews
0	439023483	2008.0	The Hunger Games	Suzanne Collins	66715	127936	560092	1481305	2706317
1	439554934	1997.0	Harry Potter and the Philosopher's Stone	J.K. Rowling, Mary GrandPré	75504	101676	455024	1156318	3011543
2	316015849	2005.0	Twilight	Stephenie Meyer	456191	436802	793319	875073	1355439
3	61120081	1960.0	To Kill a Mockingbird	Harper Lee	60427	117415	446835	1001952	1714267
4	743273567	1925.0	The Great Gatsby	F. Scott Fitzgerald	86236	197621	606158	936012	947718

- Write CSV

In [5]:

```
# Push the remade DataFrame to a new CSV file
renamed_df.to_csv("C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.1\\Output\\books_clean.csv",
                  encoding="utf-8", index=False, header=True)
```

Exploring Data with Loc and ILoc

Sunday, September 29, 2019 8:39 PM

Open LocAndiLoc.ipynb

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

- The **`loc()`** method allows its users to select data using label based indexes. In other words, it takes in strings as the keys and returns data based upon that

- Using **`loc()`** to search through rows is only really useful when the index of a dataset is a collection of strings. It is almost always useful when selecting data from columns, however, since column headers are exclusively strings. This can be done by using the **`df.set_index()`** function and passing in the desired column header for the index.

```
# Set new index to last_name
df = df_original.set_index("last_name")
df.head()
```

last_name	id	first_name	Phone Number	Time zone
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

- The **`iloc()`** method allows its users to select data, but instead of using labels, ituses integer based indexing for selection by position. Which means it selects data using a numeric index.
- The typical way in which data is called using both **`loc[]`** and **`iloc[]`** is by using a pair of brackets which contain the rows desired, followed by a comma, and then the columns desired.
  - For example: **`loc["Berry","Phone Number"]`** **or** **`iloc[1,2]`**

```
# 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
```

- It is also possible to select a range of data using **`loc[]`** and **`iloc[]`** by placing all of the values within brackets and/or using a colon to tell Pandas to look for a range.
    - For example: **`loc[["Richardson", "Berry", "Hudson", "Mcdonald", "Morales"],["id", "first_name", "Phone Number"]]`**
    - The problem with using "last_name" as the index is that the values are not unique so duplicates are returned

```
# The problem with using "last_name" as the index is that the values are not unique so duplicates are returned
richardson_to_morales = df.loc[["Richardson", "Berry", "Hudson",
                                "Mcdonald", "Morales"], ["id", "first_name", "Phone Number"]]
print(richardson_to_morales)
```

last_name	id	first_name	Phone Number
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

- **`iloc[0:4, 0:3]`**

```
# 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)
```

last_name	id	first_name	Phone Number
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

- 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.
    - For example: **`loc[:, ["first_name", "Phone Number"]`** will select all rows of data but will only return the "first_name" and "Phone Number" columns.

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

last_name	first_name	Phone Number
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

- Both these methods can be used to conditionally filter rows of data based upon the values contained within a column. The way in which this is done is by calling **`loc[]`** or **`iloc[]`** on a DataFrame and passing a logic test in place of the rows section of the call.
    - For example: **`loc[df["id"] >= 10, :]`** will return all rows of data with a value equal to or greater than 10 within the "id" column.
- It is possible to then select which columns to return by simply adding their references into the columns section of the **`loc[]`** or **`iloc[]`** expression.
- If there are multiple conditions that should be checked for, **`&`** and **`|`** may also be added into the logic test as representations of **`and`** and **`or`**. This allows for a great amount of customization.

```python
# Loc and Iloc also allow for conditional statments to filter rows of data
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
```

Break

Sunday, September 29, 2019 8:41 PM

Exercise: Good Movies

Sunday, September 29, 2019 8:39 PM

Overview

Students will now create an application that looks through IMDB data in order to find only the best movies out there using loc [ ] and iloc[ ]

Files

goodMovies_unsolved.ipynb
Movie_scores.csv

Instructions

- Use Pandas to load and display the movie_scores.csv
- 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 7 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.

Review: Good Movies

Sunday, September 29, 2019 8:39 PM

Keys

- Use Pandas to load and display the movie_scores.csv

```
# Dependencie
import pandas as pd
```

```
# Load in file
movie_file = "C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.2\\Data\\movie_scores.csv"
```

```
# Read and display the CSV with Pandas
movie_file_pd = pd.read_csv(movie_file)
movie_file_pd.head()
```

- List all the columns in the data set.

```
# List all the columns in the table
movie_file_pd.columns
```

```
Index(['FILM', 'RottenTomatoes', 'RottenTomatoes_User', 'Metacritic',
       'Metacritic_User', 'IMDB', 'Fandango_Stars', 'Fandango_Ratingvalue',
       'RT_norm', 'RT_user_norm', 'Metacritic_norm', 'Metacritic_user_nom',
       'IMDB_norm', 'RT_norm_round', 'RT_user_norm_round',
       'Metacritic_norm_round', 'Metacritic_user_norm_round',
       'IMDB_norm_round', 'Metacritic_user_vote_count', 'IMDB_user_vote_count',
       'Fandango_votes', 'Fandango_Difference'],
      dtype='object')
```

- We're only interested in IMDb data, so create a new table that takes the Film and all the columns relating to IMDB.

```
# We only want IMDb data, so create a new table that takes the Film and all the columns relating to IMDB
imdb_table = movie_file_pd[["FILM", "IMDB", "IMDB_norm",
                            "IMDB_norm_round", "IMDB_user_vote_count"]]
imdb_table.head()
```

	FILM	IMDB	IMDB_norm	IMDB_norm_round	IMDB_user_vote_count
0	Avengers: Age of Ultron (2015)	7.8	3.90	4.0	271107
1	Cinderella (2015)	7.1	3.55	3.5	65709
2	Ant-Man (2015)	7.8	3.90	4.0	103660
3	Do You Believe? (2015)	5.4	2.70	2.5	3136
4	Hot Tub Time Machine 2 (2015)	5.1	2.55	2.5	19560

- Filter out only the good movies—i.e., any film with an IMDb score greater than or equal to 7 and remove the norm ratings.

```
# We only like good movies, so find those that scored over 7, and ignore the norm rating
good_movies = movie_file_pd.loc[movie_file_pd["IMDB"] > 7, ["FILM", "IMDB", "IMDB_user_vote_count"]]
good_movies.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 that you may not have heard about - i.e., anything with under 20K votes

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

- Finally, export this file to a spreadsheet, excluding the index, so we can keep track of our future watchlist.

```
# Finally, export this file to a spread so we can keep track of out new future watch list without the index
unknown_movies.to_excel("C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.2\\Output\\movieWatchlist.xlsx", index=False)
```

Cleaning Data

Sunday, September 29, 2019 8:40 PM

Open CleaningData.ipynb

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

- While these issues may not seem significant in the grand scheme of things, they can severely hinder the analysis and visualization of a dataset by skewing the data one way or another.
- Pandas includes methods through which its users can remove missing values, replace duplicates, and change values with relative ease.

- To delete a column from a data frame: **del <DataFrame>[<Column>]**

```
# Preview of the DataFrame
# Note that FIELD8 is likely a meaningless column
df.head()
```

	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

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

	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

- In order to figure out if any rows are missing data, simply run the **count()** method on the data frame and check that all columns contain equal values.

```
# Identify incomplete rows
df.count()
```

```
LastName     1776
FirstName    1776
Employer     1743
City         1776
State        1776
Zip          1776
Amount       1776
dtype: int64
```

- In order to drop rows with missing information from a DataFrame: **<DataFrame>.dropna(how="any")**

```
# Drop all rows with missing information
df = df.dropna(how='any')
```

```
# Verify dropped rows
df.count()
```

```
LastName     1743
FirstName    1743
Employer     1743
City         1743
State        1743
Zip          1743
Amount       1743
dtype: int64
```

- To find values that have similar/misspelled values, run the **value_counts()** method on the column in question and look through the values that are returned.

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

```
None                                    249
Self                                    241
Retired                                 126
Self Employed                            39
Self-Employed                            34
                                       ...
```

- To replace similar/misspelled values, run the **replace()** method on the column in question and pass a dictionary into it with the keys being those values to replace and the value being those to replace the originals with.

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

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

```
Self-Employed                                314
None                                         249
Retired                                      126
Google                                         6
Not Employed                                   4
                                              ...
```

TEAM UP: Portland Crime

Sunday, September 29, 2019 8:40 PM

Overview

Students will now take a crime dataset from Portland and do their best to clean it up so that the DataFrame is consistent and no rows with missing data are present

Files

PortlandCrime.ipynb
Crime_incident_data2017.csv

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 "Offense Type" column and replace any similar values with one consistent value
- Create a couple DataFrames that look into one Neighborhood only and print them to the screen

Review: Portland Crime

Sunday, September 29, 2019 8:43 PM

Keys

- Read in the csv using Pandas and print out the DataFrame that is returned

```
# Import Dependencies
import pandas as pd
```

```
# Reference the file where the CSV is located
crime_csv_path = "C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.2\\Data\\crime_incident_data2017.csv"

# Import the data into a Pandas DataFrame
crime_df = pd.read_csv(crime_csv_path)
crime_df
```

- Get a count of rows within the DataFrame in order to determine if there are any null values

```
# look for missing values
crime_df.count()
```

```
Address              37365
Case Number          41032
Crime Against        41032
Neighborhood         39712
Number of Records    41032
Occur Date           41032
Occur Month Year     41032
Occur Time           41032
```

- Drop the rows which contain null values

```
# drop null rows
no_null_crime_df = crime_df.dropna(how='any')
```

```
# verify counts
no_null_crime_df.count()
```

```
Address              36146
Case Number          36146
Crime Against        36146
Neighborhood         36146
Number of Records    36146
Occur Date           36146
Occur Month Year     36146
Occur Time           36146
```

- Search through the "Offense Type" column and replace any similar values with one consistent value

```
# Check to see if there are any values with mispelled or similar values in "Offense Type"
no_null_crime_df["Offense Type"].value_counts()
```

```
Credit Card/ATM Fraud                          226
Arson                                          200
Prostitution                                   145
Pocket-Picking                                  94
Purse-Snatching                                 89
Embezzlement                                    73
Stolen Property Offenses                        57
Kidnapping/Abduction                            22
Theft From Coin-Operated Machine or Device      20
Hacking/Computer Invasion                       19
Animal Cruelty                                  17
Pornography/Obscene Material                    10
Extortion/Blackmail                              8
Assisting or Promoting Prostitution              7
Drug Equipment Violations                        6
Impersonation                                    4
Wire Fraud                                       3
Welfare Fraud                                    1
Commercial Sex Acts                              1
Name: Offense Type, dtype: int64
```

```
# Combining similar offenses together
no_null_crime_df = no_null_crime_df.replace(
    {"Commercial Sex Acts": "Prostitution", "Assisting or Promoting Prostitution": "Prostitution"})
no_null_crime_df
```

- Create a couple DataFrames that look into one Neighborhood only and print them to the screen

```
# Create a new DataFrame that looks into a specific neighborhood
vernon_crime_df = no_null_crime_df.loc[no_null_crime_df["Neighborhood"] == "Vernon"]
vernon_crime_df
```

	Address	Case Number	Crime Against	Neighborhood	Number of Records	Occur Date	Occur Month Year	Occur Time	Offense Category	Offense Count	Offense Type	Open Data Lat	Open Data Lon
6	5000 BLOCK OF NE 19TH AVE	17-901079	Property	Vernon	1	11/8/13	11/1/13	1200	Fraud Offenses	1	False Pretenses/Swindle/Confidence Game	45.5594	-122.646
7	5000 BLOCK OF NE 19TH AVE	17-901079	Property	Vernon	1	11/8/13	11/1/13	1200	Fraud Offenses	1	Identity Theft	45.5594	-122.646
147	1000 BLOCK OF NE EMERSON ST	17-901190	Property	Vernon	1	11/26/16	11/1/16	2040	Fraud Offenses	1	Identity Theft	45.5619	-122.655

Pandas Recap and Data Types

Sunday, September 29, 2019 8:45 PM

Open PandasRecap.ipynb

- Reading in data and printing the first 5 rows
  - Read_csv()
  - Head()

```
# Import the Pandas library
import pandas as pd
```

```
# Create a reference the CSV file desired
csv_path = "Resources/ufoSightings.csv"

# Read the CSV into a Pandas DataFrame
ufo_df = pd.read_csv(csv_path)

# Print the first five rows of data to the screen
ufo_df.head()
```

- Checking to see if there are any rows with missing data
  - Count()

```
# Check to see if there are any rows with missing data
ufo_df.count()
```

```
datetime                80332
city                    80332
state                   74535
country                 70662
shape                   78400
duration (seconds)      80332
duration (hours/min)    80332
comments                80317
```

- Delete Rows with missing data
  - Dropna(how="any")

```
# Remove the rows with missing data
clean_ufo_df = ufo_df.dropna(how="any")
clean_ufo_df.count()
```

```
datetime                  66516
city                      66516
state                     66516
country                   66516
shape                     66516
duration (seconds)        66516
duration (hours/min)      66516
comments                  66516
```

- Collect list of all sighting seen in the US
  - .loc[ ]

```
# Collect a list of sightings seen in the US
columns = [
    "datetime",
    "city",
    "state",
    "country",
    "shape",
    "duration (seconds)",
    "duration (hours/min)",
    "comments",
    "date posted"
]

# Filter the data so that only those sightings in the US are in a DataFrame
usa_ufo_df = clean_ufo_df.loc[clean_ufo_df["country"] == "us", columns]
usa_ufo_df.head()
```

- Count sightings per state
  - Value_counts()

```
# Count how many sightings have occured within each state
state_counts = usa_ufo_df["state"].value_counts()
state_counts
```

```
ca    8683
fl    3754
wa    3707
tx    3398
ny    2915
il    2447
az    2362
pa    2319
oh    2251
mi    1781
nc    1722
or    1667
```

- Covert state_counts series into DataFrame
  - Pd.DataFrame()

```
# Convert the state_counts Series into a DataFrame
state_ufo_counts_df = pd.DataFrame(state_counts)
state_ufo_counts_df.head()
```

	state
ca	8683
fl	3754
wa	3707
tx	3398
ny	2915

- Covert column name into "Sum of Sightings"
  - Rename()

```
# Convert the column name into "Sum of Sightings"
state_ufo_counts_df = state_ufo_counts_df.rename(
    columns={"state": "Sum of Sightings"})
state_ufo_counts_df.head()
```

	Sum of Sightings
ca	8683
fl	3754
wa	3707
tx	3398
ny	2915

- Change data type for duration (seconds)
  - Astype()

```
# Want to add up the seconds UFOs are seen? There is a problem
# Problem can be seen by examining datatypes within the DataFrame
usa_ufo_df.dtypes
```

```
datetime               object
city                   object
state                  object
country                object
shape                  object
duration (seconds)     object
duration (hours/min)   object
comments               object
date posted            object
dtype: object
```

```
# Using astype() to convert a column's data into floats
usa_ufo_df.loc[:, "duration (seconds)"] = usa_ufo_df["duration (seconds)"].astype("float")
usa_ufo_df.dtypes
```

```
datetime                object
city                    object
state                   object
country                 object
shape                   object
duration (seconds)     float64
duration (hours/min)    object
comments                object
date posted             object
dtype: object
```

Pandas Grouping

Sunday, September 29, 2019 8:41 PM

Open GroupBy.ipynb

The start of the code is much the same as earlier. Import in dependencies, remove all rows with missing data, convert the "duration (seconds)" column to numeric, filter the DataFrame so only US info is shown, and count the number of sightings per state.

- The **`df.groupby([<Columns>])`** method is then used in order to split the DataFrame into multiple groups with each group being a different state within the US.
- The object returned by the **`.groupby()`** method is a GroupBy object and cannot be accessed like a normal DataFrame.
- One of the only ways in which to access values within a GroupBy object is by using a data function on it.

```
# 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.generic.DataFrameGroupBy object at 0x00000213EE41CE10>
```

state	datetime	city	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
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

- It is possible to create new DataFrames using purely GroupBy data. This can by done by taking the `pd.DataFrame()` method and passing the GroupBy data desired in as the parameter.
- 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.

```
# 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()
```

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

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

	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

- **`df.groupby()`** method can be performed on multiple columns as well. This can be done by simply passing two or more column references into the list parameter

```
# 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 = clean_ufo_df.groupby(['country','state'])
grouped_international_data.count().head(20)
```

		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

- A new DataFrame can be created from a GroupBy object

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

		duration (seconds)

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

		duration (seconds)
country	**state**	
	al	900.00
	dc	300.00
	nt	360.00
au	oh	180.00
	sa	305.00

TEAM UP: Building a PokeDex**

Sunday, September 29, 2019 8:42 PM

Overview

Students will now take some time to create a DataFrame that visualizes the average stats for each type of Pokemon from the popular video game series. They will do so using the GroupBy() method and then converting their findings into a DataFrame

Files

pokemon.ipynb
pokemon.csv

Instructions

- Read the Pokemon CSV file with Pandas.
- 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.

Review: Building a PokeDex

Sunday, September 29, 2019 8:44 PM

Keys

- Create a new table by extracting the following columns: "Type 1", "HP", "Attack", "Sp. Atk", "Sp. Def", and "Speed".

```
# Extract the following columns: "Type 1", "HP", "Attack", "Sp. Atk", "Sp. Def", and "Speed"
pokemon_type = pokemon_pd[["Type 1", "HP", "Attack",
                           "Defense", "Sp. Atk", "Sp. Def", "Speed"]]
pokemon_type.head()
```

	Type 1	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
0	Grass	45	49	49	65	65	45
1	Grass	60	62	63	80	80	60
2	Grass	80	82	83	100	100	80
3	Grass	80	100	123	122	120	80
4	Fire	39	52	43	60	50	65

- Find the average stats for each type of Pokemon.
- Create a new DataFrame out of the averages.

```
# Create a dataframe of the average stats for each type of pokemon.
pokemon_group = pokemon_type.groupby(["Type 1"])

pokemon_comparison = pokemon_group.mean()
pokemon_comparison
```

Type 1	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Bug	56.884058	70.971014	70.724638	53.869565	64.797101	61.681159
Dark	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290
Dragon	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250
Electric	59.795455	69.090909	66.295455	90.022727	73.704545	84.500000
Fairy	74.117647	61.529412	65.705882	78.529412	84.705882	48.588235
Fighting	69.851852	96.777778	65.925926	53.111111	64.703704	66.074074
Fire	69.903846	84.769231	67.769231	88.980769	72.211538	74.442308
Flying	70.750000	78.750000	66.250000	94.250000	72.500000	102.500000
Ghost	64.437500	73.781250	81.187500	79.343750	76.468750	64.343750
Grass	67.271429	73.214286	70.800000	77.500000	70.428571	61.928571
Ground	73.781250	95.750000	84.843750	56.468750	62.750000	63.906250
Ice	72.000000	72.750000	71.416667	77.541667	76.291667	63.458333
Normal	77.275510	73.469388	59.846939	55.816327	63.724490	71.551020

- 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.

```
# Calculate the total power level of each type of pokemon by summing all of the stats together.
# Place the results into a new column.
pokemon_comparison["Total"] = pokemon_comparison.sum(axis=1)

pokemon_comparison["Total"]
```

```
Type 1
Bug         378.927536
Dark        445.741935
Dragon      550.531250
Electric    443.409091
Fairy       413.176471
Fighting    416.444444
Fire        458.076923
Flying      485.000000
Ghost       439.562500
Grass       421.142857
Ground      437.500000
Ice         433.458333
Normal      401.683673
Poison      399.142857
Psychic     475.947368
Rock        453.750000
Steel       487.703704
Water       430.455357
Name: Total, dtype: float64
```

```
# Bonus: Sort the table by strongest type and export the resulting table to a new CSV.
strongest_pokemon = pokemon_comparison.sort_values(["Total"], ascending=False)
strongest_pokemon.to_csv("output/pokemon_rankings.csv", index=True)
```

Sorting Made Easy

Sunday, September 29, 2019 8:44 PM

Open Sorting.ipynb

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

```
# 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()
```

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom	Genero
139	Angola	140	3.795	3.951642	3.638358	0.858428	1.104412	0.049869	0.000000	0.097
129	Sudan	130	4.139	4.345747	3.932253	0.659517	1.214009	0.290921	0.014996	0.182
144	Haiti	145	3.603	3.734715	3.471285	0.368610	0.640450	0.277321	0.030370	0.489
153	Burundi	154	2.905	3.074690	2.735310	0.091623	0.629794	0.151611	0.059901	0.204
151	Syria	152	3.462	3.663669	3.260331	0.777153	0.396103	0.500533	0.081539	0.493

- The parameter of "ascending" is always marked as True by default. This means that the **`sort_vales()`** method will always sort from lowest to highest
- Setting the parameter of **`ascending=False`** will sort from highest to lowest

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

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498273	0.658249
0	Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.796667	0.635423
128	Cambodia	129	4.168	4.278518	4.057483	0.601765	1.006238	0.429783	0.633376
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007

- You can sort based upon the values stored within multiple columns by passing a list of columns into the **`sort_values()`** method as a parameter.
- The first column will be the primary sorting method with ties being broken by the second column.

```
# It is possible to sort based upon multiple columns
family_and_generosity = happiness_df.sort_values(
    ["Family", "Generosity"], ascending=False)
family_and_generosity.head()
```

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom	Gener
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163	0.47
14	Ireland	15	6.977	7.043352	6.910649	1.535707	1.558231	0.809783	0.573110	0.42
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007	0.35
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498273	0.658249	0.41
7	New Zealand	8	7.314	7.379510	7.248490	1.405706	1.548195	0.816760	0.614062	0.50

- The **`df.reset_index()`** method recalculates the index for each row based upon their position within the new DataFrame which will allow for easier referencing of rows in the future.
- Passing **`drop=True`** into **`df.reset_index()`** will ensure no new column is created when the index is reset.

```
# The index can be reset to provide index numbers based on the new rankings.
new_index = family_and_generosity.reset_index(drop=True)
new_index.head()
```

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom	Genero
0	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163	0.475
1	Ireland	15	6.977	7.043352	6.910649	1.535707	1.558231	0.809783	0.573110	0.427
2	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007	0.355
3	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498273	0.658249	0.415
4	New Zealand	8	7.314	7.379510	7.248490	1.405706	1.548195	0.816760	0.614062	0.500

Exercise: Search for the Worst

Sunday, September 29, 2019 8:45 PM

Overview

Students will now take a dataset composed of soccer player statistics and will attempt to determine which players are the worst in the world at their particular position

Files

SearchForTheWorst.ipynb
Soccer2018Data.csv

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.

Review: Search for the Worst

Monday, September 30, 2019 11:24 PM

Keys

- Print out a list of all of the values within the "Preferred Position" column.

```
# Collect a list of all the unique values in "Preferred Position"
soccer_2018_df["Preferred Position"].unique()
```

```
array(['ST', 'RW', 'LW', 'GK', 'CDM', 'CB', 'RM', 'CM', 'LM', 'LB', 'CAM',
       'RB', 'CF', 'RWB', 'LWB'], dtype=object)
```

- Select a value from this list and create a new DataFrame that only includes players who prefer that position.

```
# Looking only at strikers (ST) to start
strikers_2018_df = soccer_2018_df.loc[soccer_2018_df["Preferred Position"] == "ST", :]
strikers_2018_df.head()
```

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM	RF	RM	RS
0	Cristiano Ronaldo	32	Portugal	94	94	Real Madrid CF	ST	89.0	53.0	62.0	...	61.0	53.0	82.0	62.0	91.0	89.0	92.0
3	L. Suárez	30	Uruguay	92	92	FC Barcelona	ST	87.0	58.0	65.0	...	64.0	58.0	80.0	65.0	88.0	85.0	88.0
5	R. Lewandowski	28	Poland	91	91	FC Bayern Munich	ST	84.0	57.0	62.0	...	58.0	57.0	78.0	62.0	87.0	82.0	88.0
9	G. Higuaín	29	Argentina	90	90	Juventus	ST	81.0	46.0	52.0	...	51.0	46.0	71.0	52.0	84.0	79.0	87.0
16	S. Agüero	29	Argentina	89	89	Manchester City	ST	85.0	44.0	54.0	...	52.0	44.0	75.0	54.0	87.0	84.0	86.0

5 rows × 33 columns

- 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.

```
# Sort the DataFrame by the values in the "ST" column to find the worst
strikers_2018_df = strikers_2018_df.sort_values("ST")

# Reset the index so that the index is now based on the sorting locations
strikers_2018_df = strikers_2018_df.reset_index(drop=True)

strikers_2018_df.head()
```

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM	RF	RM	RS	RW	RWB	ST
0	L. Sackey	18	Ghana	46	64	Scunthorpe United	ST	29.0	45.0	38.0	...	40.0	45.0	30.0	38.0	29.0	30.0	31.0	29.0	38.0	31.0
1	M. Zettl	18	Germany	50	67	SpVgg Unterhaching	ST	47.0	32.0	36.0	...	39.0	32.0	42.0	36.0	46.0	49.0	43.0	49.0	41.0	43.0
2	O. Sowunmi	21	England	59	71	Yeovil Town	ST	35.0	58.0	47.0	...	52.0	58.0	37.0	47.0	38.0	38.0	44.0	37.0	49.0	44.0
3	E. Mason-Clark	17	England	50	63	Barnet	ST	49.0	33.0	35.0	...	39.0	33.0	42.0	35.0	49.0	50.0	45.0	51.0	40.0	45.0
4	J. Young	17	Scotland	46	61	Swindon Town	ST	44.0	28.0	29.0	...	31.0	28.0	38.0	29.0	45.0	42.0	45.0	44.0	32.0	45.0

5 rows × 33 columns

- Print out the statistics for the worst player in a position to the screen.

```
# Save all of the information collected on the worst striker
worst_striker = strikers_2018_df.loc[0, :]
worst_striker
```

```
Name                          L. Sackey
Age                                  18
Nationality                       Ghana
Overall                              46
Potential                            64
Club                  Scunthorpe United
Preferred Position                   ST
CAM                                  29
CB                                   45
CDM                                  38
CF                                   29
CM                                   30
LAM                                  29
LB                                   40
LCB                                  45
LCM                                  30
LDM                                  38
LF                                   29
LM                                   30
```