Welcome to Class

Overview

Today's lesson will introduce students to Jupyter Notebook and the basics of the Pandas module

Class Objectives

- Connect to their development environment and open Jupyter notebook files from local directories
- Create Pandas DataFrames from scratch
- Understand how to run functions on Pandas DataFrames
- Know how to read/write DataFrames from/to CSV files using Pandas

Create a Virtual Environment

Virtual Environment

- Create an isolated environment for Python Projects
- Different projects can have different dependencies
    - Different projects might use different types and versions of libraries
- The virtual environment we create will make sure the we have all the right dependencies for future classes
- Link: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Creating a Virtual Environment

- We are going to create a virtual environment that will run Python 3.6. This will help solve issues where students have multiple versions of Python installed.
- Type "conda create -n PythonData python=3.6 anaconda" in the terminal
    - This might take a while
- Once complete, type "source activate PythonData" to activate the environment
- When "(PythonData)" appears, that means you are in the environment
- Type "python - - version" to make sure you are using the correct version of python
- **You will need to activate this environment each time you open a new console**
- You can exit the environment by entering "source deactivate" or "conda deactivate

Introduction to Jupyter Notebook

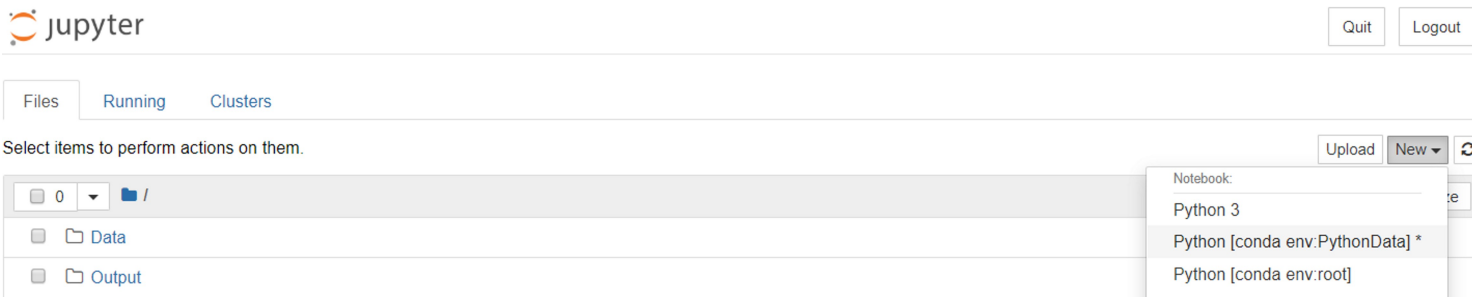
- Jupyter Notebook is an open-source application that allows its users to create documents that contain live code, equations, visualizations, and explanatory text.
- Essentially, Jupyter Notebook combines a text editor, the console, and a markdown file into
- Make sure the **PythonData** development environment that was created is activated, then type ***jupyter notebook*** into the terminal.

```
$ jupyter notebook
[I 15:06:20.131 NotebookApp] [nb_conda_kernels] enabled, 3 kernels found
[I 15:06:22.651 NotebookApp] Serving notebooks from local directory: C:\Users\jacob\OneDrive\Documents\WorkAndSchool\TeachingAssistant\DataViz\DataViz-Lesson-Plans
[I 15:06:22.651 NotebookApp] 0 active kernels
[I 15:06:22.651 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/?token=dcad7e0a3dfeaaa37f720cf0e284c7e69b918388f712f0fb
[I 15:06:22.651 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:06:22.657 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=dcad7e0a3dfeaaa37f720cf0e284c7e69b918388f712f0fb
[I 15:06:22.946 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

- Running **jupyter notebook** will automatically open up a webpage where users can navigate into any files/folders within the folder they ran the command from.
- Users can create new Jupyter Notebook files from the webpage that is opened by clicking the **"new"** button and selecting their development environment from the list that appears.
- Python files created through Jupyter Notebook are given the **ipynb** extension rather than **py** and cannot be easily read/altered within a typical text editor.

- Create a new Python file using Jupyter Notebook, making sure to set the kernel as "PythonData"
    - Setting the kernel for Jupyter projects is important because these kernels let the program know which libraries will be available for use. Only those libraries loaded into the development environment selected can be used in a Jupyter Notebook project.



- If the user's development environment does not show up within Jupyter Notebook, simply run the command `conda install -c anaconda nb_conda_kernels` within the terminal so that anaconda environments can be used as kernels.

Open JupyterIntro.ipynp [Instructor Demo]

- Each cell contains Python code which can be run independently by placing the cursor inside a cell and pressing **Shift + Enter**.
- Jupyter notebooks allow users to both to experiment with code directly and save it for later
- Values in Jupyter Notebooks are stored based upon what lines of code were run previously

Everyone Do: Netflix Remix

Key Points

- The code within a cell can be run by placing the cursor inside of it and hitting `Shift + Enter`
- If the code within a cell is not run, then the changes made within will not be saved into memory
- The code contained within a Jupyter Notebook is not linear.
    - For example, if two cells modify the same variable, then whichever block of code was run last will ultimately determine the value of the variable

In [1]:
```
# Modules
import os
import csv
```

In [2]:
```
# Prompt user for video lookup
video = input("What show or movie are you looking for? ")
```

```
What show or movie are you looking for? Grease
```

In [3]:
```
# Set path for file
csvpath = os.path.join("C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.1\\Data\\netflix_ratings.csv")
print(csvpath)

# Set variable to check if we found the video
found = False
```

```
C:\Users\dedwa\OneDrive\Desktop\DataViz\Class 4.1\Data\netflix_ratings.csv
```

In [4]:
```
# Open the CSV
with open(csvpath, newline="") as csvfile:
    csvreader = csv.reader(csvfile, delimiter=",")

    # Loop through looking for the video
    for row in csvreader:
        if row[0] == video:
            print(row[0] + " is rated " + row[1] +
                  " with a rating of " + row[5])

            # Set variable to confirm we have found the video
            found = True

    # If the video is never found, alert the user
    if found == False:
        print("We don't seem to have what you are looking for!")
```

```
Grease is rated PG with a rating of 86
```

Introduction to Pandas

Open "creating_data_frames.ipynb"

- The way Jupyter Notebook allows for the testing and visualization of code really starts to shine through when these principles are applied to large tables.
- The Pandas library is extraordinarily powerful when it comes to visualizing, analyzing, and altering large datasets.
- While Python alone is stuck using lists, tuples, and dictionaries, Pandas lets Python programmers work with "Series" and "DataFrames"
    - These two data types - unique to Pandas - are essentially structured lists, with many built-in convenience methods that allow for quick and easy manipulation of data

- In order to use **Pandas**, we must **import** is it ("import pandas as pd")

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

- A Pandas **Series** is a one-dimensional labeled array capable of holding any data type.
    - Like an array, the data is linear but also, like a dictionary, it has an index that acts like a key.
- To create a Series, simply use the pd.Series() function and place a list within the parentheses.
    - The index for the values within the Series will be the numeric index of the initial list.

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

In [2]:
```
# 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
```

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

- A Pandas **DataFrame** is a two-dimensional labeled data structure with columns of potentially different types.
    - The easiest way to think of it is like an Excel spreadsheet with each column being a Series.
- 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 will be placed inside the table

In [3]:
```
# Convert a list of dictionarys into a dataframe
states_dicts = [{"STATE": "New Jersey", "ABBREVIATION": "NJ"},
                {"STATE": "New York", "ABBREVIATION": "NY"}]

df_states = pd.DataFrame(states_dicts)
df_states
```

Out[3]:

	STATE	ABBREVIATION
0	New Jersey	NJ
1	New York	NY

- Another way is 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.

In [4]:
```
# Convert a single dictionary containing lists into a dataframe
df = pd.DataFrame(
    {"Dynasty": ["Early Dynastic Period", "Old Kingdom"],
     "Pharoh": ["Thinis", "Memphis"]
     }
)
df
```

Out[4]:

	Dynasty	Pharoh
0	Early Dynastic Period	Thinis
1	Old Kingdom	Memphis

Exercise: Data-Frame Shop**

Overview

Students will now try their hand at creating DataFrames from scratch using the two methods discussed earlier. This will also provide them with the opportunity to better understand what DataFrames look like

Files

DataFrameShop.ipynb

Instructions

- Create a DataFrame for a frame shop that contains three columns - "Frame", "Price", and "Sales" - and has five rows of data stored within it.
- Using an alternate method from that used before, create a DataFrame for an art gallery that contains three columns - "Painting", "Price", and "Popularity" - and has four rows of data stored within it.

Review: Data-Frame Shop Review

Keys

- It is important to save the DataFrames created to a variable or else they will only be printed to the screen and will not be available for use later on.
- The dictionary of lists method
  - More time effective since the keys need only be written once.
  - It can be harder to read through
    - if even one of the lists contains fewer values than the others than an error will be returned.

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

In [2]:
```
# 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
```

Out[2]:

	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

- The list of dictionaries method
  - Takes far longer to write the code for since the keys have to be re-written each time.
  - Allow the programmer to better understand what each row in their DataFrame will look like though.

In [3]:

```
# Create a DataFrame of paintings using 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
```

Out[3]:

	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

DataFrame Functions

- There many functions/methods that come packaged with Pandas that allow for quick and easy analysis of large datasets
- **`head()`** which takes a DataFrame and shows only the first five rows of data inside of it.
    - This number can be increased/decreased, however, by placing an integer within the parentheses.
- Allows the programmer to look at a minified version of a much larger table, thus allowing them to make informed changes without having to search through the entire dataset.

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

In [2]:
```
# Save path to data set in a variable
data_file = "C:\\Users\\dedwa\\OneDrive\\Desktop\\DataViz\\Class 4.1\\Data\\dataSet.csv"
```

In [3]:
```
# Use Pandas to read data
data_file_pd = pd.read_csv(data_file)
data_file_pd.head()
```

Out[3]:

	id	First Name	Last Name	Gender	Amount
0	1	Todd	Lopez	M	8067.7
1	2	Joshua	White	M	7330.1
2	3	Mary	Lewis	F	16335.0
3	4	Emily	Burns	F	12460.8
4	5	Christina	Romero	F	15271.9

- **`describe()`** which will print out a DataFrame containing some analytic information on the table and its columns

In [4]:
```
# Display a statistical overview of the DataFrame
data_file_pd.describe()
```

Out[4]:

	id	Amount
count	1000.000000	1000.000000
mean	500.500000	10051.323600
std	288.819436	5831.230806
min	1.000000	3.400000
25%	250.750000	4854.875000
50%	500.500000	10318.050000
75%	750.250000	15117.425000

50%	500.500000	10318.050000
75%	750.250000	15117.425000
max	1000.000000	19987.400000

- Most data functions can also be performed on a Series by referencing a single column within the whole DataFrame. This is done in a similar way to referencing a key within dictionary by taking the DataFrame and following it up with brackets with the desired column's header contained within like a key.

```
In [5]: # Reference a single column within a DataFrame
        data_file_pd["Amount"].head()

Out[5]: 0     8067.7
        1     7330.1
        2    16335.0
        3    12460.8
        4    15271.9
        Name: Amount, dtype: float64
```

```
In [7]: # The mean method averages the series
        average = data_file_pd["Amount"].mean()
        average

Out[7]: 10051.323600000002
```

```
In [8]: # The sum method adds every entry in the series
        total = data_file_pd["Amount"].sum()
        total

Out[8]: 10051323.600000001
```

- Multiple columns can be referenced as well 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)

In [6]:
```
# Reference multiple columns within a DataFrame
data_file_pd[["Amount", "Gender"]].head()
```

Out[6]:

	Amount	Gender
0	8067.7	M
1	7330.1	M
2	16335.0	F
3	12460.8	F
4	15271.9	F

Exercise: Training Grounds**

Overview

Students will now take a large DataFrame consisting of 200 rows, analyze it using some data functions, and then add a new column into it

Files

TrainingGrounds.ipynb

Instructions

- Using the DataFrame provided, perform all of the following actions...
    - Provide a simple, analytical overview of the dataset's numeric columns
    - Collect all of the names of the trainers within the dataset
    - Figure out how many students each trainer has
    - Find the average weight of the students at the gym
    - Find the combined weight of all of the students at the gym
    - Convert the "Membership (Days)" column into weeks and then add this new series into the DataFrame

Review: Training Grounds

Keys

- Good practice to take a look at your data before starting analysis

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

```
# A seriously gigantic DataFrame of individuals' names, their trainers, their weight, and their days as gym members
training_data = pd.DataFrame({
    "Name":["Gino Walker","Hiedi Wasser","Kerrie Wetzel","Elizabeth Sackett","Jack Mitten","Madalene Wayman","Jamee Horvath"
    "Trainer":['Bettyann Savory','Mariah Barberio','Gordon Perrine','Pa Dargan','Blanch Victoria','Aldo Byler','Aldo Byler',
    "Weight":[128,180,193,177,237,166,224,208,177,241,114,161,162,151,220,142,193,193,124,130,132,141,190,239,213,131,172,12
    "Membership (Days)":[52,70,148,124,186,157,127,155,37,185,158,129,93,69,124,13,76,153,164,161,48,121,167,69,39,163,7,34,
})
training_data.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

- Provide a simple, analytical overview of the dataset's numeric columns

```
In [3]: # Collecting a summary of all numeric data
        training_data.describe()
```

Out[3]:

	Weight	Membership (Days)
count	200.000000	200.000000
mean	180.820000	101.910000
std	39.372689	60.162025
min	110.000000	1.000000
25%	151.000000	51.000000
50%	180.500000	105.500000
75%	215.000000	157.250000
max	250.000000	200.000000

- Collect all of the names of the trainers within the dataset

```
In [4]: # Finding the names of the trainers
        training_data["Trainer"].unique()

Out[4]: array(['Bettyann Savory', 'Mariah Barberio', 'Gordon Perrine',
               'Pa Dargan', 'Blanch Victoria', 'Aldo Byler', 'Williams Camire',
               'Junie Ritenour', 'Barton Stecklein', 'Brittani Brin',
               'Phyliss Houk', 'Calvin North', 'Coleman Dunmire',
               'Harland Coolidge'], dtype=object)
```

- Figure out how many students each trainer has

```
In [5]: # Finding how many students each trainer has
        training_data["Trainer"].value_counts()

Out[5]: Bettyann Savory     20
        Coleman Dunmire     17
        Aldo Byler          17
        Phyliss Houk        16
        Brittani Brin       16
        Mariah Barberio     15
        Gordon Perrine      14
        Pa Dargan           14
        Junie Ritenour      14
        Blanch Victoria     14
        Barton Stecklein    13
        Harland Coolidge    12
        Williams Camire     11
        Calvin North         7
        Name: Trainer, dtype: int64
```

- Find the average weight of the students at the gym

```
In [6]: # Finding the average weight of all students
        training_data["Weight"].mean()

Out[6]: 180.82
```

- Find the combined weight of all of the students at the gym

```
In [7]: # Finding the combined weight of all students
        training_data["Weight"].sum()

Out[7]: 36164
```

- Convert the "Membership (Days)" column into weeks and then add this new series into the DataFrame
- Takes the values stored within the initial column (“Membership

(Days)"), divides them by seven, and then adds this edited series into a newly created column ("Membership (weeks)")

In [8]:
```
# Converting the membership days into weeks and then adding a column to the DataFrame
weeks = training_data["Membership (Days)"]/7
training_data["Membership (Weeks)"] = weeks

training_data.head()
```

Out[8]:

	Name	Trainer	Weight	Membership (Days)	Membership (Weeks)
0	Gino Walker	Bettyann Savory	128	52	7.428571
1	Hiedi Wasser	Mariah Barberio	180	70	10.000000
2	Kerrie Wetzel	Gordon Perrine	193	148	21.142857
3	Elizabeth Sackett	Pa Dargan	177	124	17.714286
4	Jack Mitten	Blanch Victoria	237	186	26.571429

Modifying Columns

Open ColumnManipulation.ipynb

- As we saw in the previous activity, columns within a DataFrame are not always placed within the desired position by default…….sometimes they may not even have a descriptive or concise enough name.
- We can modify the names/placement of columns using the **rename()** function and the use of double brackets.

- In order to collect a list of all the columns contained within a DataFrame, we use the **df.columns** call and an object containing the column headers will be printed to the screen.

```
In [3]: # Collecting a list of all columns within the DataFrame
        training_data.columns
```

```
Out[3]: Index(['Membership(Days)', 'Name', 'Trainer', 'Weight'], dtype='object')
```

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

```
In [4]: # Reorganizing the columns using double brackets
        organized_df = training_data[["Name","Trainer","Weight","Membership(Days)"]]
        organized_df.head()
```

Out[4]:

	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

- **It is also possible to remove columns in this way by simply not creating a reference to them. This will, in essence, drop them from the newly made DataFrame.*

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

parentheses.

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

In [5]:
```
# Using .rename(columns={}) in order to rename columns
renamed_df = organized_df.rename(columns={"Membership(Days)":"Membership in Days", "Weight":"Weight in Pounds"})
renamed_df.head()
```

Out[5]:

	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

Exercise: Hey Arnold!

Overview

Students will be taking a pre-made DataFrame of "Hey Arnold!" characters and reorganizing it so that it is more understandable and organized

Files

Hey_arnold.ipynb

Instructions

- Rename columns within the Dataframe
  - Character_in_Show -> Character
  - Color_of_hair -> Hair Color
  - Height -> Height
  - Football_Shaped_Head -> Football Head
- Create a new Data Frame with columns in this order
  - Character
  - Football Head
  - Hair Color
  - Height

Review: Hey Arnold!

Keys

- Take the currently named columns and convert them into the more readable versions using `hey_arnold.rename(columns={})` and applying the changes to a new data frame.

```
In [3]:
# Rename columns for readability
hey_arnold_renamed = hey_arnold.rename(columns={"Character_in_show": "Character",
                                                "color_of_hair": "Hair Color",
                                                "Height": "Height",
                                                "Football_Shaped_Head": "Football Head"
                                                })
hey_arnold_renamed
```

Out[3]:

	Character	Hair Color	Height	Football Head
0	Arnold	blonde	average	True
1	Gerald	black	tallish	False
2	Helga	blonde	tallish	False
3	Phoebe	black	short	False
4	Harold	unknown	tall	False
5	Eugene	red	short	False

- Then, using double brackets, the new columns are reorganized and placed into another new variable which now holds the fully formatted data frame.

```
In [4]:
# Organize the columns so they are in a more logical order
hey_arnold_alphabetical = hey_arnold_renamed[[
    "Character", "Football Head", "Hair Color", "Height"]]
hey_arnold_alphabetical
```

Out[4]:

	Character	Football Head	Hair Color	Height
0	Arnold	True	blonde	average
1	Gerald	False	black	tallish
2	Helga	False	blonde	tallish
3	Phoebe	False	black	short
4	Harold	False	unknown	tall
5	Eugene	False	red	short

Break

Reading and Writing CSV

Sunday, September 29, 2019 12:38 PM

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. From then on, the DataFrame can be altered and manipulated like normal.
    - *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)
```

Exercise: GoodReads - Part II**

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
GoodReadsSummary.ipynb
New csv file

Instructions

- Using the modified DataFrame that was created earlier, create a summary table for the dataset that includes the following pieces of information...
    - The count of unique authors within the DataFrame
    - The year of the earliest published book in the DataFrame
    - The year of the latest published book in the DataFrame
    - The total number of reviews within the DataFrame

Review: GoodReads - Part II

Keys

- The count of unique authors can be found by using **len(goodreads_df["Authors"].unique())**
- Looks into the list of Authors, places all of the unique values into a list, and then finds the list's length.

In [5]:
```
# Calculate the number of unique authors in the DataFrame
author_count = len(goodreads_df["Authors"].unique())
```

- To find the earliest year, use the min() method on the "Publication Year" column.
- To find the latest year, use the max() method on the "Publication Year" column.

```
# Calculate the earliest/latest year a book was published
earliest_year = goodreads_df["Publication Year"].min()
latest_year = goodreads_df["Publication Year"].max()
```

- To find the total number of reviews given, use the sum() function on each of the " Star Reviews" columns and add up all of the values.

```
# Calculate the total reviews for the entire dataset
# Hint: use the pandas' sum() method to get the sum for each row
total_reviews = goodreads_df['One Star Reviews'].sum() + goodreads_df['Two Star Reviews'].sum() + goodreads_df['Three S
```

- In order to add these values into the summary DataFrame, the values must be placed within brackets. Without doing this, Pandas will return an error

In [6]:
```
# Place all of the data found into a summary DataFrame
summary_table = pd.DataFrame({"Total Unique Authors": [author_count],
                              "Earliest Year": earliest_year,
                              "Latest Year": latest_year,
                              "Total Reviews": total_reviews})
summary_table
```

Out[6]:

	Total Unique Authors	Earliest Year	Latest Year	Total Reviews
0	4664	-1750.0	2017.0	596873216