In [1]:
```
# Running the basic "Hello World" code
hello = "Hello World"
print(hello)
```

```
Hello World
```

In [2]:
```
# Doing simple math
4 + 4
```

Out[2]: 8

In [3]:
```
# Storing results in variables
a = 5
```

In [4]:
```
# Using those variables elsewhere in the code
a
```

Out[4]: 5

In [5]:
```
# Variables will hold the value most recently run
# This means that, if we run the code above, it will now print 2
a = 2
```

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? Last Man Standing
```

In [3]:
```
# Set path for file
csvpath = os.path.join("Resources", "netflix_ratings.csv")
print(csvpath)

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

```
Resources/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!")
```

```
Last Man Standing is rated TV-PG with a rating of 97
```

In [ ]:

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

In [2]:
```python
# We can create a Pandas Series from a raw list
data_series = pd.Series(["UCLA", "UC Berkeley", "UC Irvine",
                         "University of Central Florida", "Rutgers University"])
data_series
```

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

In [3]:
```python
# 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]:

	ABBREVIATION	STATE
0	NJ	New Jersey
1	NY	New York

In [4]:
```python
# 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

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
```

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]:

	ABBREVIATION	STATE
0	NJ	New Jersey
1	NY	New York

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

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

In [2]:
```
# Save path to data set in a variable
data_file = "Resources/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

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
max	1000.000000	19987.400000

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 [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

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

In [9]:
```
# The unique method shows every element of the series that appears only once
unique = data_file_pd["Last Name"].unique()
unique
```

Out[9]:
```
array(['Lopez', 'White', 'Lewis', 'Burns', 'Romero', 'Andrews', 'Baker',
       'Diaz', 'Burke', 'Richards', 'Hansen', 'Tucker', 'Wheeler',
       'Turner', 'Reynolds', 'Carpenter', 'Scott', 'Ryan', 'Marshall',
       'Fernandez', 'Olson', 'Riley', 'Woods', 'Wells', 'Gutierrez',
       'Harvey', 'Ruiz', 'Lee', 'Welch', 'Cooper', 'Nichols', 'Murray',
       'Gomez', 'Green', 'Jacobs', 'Griffin', 'Perry', 'Dunn', 'Gardner',
       'Gray', 'Walker', 'Harris', 'Lawrence', 'Black', 'Simpson', 'Sims',
       'Weaver', 'Carr', 'Owens', 'Stephens', 'Butler', 'Matthews', 'Cox',
       'Brooks', 'Austin', 'Moore', 'Hunter', 'Cunningham', 'Lane',
       'Montgomery', 'Vasquez', 'Freeman', 'Hernandez', 'Alexander',
       'Pierce', 'Mcdonald', 'Kelly', 'Foster', 'Bell', 'Johnson',
       'Bowman', 'Porter', 'Wood', 'Reid', 'Willis', 'Bishop',
       'Washington', 'Gonzales', 'Davis', 'Martinez', 'Martin', 'Long',
       'Howell', 'Hawkins', 'Knight', 'Price', 'Day', 'Bailey', 'Flores',
       'Young', 'Evans', 'Cruz', 'Chavez', 'Barnes', 'Coleman', 'Burton',
       'Clark', 'Carter', 'Franklin', 'Ellis', 'Miller', 'Allen', 'Mason',
       'Patterson', 'Stevens', 'Kim', 'Kelley', 'Robinson', 'Hughes',
       'Morgan', 'Dean', 'Stewart', 'Murphy', 'Fox', 'Simmons', 'Thompson',
       'Fuller', 'Peterson', 'Hanson', 'Wright', 'Reed', 'Graham',
       'Parker', 'Boyd', 'Taylor', 'Greene', 'George', 'Mills', 'Duncan',
       'Hill', 'Jordan', 'Stanley', 'Hall', 'James', 'Stone', 'Warren',
       'Fowler', 'Williamson', 'Lynch', 'Harper', 'Little', 'Nguyen',
       'Morrison', 'Ramirez', 'Howard', 'Watkins', 'Robertson', 'Powell',
       'Sanchez', 'Sanders', 'Grant', 'Ross', 'Mitchell', 'Henderson',
       'Rose', 'Perez', 'Berry', 'Watson', 'Gordon', 'Morales', 'Arnold',
       'Morris', 'Crawford', 'Smith', 'Medina', 'Alvarez', 'Collins',
       'Rodriguez', 'Mccoy', 'Bennett', 'Richardson', 'Chapman',
       'Johnston', 'Gilbert', 'Ford', 'Russell', 'Nelson', 'Castillo',
       'Cole', 'Rice', 'Payne', 'Frazier', 'Webb', 'Armstrong', 'Wilson',
       'Garza', 'Garrett', 'Spencer', 'Peters', 'Sullivan', 'Brown',
       'Williams', 'Gonzalez', 'Palmer', 'Fields', 'Snyder', 'Jackson',
       'Edwards', 'Anderson', 'Cook', 'Ramos', 'Harrison', 'Lawson',
       'Banks', 'Wallace', 'Ortiz', 'Gibson', 'Reyes', 'Shaw', 'Ward',
       'Perkins', 'Bradley', 'Rivera', 'Jenkins', 'Hart', 'Phillips',
       'Garcia', 'Fisher', 'King', 'Larson', 'Hunt', 'Jones', 'Hudson',
       'Myers', 'Hayes', 'Dixon', 'Schmidt', 'Moreno', 'Rogers', 'Thomas',
       'Meyer', 'Daniels', 'Bryant', 'Henry', 'Campbell', 'Ferguson',
       'Oliver', 'Ray', 'Carroll', 'Wagner', 'Kennedy', 'Holmes'], dtype=ob
ject)
```

In [10]:
```
# The value_counts method counts unique values in a column
count = data_file_pd["Gender"].value_counts()
count
```

Out[10]:
```
M    515
F    485
Name: Gender, dtype: int64
```

In [11]:

```
# Calculations can also be performed on Series and added into DataFrames as r
thousands_of_dollars = data_file_pd["Amount"]/1000
data_file_pd["Thousands of Dollars"] = thousands_of_dollars

data_file_pd.head()
```

Out[11]:

	id	First Name	Last Name	Gender	Amount	Thousands of Dollars
0	1	Todd	Lopez	M	8067.7	8.0677
1	2	Joshua	White	M	7330.1	7.3301
2	3	Mary	Lewis	F	16335.0	16.3350
3	4	Emily	Burns	F	12460.8	12.4608
4	5	Christina	Romero	F	15271.9	15.2719

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

In [2]:
```
# A seriously gigantic DataFrame of individuals' names, their trainers, their weight, and thei
training_data = pd.DataFrame({
    "Name":["Gino Walker","Hiedi Wasser","Kerrie Wetzel","Elizabeth Sackett","Jack Mitten","Ma
    "Trainer":['Bettyann Savory','Mariah Barberio','Gordon Perrine','Pa Dargan','Blanch Victor
    "Weight":[128,180,193,177,237,166,224,208,177,241,114,161,162,151,220,142,193,193,124,130,
    "Membership (Days)":[52,70,148,124,186,157,127,155,37,185,158,129,93,69,124,13,76,153,164,
})
training_data.head()
```

Out[2]:

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

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

Out[3]:

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

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)
```

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
Brittani Brin       16
Phyliss Houk        16
Mariah Barberio     15
Junie Ritenour      14
Gordon Perrine      14
Blanch Victoria     14
Pa Dargan           14
Barton Stecklein    13
Harland Coolidge    12
Williams Camire     11
Calvin North         7
Name: Trainer, dtype: int64
```

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

Out[6]:
```
180.82
```

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

Out[7]:
```
36164
```

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]:

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

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

In [2]:
```python
# A 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","Jame
    "Trainer":['Bettyann Savory','Mariah Barberio','Gordon Perrine','Pa Dargan','Blanch Victoria','Aldo Byler','Al
    "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,
    "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,
})
training_data.head(10)
```

Out[2]:

	Membership(Days)	Name	Trainer	Weight
0	52	Gino Walker	Bettyann Savory	128
1	70	Hiedi Wasser	Mariah Barberio	180
2	148	Kerrie Wetzel	Gordon Perrine	193
3	124	Elizabeth Sackett	Pa Dargan	177
4	186	Jack Mitten	Blanch Victoria	237
5	157	Madalene Wayman	Aldo Byler	166
6	127	Jamee Horvath	Aldo Byler	224
7	155	Arlena Reddin	Williams Camire	208
8	37	Tula Levan	Junie Ritenour	177
9	185	Teisha Dreier	Gordon Perrine	241

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

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

In [4]:
```python
# 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

In [5]:
```python
# 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

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

In [2]:
```python
# Create a DataFrame with given columns and value
hey_arnold = pd.DataFrame(
    {"Character_in_show": ["Arnold", "Gerald", "Helga", "Phoebe", "Harold", "Eugene"],
     "color_of_hair": ["blonde", "black", "blonde", "black", "unknown", "red"],
     "Height": ["average", "tallish", "tallish", "short", "tall", "short"],
     "Football_Shaped_Head": [True, False, False, False, False, False]
     })

hey_arnold
```

Out[2]:

	Character_in_show	Football_Shaped_Head	Height	color_of_hair
0	Arnold	True	average	blonde
1	Gerald	False	tallish	black
2	Helga	False	tallish	blonde
3	Phoebe	False	short	black
4	Harold	False	tall	unknown
5	Eugene	False	short	red

In [3]:
```python
# 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	Football Head	Height	Hair Color
0	Arnold	True	average	blonde
1	Gerald	False	tallish	black
2	Helga	False	tallish	blonde
3	Phoebe	False	short	black
4	Harold	False	tall	unknown
5	Eugene	False	short	red

In [4]:
```python
# 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

In [ ]:

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

In [2]:
```python
# Store filepath in a variable
file_one = "Resources/DataOne.csv"
```

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

In [4]:
```python
# Show just the header
file_one_df.head()
```

Out[4]:

	id	first_name	last_name	email	gender
0	1	David	Jordan	djordan0@home.pl	Male
1	2	Stephen	Riley	sriley1@hugedomains.com	Male
2	3	Evelyn	Grant	egrant2@livejournal.com	Female
3	4	Joe	Mendoza	jmendoza3@un.org	Male
4	5	Benjamin	Rodriguez	brodriguez4@elpais.com	Male

In [5]:
```python
# Show a single column
file_one_df["first_name"].head()
```

Out[5]:
```
0       David
1     Stephen
2      Evelyn
3         Joe
4    Benjamin
Name: first_name, dtype: object
```

In [6]:
```python
# Show mulitple specific columns--note the extra brackets
file_one_df[["first_name", "email"]].head()
```

Out[6]:

	first_name	email
0	David	djordan0@home.pl
1	Stephen	sriley1@hugedomains.com
2	Evelyn	egrant2@livejournal.com
3	Joe	jmendoza3@un.org
4	Benjamin	brodriguez4@elpais.com

In [7]:
```
# Head does not change the DataFrame--it only displays it
file_one_df.head()
```

Out[7]:

	id	first_name	last_name	email	gender
0	1	David	Jordan	djordan0@home.pl	Male
1	2	Stephen	Riley	sriley1@hugedomains.com	Male
2	3	Evelyn	Grant	egrant2@livejournal.com	Female
3	4	Joe	Mendoza	jmendoza3@un.org	Male
4	5	Benjamin	Rodriguez	brodriguez4@elpais.com	Male

In [8]:
```
# Export file as a CSV, without the Pandas index, but with the header
file_one_df.to_csv("Output/fileOne.csv", index=False, header=True)
```

In [1]:

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

In [2]:

```
# Make a reference to the books.csv file path
csv_path = "Resources/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
0	1	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0
1	2	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0
2	3	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	2005.0
3	4	2657	2657	3275794	487	61120081	9.780061e+12	Harper Lee	1960.0
4	5	4671	4671	245494	1356	743273567	9.780743e+12	F. Scott Fitzgerald	1925.0

5 rows × 23 columns

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

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

In [5]:

```
# Push the remade DataFrame to a new CSV file
renamed_df.to_csv("Output/books_clean.csv",
                  encoding="utf-8", index=False, header=True)
```

In [1]:

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

In [2]:

```
# File to Load
goodreads_path = "Resources/books_clean.csv"

# Read the modified GoodReads csv and store into Pandas DataFrame
goodreads_df = pd.read_csv(goodreads_path, encoding="utf-8")
goodreads_df.head()
```

Out[2]:

	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

In [3]:

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

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

# Calculate the total reviews for the entire dataset
# Hint: use the pandas' sum() method to get the sum for each row
goodreads_df['Total Reviews'] = goodreads_df.iloc[:, 4:].sum(axis=1)
total_reviews = sum(goodreads_df['Total Reviews'])
```

In [4]:

```
# 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[4]:

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

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

In [2]:
```
file = "Resources/sampleData.csv"
```

In [3]:
```
df_original = pd.read_csv(file)
df_original.head()
```

Out[3]:

	id	first_name	last_name	Phone Number	Time zone
0	1	Peter	Richardson	7-(789)867-9023	Europe/Moscow
1	2	Janice	Berry	86-(614)973-1727	Asia/Harbin
2	3	Andrea	Hudson	86-(918)527-6371	Asia/Shanghai
3	4	Arthur	Mcdonald	420-(553)779-7783	Europe/Prague
4	5	Kathy	Morales	351-(720)541-2124	Europe/Lisbon

In [4]:
```
# Set new index to last_name
df = df_original.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

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
```

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
```

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

Out[7]:

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

In [8]:
```
# the following logic test/conditional statement returns a series of boolean values
named_billy = df["first_name"] == "Billy"
named_billy.head()
```

Out[8]:
```
last_name
Richardson    False
Berry         False
Hudson        False
Mcdonald      False
Morales       False
Name: first_name, dtype: bool
```

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
```

In [1]:
```python
# Dependencie
import pandas as pd
```

In [2]:
```python
# Load in file
movie_file = "Resources/movie_scores.csv"
```

In [3]:
```python
# Read and display the CSV with Pandas
movie_file_pd = pd.read_csv(movie_file)
movie_file_pd.head()
```

Out[3]:

	FILM	RottenTomatoes	RottenTomatoes_User	Metacritic	Metacritic_User	IMDB	Fandango_Stars	Fandango_Ratingvalue	RT_n
0	Avengers: Age of Ultron (2015)	74	86	66	7.1	7.8	5.0	4.5	(
1	Cinderella (2015)	85	80	67	7.5	7.1	5.0	4.5	(
2	Ant-Man (2015)	80	90	64	8.1	7.8	5.0	4.5	(
3	Do You Believe? (2015)	18	84	22	4.7	5.4	5.0	4.5	(
4	Hot Tub Time Machine 2 (2015)	14	28	29	3.4	5.1	3.5	3.0	(

5 rows × 22 columns

In [4]:
```python
# List all the columns in the table
movie_file_pd.columns
```

Out[4]:
```
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')
```

In [5]:
```python
# 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()
```

Out[5]:

	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

In [6]:
```python
# 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()
```

Out[6]:

	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

In [7]:
```
# 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()
```

Out[7]:

	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

In [8]:
```
# 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("output/movieWatchlist.xlsx", index=False)
```

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

In [2]:
```
# Name of the CSV file
file = 'Resources/donors2008.csv'
```

In [3]:
```
# The correct encoding must be used to read the CSV in pandas
df = pd.read_csv(file, encoding="ISO-8859-1")
```

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

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
```

In [9]:
```
# The Amount column is the wrong data type. It should be numeric.
df.dtypes
```

Out[9]:
```
LastName      object
FirstName     object
Employer      object
City          object
State         object
Zip           object
Amount       float64
dtype: object
```

In [10]:
```
# Use pd.to_numeric() method to convert the datatype of the Amount column
df['Amount'] = pd.to_numeric(df['Amount'])
```

In [11]:
```
# Verify that the Amount column datatype has been made numeric
df['Amount'].dtype
```

Out[11]:
```
dtype('float64')
```

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
Google                                                                          6
Not Employed                                                                    4
Unemployed                                                                      4
Bank Of America                                                                 3
University of California                                                        3
Social Security Administration                                                  3
Davis Polk & Wardwell                                                           2
Berger & Montague                                                               2
Henry Crown & Company                                                           2
Jones Day                                                                       2
Federal Government                                                              2
University Hospital                                                             2
Northern Trust                                                                  2
Newton Public Schools                                                           2
Covington & Burling                                                             2
Microsoft                                                                       2
Ariel Investments                                                               2
CSC                                                                             2
UCLA                                                                            2
LMI                                                                             2
Mayer Brown                                                                     2
Google, Inc.                                                                    2
Freelance                                                                       2
Hugo Neu Corporation                                                            2
Harvard University                                                              2
                                                                             ... 
Alexander & Associates, Inc.                                                    1
Foley And Lardner                                                               1
NY Philharmonic                                                                 1
Advocate Health Care                                                            1
Philip A Greider, MD, Inc                                                       1
Plymouth State University/NH Chapter, National Association of Social Workers    1
The Torrey Funds                                                                1
Rutgers University                                                              1
Red Pen Creative, Inc.                                                          1
Levenson Mcdavid Architects P.C.                                                1
Echo Mountain                                                                   1
Università Bocconi                                                              1
Wayne State University                                                          1
Juniper Networks/Intuit                                                         1
AOL                                                                             1
Perot Systems                                                                   1
World Bank Group: International Finance Corporation                             1
Auburn University                                                               1
Kachina Accounting Services Inc                                                 1
Exxon Mobil Corporation                                                         1
Paddock Publishing, Inc                                                         1
Lewis Rice & Fingersh, LC                                                       1
Newsweb Corporation                                                             1
J.P. Morgan Securities Inc.                                                     1
Retired Dod Civ Servant                                                         1
Lockwood & Hartley, ALC                                                         1
DBD                                                                             1
University of Nevada, Las Vegas                                                 1
Sacramento City Unified School District                                         1
```

```
D Magazine Partners                                                  1
Name: Employer, Length: 1011, dtype: int64
```

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
Unemployed                                                                     4
Not Employed                                                                   4
University of California                                                       3
Bank Of America                                                                3
Social Security Administration                                                 3
Microsoft                                                                      2
Jones Day                                                                      2
Hugo Neu Corporation                                                           2
Freelance                                                                      2
Ariel Investments                                                              2
Henry Crown & Company                                                          2
Newton Public Schools                                                          2
Federal Government                                                             2
Berger & Montague                                                              2
LMI                                                                            2
University Hospital                                                            2
Covington & Burling                                                            2
Mayer Brown                                                                    2
CSC                                                                            2
Davis Polk & Wardwell                                                          2
Google, Inc.                                                                   2
Northern Trust                                                                 2
United Health Group                                                            2
Harvard University                                                             2
Sidley Austin LLP                                                              2
Rainey Cluss LLC                                                               2
                                                                            ...
Alexander & Associates, Inc.                                                   1
Foley And Lardner                                                              1
NY Philharmonic                                                                1
Advocate Health Care                                                           1
Philip A Greider, MD, Inc                                                      1
Plymouth State University/NH Chapter, National Association of Social Workers    1
The Torrey Funds                                                               1
Rutgers University                                                             1
Red Pen Creative, Inc.                                                         1
Levenson Mcdavid Architects P.C.                                               1
Echo Mountain                                                                  1
Università Bocconi                                                             1
Wayne State University                                                         1
Juniper Networks/Intuit                                                        1
AOL                                                                            1
Perot Systems                                                                  1
World Bank Group: International Finance Corporation                            1
Auburn University                                                              1
Kachina Accounting Services Inc                                                1
Exxon Mobil Corporation                                                        1
Paddock Publishing, Inc                                                        1
Lewis Rice & Fingersh, LC                                                      1
Newsweb Corporation                                                            1
J.P. Morgan Securities Inc.                                                    1
Retired Dod Civ Servant                                                        1
Lockwood & Hartley, ALC                                                        1
DBD                                                                            1
University of Nevada, Las Vegas                                                1
Sacramento City Unified School District                                        1
```

```
D Magazine Partners                                                     1
Name: Employer, Length: 1009, dtype: int64
```

In [15]:
```
df['Employer'] = df['Employer'].replace({'Not Employed': 'Unemployed'})
df['Employer'].value_counts()
```

Out[15]:
```
Self-Employed                                                          314
None                                                                   249
Retired                                                                126
Unemployed                                                               8
Google                                                                   6
University of California                                                 3
Social Security Administration                                           3
Bank Of America                                                          3
Henry Crown & Company                                                    2
Federal Government                                                       2
Berger & Montague                                                        2
Jones Day                                                                2
University Hospital                                                      2
Covington & Burling                                                      2
Freelance                                                                2
Google, Inc.                                                             2
LMI                                                                      2
Davis Polk & Wardwell                                                    2
Microsoft                                                                2
Mayer Brown                                                              2
Northern Trust                                                           2
Ariel Investments                                                        2
Newton Public Schools                                                    2
CSC                                                                      2
Hugo Neu Corporation                                                     2
ExxonMobil                                                               2
UCLA                                                                     2
Rainey Cluss LLC                                                         2
Harvard University                                                       2
University Of Michigan                                                   2
                                                                      ...
Alexander & Associates, Inc.                                             1
Foley And Lardner                                                        1
NY Philharmonic                                                          1
Advocate Health Care                                                     1
Philip A Greider, MD, Inc                                                1
Plymouth State University/NH Chapter, National Association of Social Workers     1
The Torrey Funds                                                         1
Rutgers University                                                       1
Red Pen Creative, Inc.                                                   1
Levenson Mcdavid Architects P.C.                                         1
Echo Mountain                                                            1
Università Bocconi                                                       1
Wayne State University                                                   1
Juniper Networks/Intuit                                                  1
AOL                                                                      1
Perot Systems                                                            1
World Bank Group: International Finance Corporation                      1
Auburn University                                                        1
Kachina Accounting Services Inc                                          1
Exxon Mobil Corporation                                                  1
Paddock Publishing, Inc                                                  1
Lewis Rice & Fingersh, LC                                                1
Newsweb Corporation                                                      1
J.P. Morgan Securities Inc.                                              1
Retired Dod Civ Servant                                                  1
Lockwood & Hartley, ALC                                                  1
DBD                                                                      1
University of Nevada, Las Vegas                                          1
Sacramento City Unified School District                                  1
```

```
D Magazine Partners                                                              1
Name: Employer, Length: 1008, dtype: int64
```

In [16]:

```
# Display a statistical overview
# We can infer the maximum allowable individual contribution from 'max'
df.describe()
```

Out[16]:

	Amount
count	1743.000000
mean	640.124750
std	1242.343265
min	5.000000
25%	200.000000
50%	250.000000
75%	500.000000
max	5000.000000

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

In [2]:
```python
# Reference the file where the CSV is located
crime_csv_path = "Resources/crime_incident_data2017.csv"

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

Out[2]:

	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	Open Data X	Open Data Y	Report Date	Report Month Year
0	NaN	17-X4762181	Person	NaN	1	1/1/96	1/1/96	800	Sex Offenses	1	Rape	NaN	NaN	NaN	NaN	1/26/17	1/1/17
1	NaN	17-X4757824	Property	Centennial	1	1/20/00	1/1/00	1615	Fraud Offenses	1	Identity Theft	NaN	NaN	NaN	NaN	1/20/17	1/1/17
2	200 BLOCK OF SE 78TH AVE	17-900367	Property	Montavilla	1	12/1/03	12/1/03	800	Fraud Offenses	1	False Pretenses/Swindle/Confidence Game	45.5207	-122.583	7668150.0	682825.0	1/9/17	1/1/17
3	NaN	17-X4748982	Property	Southwest Hills	1	1/1/10	1/1/10	0	Fraud Offenses	1	Identity Theft	NaN	NaN	NaN	NaN	1/5/17	1/1/17
4	NaN	17-X4748982	Property	Southwest Hills	1	1/1/10	1/1/10	0	Larceny Offenses	1	All Other Larceny	NaN	NaN	NaN	NaN	1/5/17	1/1/17
5	5400 BLOCK OF NE MALLORY AVE	17-900129	Property	King	1	11/28/10	11/1/10	1612	Fraud Offenses	1	Identity Theft	45.5625	-122.664	7647987.0	698581.0	1/3/17	1/1/17
	5000 BLOCK OF	17-							Fraud		False						

In [3]:
```python
# look for missing values
crime_df.count()
```

Out[3]:
```
Address              37365
Case Number          41032
Crime Against        41032
Neighborhood         39712
Number of Records    41032
Occur Date           41032
Occur Month Year     41032
Occur Time           41032
Offense Category     41032
Offense Count        41032
Offense Type         41032
Open Data Lat        36712
Open Data Lon        36712
Open Data X          36712
Open Data Y          36712
Report Date          41032
Report Month Year    41032
dtype: int64
```

In [4]:
```python
# drop null rows
no_null_crime_df = crime_df.dropna(how='any')
```

In [5]:
```
# verify counts
no_null_crime_df.count()
```

Out[5]:
```
Address              36146
Case Number          36146
Crime Against        36146
Neighborhood         36146
Number of Records    36146
Occur Date           36146
Occur Month Year     36146
Occur Time           36146
Offense Category     36146
Offense Count        36146
Offense Type         36146
Open Data Lat        36146
Open Data Lon        36146
Open Data X          36146
Open Data Y          36146
Report Date          36146
Report Month Year    36146
dtype: int64
```

In [6]:
```
# Check to see if there are any values with mispelled or similar values in "Offense Type"
no_null_crime_df["Offense Type"].value_counts()
```

```
Counterfeiting/Forgery                          448
Weapons Law Violations                          266
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
```

In [7]:

```
# 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
```

Out[7]:

	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	Open Data X	Open Data Y	Report Date	Report Month Year
2	200 BLOCK OF SE 78TH AVE	17-900367	Property	Montavilla	1	12/1/03	12/1/03	800	Fraud Offenses	1	False Pretenses/Swindle/Confidence Game	45.5207	-122.583	7668150.0	682825.0	1/9/17	1/1/17
5	5400 BLOCK OF NE MALLORY AVE	17-900129	Property	King	1	11/28/10	11/1/10	1612	Fraud Offenses	1	Identity Theft	45.5625	-122.664	7647987.0	698581.0	1/3/17	1/1/17
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	7652567.0	697337.0	1/26/17	1/1/17
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	7652567.0	697337.0	1/26/17	1/1/17
8	12000 BLOCK OF SE PINE ST	17-900253	Property	Hazelwood	1	1/6/14	1/1/14	805	Fraud Offenses	1	Credit Card/ATM Fraud	45.5204	-122.539	7679522.0	682404.0	1/6/17	1/1/17
9	12000 BLOCK OF SE PINE ST	17-900253	Property	Hazelwood	1	1/6/14	1/1/14	805	Fraud Offenses	1	Identity Theft	45.5204	-122.539	7679522.0	682404.0	1/6/17	1/1/17

In [8]:

```
# Check to see if you comnbined similar offenses correctly in "Offense Type".
no_null_crime_df["Offense Type"].value_counts()
```

Out[8]:

```
Theft From Motor Vehicle                        6947
Motor Vehicle Theft                             4689
All Other Larceny                               4558
Vandalism                                       3863
Burglary                                        2824
Shoplifting                                     2259
Identity Theft                                  1794
Simple Assault                                  1216
Drug/Narcotic Violations                        1095
Theft of Motor Vehicle Parts or Accessories     1073
Intimidation                                     900
Theft From Building                              895
False Pretenses/Swindle/Confidence Game          870
Aggravated Assault                               839
Robbery                                          608
Counterfeiting/Forgery                           448
Weapons Law Violations                           266
Credit Card/ATM Fraud                            226
Arson                                            200
Prostitution                                     153
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
Drug Equipment Violations                          6
Impersonation                                      4
Wire Fraud                                         3
Welfare Fraud                                      1
Name: Offense Type, dtype: int64
```

In [9]:
```python
# 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
```

Out[9]:

	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	Open Data X	Open Data Y	Report Date	Report Month Year
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	7652567.0	697337.0	1/26/17	1/1/17
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	7652567.0	697337.0	1/26/17	1/1/17
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	7650320.0	698297.0	1/29/17	1/1/17
148	1000 BLOCK OF NE EMERSON ST	17-901190	Property	Vernon	1	11/26/16	11/1/16	2040	Larceny Offenses	1	All Other Larceny	45.5619	-122.655	7650320.0	698297.0	1/29/17	1/1/17
271	5300 BLOCK OF NE 14TH PL	17-2593	Property	Vernon	1	12/19/16	12/1/16	900	Larceny Offenses	1	All Other Larceny	45.5618	-122.651	7651314.0	698264.0	1/3/17	1/1/17
572	5400 BLOCK OF NE 13TH AVE	17-900012	Property	Vernon	1	1/1/17	1/1/17	725	Larceny Offenses	1	Theft From Motor Vehicle	45.5625	-122.652	7650993.0	698515.0	1/1/17	1/1/17
742	5700 BLOCK OF	17-2395	Property	Vernon	1	1/2/17	1/1/17	2000	Vandalism	1	Vandalism	45.5643	-122.653	7650716.0	699162.0	1/3/17	1/1/17

In [ ]:

In [ ]:

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

In [2]:
```python
# Reference the file where the CSV is located
crime_csv_path = "Resources/crime_incident_data2017.csv"

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

Out[2]:

	Address	Case Number	Crime Against	Neighborhood	Number of Records	Occur Date	Occur Month Year	Occur Time	Offense Category	Offense Count	
0	NaN	17-X4762181	Person	NaN	1	1/1/96	1/1/96	800	Sex Offenses	1	
1	NaN	17-X4757824	Property	Centennial	1	1/20/00	1/1/00	1615	Fraud Offenses	1	
2	200 BLOCK OF SE 78TH AVE	17-900367	Property	Montavilla	1	12/1/03	12/1/03	800	Fraud Offenses	1	Pret
3	NaN	17-X4748982	Property	Southwest Hills	1	1/1/10	1/1/10	0	Fraud Offenses	1	
4	NaN	17-X4748982	Property	Southwest Hills	1	1/1/10	1/1/10	0	Larceny Offenses	1	
5	5400 BLOCK OF NE MALLORY	17-900120	Property	King	1	11/28/10	11/1/10	1612	Fraud Offenses	1	

In [3]:
```python
# look for missing values
crime_df.count()
```

Out[3]:
```
Address              37365
Case Number          41032
Crime Against        41032
Neighborhood         39712
Number of Records    41032
Occur Date           41032
Occur Month Year     41032
Occur Time           41032
Offense Category     41032
Offense Count        41032
Offense Type         41032
Open Data Lat        36712
Open Data Lon        36712
Open Data X          36712
Open Data Y          36712
Report Date          41032
Report Month Year    41032
dtype: int64
```

In [4]:
```python
# drop null rows
no_null_crime_df = crime_df.dropna(how='any')
```

In [5]:
```python
# verify counts
no_null_crime_df.count()
```

Out[5]:
```
Address              36146
Case Number          36146
Crime Against        36146
Neighborhood         36146
Number of Records    36146
Occur Date           36146
Occur Month Year     36146
Occur Time           36146
Offense Category     36146
Offense Count        36146
Offense Type         36146
Open Data Lat        36146
Open Data Lon        36146
Open Data X          36146
Open Data Y          36146
Report Date          36146
Report Month Year    36146
dtype: int64
```

In [6]:
```python
# Check to see if there are any values with mispelled or similar values in "Offense Type"
no_null_crime_df["Offense Type"].value_counts()
```

```
Counterfeiting/Forgery                          448
Weapons Law Violations                          266
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
```

In [7]:
```python
# 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
```

Out[7]:

	Address	Case Number	Crime Against	Neighborhood	Number of Records	Occur Date	Occur Month Year	Occur Time	Offense Category	Offense Count	
2	200 BLOCK OF SE 78TH AVE	17-900367	Property	Montavilla	1	12/1/03	12/1/03	800	Fraud Offenses	1	Pret
5	5400 BLOCK OF NE MALLORY AVE	17-900129	Property	King	1	11/28/10	11/1/10	1612	Fraud Offenses	1	
6	5000 BLOCK OF NE 19TH AVE	17-901079	Property	Vernon	1	11/8/13	11/1/13	1200	Fraud Offenses	1	Pret
7	5000 BLOCK OF NE 19TH AVE	17-901079	Property	Vernon	1	11/8/13	11/1/13	1200	Fraud Offenses	1	
	12000 BLOCK	17-							Fraud		

In [8]:
```python
# Check to see if you comnbined similar offenses correctly in "Offense Type".
no_null_crime_df["Offense Type"].value_counts()
```

Out[8]:
```
Theft From Motor Vehicle                        6947
Motor Vehicle Theft                             4689
All Other Larceny                               4558
Vandalism                                       3863
Burglary                                        2824
Shoplifting                                     2259
Identity Theft                                  1794
Simple Assault                                  1216
Drug/Narcotic Violations                        1095
Theft of Motor Vehicle Parts or Accessories     1073
Intimidation                                     900
Theft From Building                              895
False Pretenses/Swindle/Confidence Game          870
Aggravated Assault                               839
Robbery                                          608
Counterfeiting/Forgery                           448
Weapons Law Violations                           266
Credit Card/ATM Fraud                            226
Arson                                            200
Prostitution                                     153
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
Drug Equipment Violations                          6
Impersonation                                      4
Wire Fraud                                         3
Welfare Fraud                                      1
Name: Offense Type, dtype: int64
```

In [9]:
```python
# 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
```

Out[9]:

	Address	Case Number	Crime Against	Neighborhood	Number of Records	Occur Date	Occur Month Year	Occur Time	Offense Category	Offer Cou
6	5000 BLOCK OF NE 19TH AVE	17-901079	Property	Vernon	1	11/8/13	11/1/13	1200	Fraud Offenses	
7	5000 BLOCK OF NE 19TH AVE	17-901079	Property	Vernon	1	11/8/13	11/1/13	1200	Fraud Offenses	
147	1000 BLOCK OF NE EMERSON ST	17-901190	Property	Vernon	1	11/26/16	11/1/16	2040	Fraud Offenses	
148	1000 BLOCK OF NE EMERSON ST	17-901190	Property	Vernon	1	11/26/16	11/1/16	2040	Larceny Offenses	
271	5300 BLOCK OF NE 14TH PL	17-2593	Property	Vernon	1	12/19/16	12/1/16	900	Larceny Offenses	
572	5400 BLOCK OF NE 13TH AVE	17-900012	Property	Vernon	1	1/1/17	1/1/17	725	Larceny Offenses	

In [ ]:

In [ ]:

In [1]:
```python
# Import the Pandas library
import pandas as pd
```

In [2]:
```python
# 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()
```

```
/Users/arwenshackelford/anaconda3/envs/dev/lib/python3.6/site-packages/IPython/core/interactiveshell.p
y:2714: DtypeWarning: Columns (5,9) have mixed types. Specify dtype option on import or set low_memory=
False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	datetime	city	state	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700	45 minutes	This event took place in early fall around 194...	4/27/2004	29.8830556	-97.941111
1	10/10/1949 21:00	lackland afb	tx	NaN	light	7200	1-2 hrs	1949 Lackland AFB&#44 TX. Lights racing acros...	12/16/2005	29.38421	-98.581082
2	10/10/1955 17:00	chester (uk/england)	NaN	gb	circle	20	20 seconds	Green/Orange circular disc over Chester&#44 En...	1/21/2008	53.2	-2.916667
3	10/10/1956 21:00	edna	tx	us	circle	20	1/2 hour	My older brother and twin sister were leaving ...	1/17/2004	28.9783333	-96.645833
4	10/10/1960 20:00	kaneohe	hi	us	light	900	15 minutes	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004	21.4180556	-157.803611

In [3]:
```python
# Check to see if there are any rows with missing data
ufo_df.count()
```

Out[3]:
```
datetime                80332
city                    80332
state                   74535
country                 70662
shape                   78400
duration (seconds)      80332
duration (hours/min)    80332
comments                80317
date posted             80332
latitude                80332
longitude               80332
dtype: int64
```

In [4]:
```
# Remove the rows with missing data
clean_ufo_df = ufo_df.dropna(how="any")
clean_ufo_df.count()
```

Out[4]:
```
datetime                  66516
city                      66516
state                     66516
country                   66516
shape                     66516
duration (seconds)        66516
duration (hours/min)      66516
comments                  66516
date posted               66516
latitude                  66516
longitude                 66516
dtype: int64
```

In [5]:
```
# 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()
```

Out[5]:

	datetime	city	state	country	shape	duration (seconds)	duration (hours/min)	comments	date posted
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700	45 minutes	This event took place in early fall around 194...	4/27/2004
3	10/10/1956 21:00	edna	tx	us	circle	20	1/2 hour	My older brother and twin sister were leaving ...	1/17/2004
4	10/10/1960 20:00	kaneohe	hi	us	light	900	15 minutes	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004
5	10/10/1961 19:00	bristol	tn	us	sphere	300	5 minutes	My father is now 89 my brother 52 the girl wit...	4/27/2007
7	10/10/1965 23:45	norwalk	ct	us	disk	1200	20 minutes	A bright orange color changing to reddish colo...	10/2/1999

In [6]:
```
# Count how many sightings have occured within each state
state_counts = usa_ufo_df["state"].value_counts()
state_counts
```

Out[6]:
```
ca    8683
fl    3754
wa    3707
tx    3398
ny    2915
il    2447
az    2362
pa    2319
oh    2251
mi    1781
nc    1722
or    1667
mo    1431
co    1385
in    1268
va    1248
ma    1238
nj    1236
ga    1235
wi    1205
tn    1091
mn     996
sc     986
ct     865
ky     843
md     818
nv     778
ok     714
nm     693
ia     669
al     629
ut     611
ks     599
ar     578
la     547
me     544
id     508
nh     482
mt     460
wv     438
ne     373
ms     368
ak     311
hi     257
vt     254
ri     224
sd     177
wy     169
de     165
nd     123
pr      24
dc       7
Name: state, dtype: int64
```

In [7]:
```
# Convert the state_counts Series into a DataFrame
state_ufo_counts_df = pd.DataFrame(state_counts)
state_ufo_counts_df.head()
```

Out[7]:

	state
ca	8683
fl	3754
wa	3707
tx	3398
ny	2915

In [8]:
```
# 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()
```

Out[8]:

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

In [9]:
```
# 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
```

Out[9]:
```
datetime                object
city                    object
state                   object
country                 object
shape                   object
duration (seconds)      object
duration (hours/min)    object
comments                object
date posted             object
dtype: object
```

In [10]:
```
# 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
```

Out[10]:
```
datetime                 object
city                     object
state                    object
country                  object
shape                    object
duration (seconds)      float64
duration (hours/min)     object
comments                 object
date posted              object
dtype: object
```

In [11]:
```
# Now it is possible to find the sum of seconds
usa_ufo_df["duration (seconds)"].sum()
```

Out[11]: 351281285.38

In [ ]:

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

In [2]:
```
# 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()
```

```
/Users/arwenshackelford/anaconda3/envs/dev/lib/python3.6/site-packages/IPython/core/interactiveshel
l.py:2714: DtypeWarning: Columns (5,9) have mixed types. Specify dtype option on import or set low_m
emory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	datetime	city	state	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitud
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700	45 minutes	This event took place in early fall around 194...	4/27/2004	29.8830556	-97.94111
1	10/10/1949 21:00	lackland afb	tx	NaN	light	7200	1-2 hrs	1949 Lackland AFB&#44 TX. Lights racing acros...	12/16/2005	29.38421	-98.58108
2	10/10/1955 17:00	chester (uk/england)	NaN	gb	circle	20	20 seconds	Green/Orange circular disc over Chester&#44 En...	1/21/2008	53.2	-2.91666
3	10/10/1956 21:00	edna	tx	us	circle	20	1/2 hour	My older brother and twin sister were leaving ...	1/17/2004	28.9783333	-96.64583
4	10/10/1960 20:00	kaneohe	hi	us	light	900	15 minutes	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004	21.4180556	-157.80361

In [3]:
```
# Remove the rows with missing data
clean_ufo_df = ufo_df.dropna(how="any")
clean_ufo_df.count()
```

Out[3]:
```
datetime                66516
city                    66516
state                   66516
country                 66516
shape                   66516
duration (seconds)      66516
duration (hours/min)    66516
comments                66516
date posted             66516
latitude                66516
longitude               66516
dtype: int64
```

In [4]:
```
clean_ufo_df.head()
```

Out[4]:

	datetime	city	state	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700	45 minutes	This event took place in early fall around 194...	4/27/2004	29.8830556	-97.941111
3	10/10/1956 21:00	edna	tx	us	circle	20	1/2 hour	My older brother and twin sister were leaving ...	1/17/2004	28.9783333	-96.645833
4	10/10/1960 20:00	kaneohe	hi	us	light	900	15 minutes	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004	21.4180556	-157.803611
5	10/10/1961 19:00	bristol	tn	us	sphere	300	5 minutes	My father is now 89 my brother 52 the girl wit...	4/27/2007	36.595	-82.188889
7	10/10/1965 23:45	norwalk	ct	us	disk	1200	20 minutes	A bright orange color changing to reddish colo...	10/2/1999	41.1175	-73.408333

In [5]:
```
# Converting the "duration (seconds)" column's values to numeric
converted_ufo = clean_ufo_df.copy()
converted_ufo["duration (seconds)"] = converted_ufo.loc[:, "duration (seconds)"].astype(float)
```

In [6]:
```
converted_ufo.head()
```

Out[6]:

	datetime	city	state	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700.0	45 minutes	This event took place in early fall around 194...	4/27/2004	29.8830556	-97.941111
3	10/10/1956 21:00	edna	tx	us	circle	20.0	1/2 hour	My older brother and twin sister were leaving ...	1/17/2004	28.9783333	-96.645833
4	10/10/1960 20:00	kaneohe	hi	us	light	900.0	15 minutes	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004	21.4180556	-157.803611
5	10/10/1961 19:00	bristol	tn	us	sphere	300.0	5 minutes	My father is now 89 my brother 52 the girl wit...	4/27/2007	36.595	-82.188889
7	10/10/1965 23:45	norwalk	ct	us	disk	1200.0	20 minutes	A bright orange color changing to reddish colo...	10/2/1999	41.1175	-73.408333

In [7]:
```
# Filter the data so that only those sightings in the US are in a DataFrame
usa_ufo_df = converted_ufo.loc[converted_ufo["country"] == "us", :]
usa_ufo_df.head()
```

Out[7]:

	datetime	city	state	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
0	10/10/1949 20:30	san marcos	tx	us	cylinder	2700.0	45 minutes	This event took place in early fall around 194...	4/27/2004	29.8830556	-97.941111
3	10/10/1956 21:00	edna	tx	us	circle	20.0	1/2 hour	My older brother and twin sister were leaving ...	1/17/2004	28.9783333	-96.645833
4	10/10/1960 20:00	kaneohe	hi	us	light	900.0	15 minutes	AS a Marine 1st Lt. flying an FJ4B fighter/att...	1/22/2004	21.4180556	-157.803611
5	10/10/1961 19:00	bristol	tn	us	sphere	300.0	5 minutes	My father is now 89 my brother 52 the girl wit...	4/27/2007	36.595	-82.188889
7	10/10/1965 23:45	norwalk	ct	us	disk	1200.0	20 minutes	A bright orange color changing to reddish colo...	10/2/1999	41.1175	-73.408333

In [8]:
```
# Count how many sightings have occured within each state
state_counts = usa_ufo_df["state"].value_counts()
state_counts.head()
```

Out[8]:
```
ca    8683
fl    3754
wa    3707
tx    3398
ny    2915
Name: state, dtype: int64
```

In [9]:
```python
# 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

In [10]:
```
grouped_usa_df["duration (seconds)"].sum()
```

Out[10]:
```
state
ak     1455863.00
al      900453.50
ar    66986144.50
az    15453494.60
ca    24865571.47
co     1923709.00
ct     3110318.80
dc        1645.50
de      142969.50
fl    55900005.00
ga     9519878.10
hi     6732485.00
ia      613576.00
id      475270.30
il     2133923.07
in     4032395.70
ks      830518.50
ky     3435497.50
la     6819072.00
ma     1602861.00
md      688074.30
me      654476.90
mi     1895119.10
mn     1382802.33
mo     1614738.80
ms     3396695.00
mt     1050599.00
nc     2056718.35
nd      140274.00
ne      412354.00
nh     1072798.50
nj     7784974.00
nm     4055283.59
nv     2393413.95
ny     8898149.55
oh     3284932.80
ok      853112.30
or     1774625.28
pa     9110355.00
pr       26200.00
ri      472900.50
sc     1089566.80
sd      480358.50
tn     1854526.30
tx     8444239.25
ut     3417964.00
va    13606781.00
vt      264785.50
wa    56618769.44
wi     2323749.30
wv     2974853.00
wy      251443.00
Name: duration (seconds), dtype: float64
```

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_table = pd.DataFrame({"Number of Sightings": state_counts,
                                    "Total Visit Time": state_duration})
state_summary_table.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

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]:

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

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

Out[14]:

country	state	duration (seconds)
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

In [ ]:

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

In [2]:
```python
# Save file path to variable
pokemon_csv = "Resources/Pokemon.csv"
```

In [3]:
```python
# Read with Pandas
pokemon_pd = pd.read_csv(pokemon_csv)
pokemon_pd.head()
```

Out[3]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

In [4]:
```python
# 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()
```

Out[4]:

	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

In [5]:
```python
# 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
```

Out[5]:

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
Poison	67.250000	74.678571	68.821429	60.428571	64.392857	63.571429
Psychic	70.631579	71.456140	67.684211	98.403509	86.280702	81.491228
Rock	65.363636	92.863636	100.795455	63.340909	75.477273	55.909091
Steel	65.222222	92.703704	126.370370	67.518519	80.629630	55.259259
Water	72.062500	74.151786	72.946429	74.812500	70.517857	65.964286

In [6]:
```python
# 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"]
```

Out[6]:
```
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
```

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

In [2]:
```python
csv_path = "Resources/Happiness_2017.csv"
happiness_df = pd.read_csv(csv_path)
happiness_df.head()
```

Out[2]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom	Generosity	Trust..Government.Corruption.	Dystopia.Residual
0	Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.796667	0.635423	0.362012	0.315964	2.277027
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007	0.355280	0.400770	2.313707
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163	0.475540	0.153527	2.322715
3	Switzerland	4	7.494	7.561772	7.426227	1.564980	1.516912	0.858131	0.620071	0.290549	0.367007	2.276716
4	Finland	5	7.469	7.527542	7.410458	1.443572	1.540247	0.809158	0.617951	0.245483	0.382612	2.430182

In [3]:
```python
# 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..Life.Expectancy.	Freedom	Generosity	Trust..Government.Corruption.	Dystopia.Residual
139	Angola	140	3.795	3.951642	3.638358	0.858428	1.104412	0.049869	0.000000	0.097926	0.069720	1.614482
129	Sudan	130	4.139	4.345747	3.932253	0.659517	1.214009	0.290921	0.014996	0.182317	0.089848	1.687066
144	Haiti	145	3.603	3.734715	3.471285	0.368610	0.640450	0.277321	0.030370	0.489204	0.099872	1.697168
153	Burundi	154	2.905	3.074690	2.735310	0.091623	0.629794	0.151611	0.059901	0.204435	0.084148	1.683024
151	Syria	152	3.462	3.663669	3.260331	0.777153	0.396103	0.500533	0.081539	0.493664	0.151347	1.061574

In [4]:
```python
# 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..Life.Expectancy.	Freedom	Generosity	Trust..Government.Corruption.	Dystopia.Residual
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498273	0.658249	0.415984	0.246528	1.816914
0	Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.796667	0.635423	0.362012	0.315964	2.277027
128	Cambodia	129	4.168	4.278518	4.057483	0.601765	1.006238	0.429783	0.633376	0.385923	0.068106	1.042941
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163	0.475540	0.153527	2.322715
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007	0.355280	0.400770	2.313707

In [5]:
```python
# It is possible to sort based upon multiple columns
family_and_generosity = happiness_df.sort_values(
    ["Family", "Generosity"], ascending=False)
family_and_generosity.head()
```

Out[5]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom	Generosity	Trust..Government.Corruption.	Dystopia.Residual
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163	0.475540	0.153527	2.322715
14	Ireland	15	6.977	7.043352	6.910649	1.535707	1.558231	0.809783	0.573110	0.427858	0.298388	1.773869
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007	0.355280	0.400770	2.313707
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498273	0.658249	0.415984	0.246528	1.816914
7	New Zealand	8	7.314	7.379510	7.248490	1.405706	1.548195	0.816760	0.614062	0.500005	0.382817	2.046456

In [6]:
```
# 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()
```

Out[6]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy.	Freedom	Generosity	Trust..Government.Corruption.	Dystopia.Residual
0	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833552	0.627163	0.475540	0.153527	2.322715
1	Ireland	15	6.977	7.043352	6.910649	1.535707	1.558231	0.809783	0.573110	0.427858	0.298388	1.773869
2	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792566	0.626007	0.355280	0.400770	2.313707
3	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498273	0.658249	0.415984	0.246528	1.816914
4	New Zealand	8	7.314	7.379510	7.248490	1.405706	1.548195	0.816760	0.614062	0.500005	0.382817	2.046456

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

In [2]:
```
csv_path = "Resources/Happiness_2017.csv"
happiness_df = pd.read_csv(csv_path)
happiness_df.head()
```

Out[2]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectanc
0	Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.79666
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.79256
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.83355
3	Switzerland	4	7.494	7.561772	7.426227	1.564980	1.516912	0.85813
4	Finland	5	7.469	7.527542	7.410458	1.443572	1.540247	0.80915

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..Life.Expectancy
139	Angola	140	3.795	3.951642	3.638358	0.858428	1.104412	0.04986
129	Sudan	130	4.139	4.345747	3.932253	0.659517	1.214009	0.29092
144	Haiti	145	3.603	3.734715	3.471285	0.368610	0.640450	0.27732
153	Burundi	154	2.905	3.074690	2.735310	0.091623	0.629794	0.15161
151	Syria	152	3.462	3.663669	3.260331	0.777153	0.396103	0.50053

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..Life.Expecta
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.498
0	Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.796
128	Cambodia	129	4.168	4.278518	4.057483	0.601765	1.006238	0.429
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.833
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.792

In [5]:
```
# It is possible to sort based upon multiple columns
family_and_generosity = happiness_df.sort_values(
    ["Family", "Generosity"], ascending=False)
family_and_generosity.head()
```

Out[5]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectan
2	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.8335
14	Ireland	15	6.977	7.043352	6.910649	1.535707	1.558231	0.8097
1	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.7925
46	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.4982
7	New Zealand	8	7.314	7.379510	7.248490	1.405706	1.548195	0.8167

In [6]:
```
# 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()
```

Out[6]:

	Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy..GDP.per.Capita.	Family	Health..Life.Expectancy
0	Iceland	3	7.504	7.622030	7.385970	1.480633	1.610574	0.83355
1	Ireland	15	6.977	7.043352	6.910649	1.535707	1.558231	0.80978
2	Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.79256
3	Uzbekistan	47	5.971	6.065538	5.876463	0.786441	1.548969	0.49827
4	New Zealand	8	7.314	7.379510	7.248490	1.405706	1.548195	0.81676

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

In [2]:
```python
# Create reference to CSV file
csv_path = "Resources/Soccer2018Data.csv"

# Import the CSV into a pandas DataFrame
soccer_2018_df = pd.read_csv(csv_path, low_memory=False)
soccer_2018_df
```

Out[2]:

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM	RF	RM	RS	RW	RWB	ST
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	91.0	66.0	92.0
1	L. Messi	30	Argentina	93	93	FC Barcelona	RW	92.0	45.0	59.0	...	57.0	45.0	84.0	59.0	92.0	90.0	88.0	91.0	62.0	88.0
2	Neymar	25	Brazil	92	94	Paris Saint-Germain	LW	88.0	46.0	59.0	...	59.0	46.0	79.0	59.0	88.0	87.0	84.0	89.0	64.0	84.0
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	87.0	68.0	88.0
4	M. Neuer	31	Germany	92	92	FC Bayern Munich	GK	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
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	84.0	61.0	88.0
6	De Gea	26	Spain	90	92	Manchester United	GK	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	E. Hazard	26	Belgium	90	91	Chelsea	LW	88.0	47.0	61.0	...	59.0	47.0	81.0	61.0	87.0	87.0	82.0	88.0	64.0	82.0

In [3]:
```python
# Collect a list of all the unique values in "Preferred Position"
soccer_2018_df["Preferred Position"].unique()
```

Out[3]:
```
array(['ST', 'RW', 'LW', 'GK', 'CDM', 'CB', 'RM', 'CM', 'LM', 'LB', 'CAM',
       'RB', 'CF', 'RWB', 'LWB'], dtype=object)
```

In [4]:
```
# Looking only at strikers (ST) to start
strikers_2018_df = soccer_2018_df.loc[soccer_2018_df["Preferred Position"] == "ST", :]
strikers_2018_df.head()
```

Out[4]:

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM	RF	RM	RS	RW	RWB	ST
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	91.0	66.0	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	87.0	68.0	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	84.0	61.0	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	82.0	55.0	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	86.0	57.0	86.0

5 rows × 33 columns

In [5]:
```
# 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()
```

Out[5]:

	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

In [6]:
```
# Save all of the information collected on the worst striker
worst_striker = strikers_2018_df.loc[0, :]
worst_striker
```

Out[6]:
```
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
LS                                 31
LW                                 29
LWB                                38
RAM                                29
RB                                 40
RCB                                45
RCM                                30
RDM                                38
RF                                 29
RM                                 30
RS                                 31
RW                                 29
RWB                                38
ST                                 31
Name: 0, dtype: object
```

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

In [2]:
```python
# Create reference to CSV file
csv_path = "Resources/Soccer2018Data.csv"

# Import the CSV into a pandas DataFrame
soccer_2018_df = pd.read_csv(csv_path, low_memory=False)
soccer_2018_df
```

Out[2]:

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM
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
1	L. Messi	30	Argentina	93	93	FC Barcelona	RW	92.0	45.0	59.0	...	57.0	45.0	84.0	59.0
2	Neymar	25	Brazil	92	94	Paris Saint-Germain	LW	88.0	46.0	59.0	...	59.0	46.0	79.0	59.0
3	L. Suárez	30	Uruguay	92	92	FC Barcelona	ST	87.0	58.0	65.0	...	64.0	58.0	80.0	65.0
4	M. Neuer	31	Germany	92	92	FC Bayern Munich	GK	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
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
6	De Gea	26	Spain	90	92	Manchester United	GK	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

In [3]:
```python
# Collect a list of all the unique values in "Preferred Position"
soccer_2018_df["Preferred Position"].unique()
```

Out[3]:
```
array(['ST', 'RW', 'LW', 'GK', 'CDM', 'CB', 'RM', 'CM', 'LM', 'LB', 'CAM',
       'RB', 'CF', 'RWB', 'LWB'], dtype=object)
```

In [4]:
```python
# Looking only at strikers (ST) to start
strikers_2018_df = soccer_2018_df.loc[soccer_2018_df["Preferred Position"] == "ST", :]
strikers_2018_df.head()
```

Out[4]:

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM	RF	RM
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
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
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
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
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

5 rows × 33 columns

In [5]:
```
# 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()
```

Out[5]:

	Name	Age	Nationality	Overall	Potential	Club	Preferred Position	CAM	CB	CDM	...	RB	RCB	RCM	RDM	RF	RM	I
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	3
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	4
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	4
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	4
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	4

5 rows × 33 columns

In [6]:
```
# Save all of the information collected on the worst striker
worst_striker = strikers_2018_df.loc[0, :]
worst_striker
```

Out[6]:
```
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
LS                               31
LW                               29
LWB                              38
RAM                              29
RB                               40
RCB                              45
RCM                              30
RDM                              38
RF                               29
RM                               30
RS                               31
RW                               29
RWB                              38
ST                               31
Name: 0, dtype: object
```

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

In [2]:
```python
raw_data_info = {
    "customer_id": [112, 403, 999, 543, 123],
    "name": ["John", "Kelly", "Sam", "April", "Bobbo"],
    "email": ["jman@gmail", "kelly@aol.com", "sports@school.edu", "April@yahoo.com", "HeyImBobbo@msn.com"]
}
info_pd = pd.DataFrame(raw_data_info, columns=["customer_id", "name", "email"])
info_pd
```

Out[2]:

	customer_id	name	email
0	112	John	jman@gmail
1	403	Kelly	kelly@aol.com
2	999	Sam	sports@school.edu
3	543	April	April@yahoo.com
4	123	Bobbo	HeyImBobbo@msn.com

In [3]:
```python
# Create DataFrames
raw_data_items = {
    "customer_id": [403, 112, 543, 999, 654],
    "item": ["soda", "chips", "TV", "Laptop", "Cooler"],
    "cost": [3.00, 4.50, 600, 900, 150]
}
items_pd = pd.DataFrame(raw_data_items, columns=[
                        "customer_id", "item", "cost"])
items_pd
```

Out[3]:

	customer_id	item	cost
0	403	soda	3.0
1	112	chips	4.5
2	543	TV	600.0
3	999	Laptop	900.0
4	654	Cooler	150.0

In [4]:
```python
# Merge two dataframes using an inner join
merge_table = pd.merge(info_pd, items_pd, on="customer_id")
merge_table
```

Out[4]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0

In [5]:
```python
# Merge two dataframes using an outer join
merge_table = pd.merge(info_pd, items_pd, on="customer_id", how="outer")
merge_table
```

Out[5]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	123	Bobbo	HeyImBobbo@msn.com	NaN	NaN
5	654	NaN	NaN	Cooler	150.0

In [6]:
```python
# Merge two dataframes using a left join
merge_table = pd.merge(info_pd, items_pd, on="customer_id", how="left")
merge_table
```

Out[6]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	123	Bobbo	HeyImBobbo@msn.com	NaN	NaN

In [7]:
```python
# Merge two dataframes using a right join
merge_table = pd.merge(info_pd, items_pd, on="customer_id", how="right")
merge_table
```

Out[7]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	654	NaN	NaN	Cooler	150.0

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

In [2]:
```
bitcoin_csv = "Resources/bitcoin_cash_price.csv"
dash_csv = "Resources/dash_price.csv"
```

In [3]:
```
bitcoin_df = pd.read_csv(bitcoin_csv)
dash_df = pd.read_csv(dash_csv)
```

In [4]:
```
bitcoin_df.head()
```

Out[4]:

	Date	Open	High	Low	Close	Volume	Market Cap
0	17-Sep-17	438.90	438.90	384.06	419.86	221828000.0	7,279,520,000
1	16-Sep-17	424.49	450.98	388.20	440.22	313583000.0	7,039,590,000
2	15-Sep-17	369.49	448.39	301.69	424.02	707231000.0	6,126,800,000
3	14-Sep-17	504.22	510.47	367.04	367.04	257431000.0	8,359,650,000
4	13-Sep-17	509.47	519.20	471.22	503.61	340344000.0	8,445,540,000

In [5]:
```
dash_df.head()
```

Out[5]:

	Date	Open	High	Low	Close	Volume	Market Cap
0	17-Sep-17	298.59	315.58	278.17	313.84	38081600.0	2,257,850,000
1	16-Sep-17	284.50	301.23	276.57	298.86	43702600.0	2,150,800,000
2	15-Sep-17	236.05	300.11	220.51	284.36	72695500.0	1,784,040,000
3	14-Sep-17	301.11	303.74	236.24	236.24	35013800.0	2,275,100,000
4	13-Sep-17	324.72	325.16	287.25	301.29	28322500.0	2,452,930,000

In [6]:
```
# Merge the two DataFrames together based on the Dates they share
crypto_df = pd.merge(bitcoin_df, dash_df, on="Date")
crypto_df.head()
```

Out[6]:

	Date	Open_x	High_x	Low_x	Close_x	Volume_x	Market Cap_x	Open_y	High_y	Low_y	Close_y	Volume_y	Market Cap_y
0	17-Sep-17	438.90	438.90	384.06	419.86	221828000.0	7,279,520,000	298.59	315.58	278.17	313.84	38081600.0	2,257,850,000
1	16-Sep-17	424.49	450.98	388.20	440.22	313583000.0	7,039,590,000	284.50	301.23	276.57	298.86	43702600.0	2,150,800,000
2	15-Sep-17	369.49	448.39	301.69	424.02	707231000.0	6,126,800,000	236.05	300.11	220.51	284.36	72695500.0	1,784,040,000
3	14-Sep-17	504.22	510.47	367.04	367.04	257431000.0	8,359,650,000	301.11	303.74	236.24	236.24	35013800.0	2,275,100,000
4	13-Sep-17	509.47	519.20	471.22	503.61	340344000.0	8,445,540,000	324.72	325.16	287.25	301.29	28322500.0	2,452,930,000

In [7]:
```
# Rename columns so that they are differentiated
crypto_df = crypto_df.rename(columns={"Open_x": "Bitcoin Open", "High_x": "Bitcoin High", "Low_x": "Bitcoin Low",
                                      "Close_x": "Bitcoin Close", "Volume_x": "Bitcoin Volume", "Market Cap_x": "Bitcoin Market Cap"})

crypto_df = crypto_df.rename(columns={"Open_y": "Dash Open", "High_y": "Dash High", "Low_y": "Dash Low",
                                      "Close_y": "Dash Close", "Volume_y": "Dash Volume", "Market Cap_y": "Dash Market Cap"})

crypto_df.head()
```

Out[7]:

	Date	Bitcoin Open	Bitcoin High	Bitcoin Low	Bitcoin Close	Bitcoin Volume	Bitcoin Market Cap	Dash Open	Dash High	Dash Low	Dash Close	Dash Volume	Dash Market Cap
0	17-Sep-17	438.90	438.90	384.06	419.86	221828000.0	7,279,520,000	298.59	315.58	278.17	313.84	38081600.0	2,257,850,000
1	16-Sep-17	424.49	450.98	388.20	440.22	313583000.0	7,039,590,000	284.50	301.23	276.57	298.86	43702600.0	2,150,800,000
2	15-Sep-17	369.49	448.39	301.69	424.02	707231000.0	6,126,800,000	236.05	300.11	220.51	284.36	72695500.0	1,784,040,000
3	14-Sep-17	504.22	510.47	367.04	367.04	257431000.0	8,359,650,000	301.11	303.74	236.24	236.24	35013800.0	2,275,100,000
4	13-Sep-17	509.47	519.20	471.22	503.61	340344000.0	8,445,540,000	324.72	325.16	287.25	301.29	28322500.0	2,452,930,000

In [8]:
```
# alternatively you can set your suffixes when the merge occurs
alternative_merge = pd.merge(
    bitcoin_df, dash_df, on="Date", suffixes=("_Bitcoin", "_Dash"))
alternative_merge.head()
```

Out[8]:

	Date	Open_Bitcoin	High_Bitcoin	Low_Bitcoin	Close_Bitcoin	Volume_Bitcoin	Market Cap_Bitcoin	Open_Dash	High_Dash	Low_Dash	Close_Dash	Volume_Dash	Market Cap_Dash
0	17-Sep-17	438.90	438.90	384.06	419.86	221828000.0	7,279,520,000	298.59	315.58	278.17	313.84	38081600.0	2,257,850,000
1	16-Sep-17	424.49	450.98	388.20	440.22	313583000.0	7,039,590,000	284.50	301.23	276.57	298.86	43702600.0	2,150,800,000
2	15-Sep-17	369.49	448.39	301.69	424.02	707231000.0	6,126,800,000	236.05	300.11	220.51	284.36	72695500.0	1,784,040,000
3	14-Sep-17	504.22	510.47	367.04	367.04	257431000.0	8,359,650,000	301.11	303.74	236.24	236.24	35013800.0	2,275,100,000
4	13-Sep-17	509.47	519.20	471.22	503.61	340344000.0	8,445,540,000	324.72	325.16	287.25	301.29	28322500.0	2,452,930,000

In [9]:
```
# Collecting best open for Bitcoin and Dash
bitcoin_open = crypto_df["Bitcoin Open"].max()
dash_open = crypto_df["Dash Open"].max()

# Collecting best close for Bitcoin and Dash
bitcoin_close = crypto_df["Bitcoin Close"].max()
dash_close = crypto_df["Dash Close"].max()

# Collecting the total volume for Bitcoin and Dash
bitcoin_volume = round(crypto_df["Bitcoin Volume"].sum()/1000000, 2)
dash_volume = round(crypto_df["Dash Volume"].sum()/1000000, 2)
```

In [10]:

```
# Creating a summary DataFrame using above values
summary_df = pd.DataFrame({"Best Bitcoin Open": [bitcoin_open],
                           "Best Bitcoin Close": [bitcoin_close],
                           "Total Bitcoin Volume": str(bitcoin_volume)+" million",
                           "Best Dash Open": [dash_open],
                           "Best Dash Close": [dash_close],
                           "Total Dash Volume": str(dash_volume)+" million"})

summary_df
```

Out[10]:

	Best Bitcoin Close	Best Bitcoin Open	Best Dash Close	Best Dash Open	Total Bitcoin Volume	Total Dash Volume
0	754.56	772.42	399.85	400.42	24383.05 million	2960.28 million

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

In [2]:
```python
raw_data = {
    'Class': ['Oct', 'Oct', 'Jan', 'Jan', 'Oct', 'Jan'],
    'Name': ["Cyndy", "Logan", "Laci", "Elmer", "Crystle", "Emmie"],
    'Test Score': [90, 59, 72, 88, 98, 60]}
df = pd.DataFrame(raw_data)
df
```

Out[2]:

	Class	Name	Test Score
0	Oct	Cyndy	90
1	Oct	Logan	59
2	Jan	Laci	72
3	Jan	Elmer	88
4	Oct	Crystle	98
5	Jan	Emmie	60

In [3]:
```python
# Create the bins in which Data will be held
# Bins are 0, 59, 69, 79, 89, 100.
bins = [0, 59, 69, 79, 89, 100]

# Create the names for the four bins
group_names = ["F", "D", "C", "B", "A"]
```

In [4]:
```python
df["Test Score Summary"] = pd.cut(df["Test Score"], bins, labels=group_names)
df
```

Out[4]:

	Class	Name	Test Score	Test Score Summary
0	Oct	Cyndy	90	A
1	Oct	Logan	59	F
2	Jan	Laci	72	C
3	Jan	Elmer	88	B
4	Oct	Crystle	98	A
5	Jan	Emmie	60	D

In [5]:

```
# Creating a group based off of the bins
df = df.groupby("Test Score Summary")
df.max()
```

Out[5]:

Test Score Summary	Class	Name	Test Score
F	Oct	Logan	59
D	Jan	Emmie	60
C	Jan	Laci	72
B	Jan	Elmer	88
A	Oct	Cyndy	98

In [ ]:

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

In [2]:
```
# Create a path to the csv and read it into a Pandas DataFrame
csv_path = "Resources/ted_talks.csv"
ted_df = pd.read_csv(csv_path)

ted_df.head()
```

Out[2]:

	comments	description	duration	event	languages	main_speaker	name	title	views
0	4553	Sir Ken Robinson makes an entertaining and pro...	1164	TED2006	60	Ken Robinson	Ken Robinson: Do schools kill creativity?	Do schools kill creativity?	47227110
1	265	With the same humor and humanity he exuded in ...	977	TED2006	43	Al Gore	Al Gore: Averting the climate crisis	Averting the climate crisis	3200520
2	124	New York Times columnist David Pogue takes aim...	1286	TED2006	26	David Pogue	David Pogue: Simplicity sells	Simplicity sells	1636292
3	200	In an emotionally charged talk, MacArthur-winn...	1116	TED2006	35	Majora Carter	Majora Carter: Greening the ghetto	Greening the ghetto	1697550
4	593	You've never seen data presented like this. Wi...	1190	TED2006	48	Hans Rosling	Hans Rosling: The best stats you've ever seen	The best stats you've ever seen	12005869

In [3]:
```
# Figure out the minimum and maximum views for a TED Talk
print(ted_df["views"].max())
print(ted_df["views"].min())
```

```
47227110
50443
```

In [4]:
```
# Create bins in which to place values based upon TED Talk views
bins = [0, 199999, 399999, 599999, 799999, 999999,
        1999999, 2999999, 3999999, 4999999, 50000000]

# Create labels for these bins
group_labels = ["0 to 199k", "200k to 399k", "400k to 599k", "600k to 799k", "800k to 999k", "1mil to 2mil",
                "2mil to 3mil", "3mil to 4mil", "4mil to 5mil", "5mil to 50mil"]
```

In [5]:
```
# Slice the data and place it into bins
pd.cut(ted_df["views"], bins, labels=group_labels).head()
```

Out[5]:
```
0    5mil to 50mil
1     3mil to 4mil
2     1mil to 2mil
3     1mil to 2mil
4    5mil to 50mil
Name: views, dtype: category
Categories (10, object): [0 to 199k < 200k to 399k < 400k to 599k < 600k to 799k ... 2mil to 3mil < 3mil to 4mil < 4mil to 5mil
< 5mil to 50mil]
```

In [6]:
```
# Place the data series into a new column inside of the DataFrame
ted_df["View Group"] = pd.cut(ted_df["views"], bins, labels=group_labels)
ted_df.head()
```

Out[6]:

	comments	description	duration	event	languages	main_speaker	name	title	views	View Group
0	4553	Sir Ken Robinson makes an entertaining and pro...	1164	TED2006	60	Ken Robinson	Ken Robinson: Do schools kill creativity?	Do schools kill creativity?	47227110	5mil to 50mil
1	265	With the same humor and humanity he exuded in ...	977	TED2006	43	Al Gore	Al Gore: Averting the climate crisis	Averting the climate crisis	3200520	3mil to 4mil
2	124	New York Times columnist David Pogue takes aim...	1286	TED2006	26	David Pogue	David Pogue: Simplicity sells	Simplicity sells	1636292	1mil to 2mil
3	200	In an emotionally charged talk, MacArthur-winn...	1116	TED2006	35	Majora Carter	Majora Carter: Greening the ghetto	Greening the ghetto	1697550	1mil to 2mil
4	593	You've never seen data presented like this. Wi...	1190	TED2006	48	Hans Rosling	Hans Rosling: The best stats you've ever seen	The best stats you've ever seen	12005869	5mil to 50mil

In [7]:

```
# Create a GroupBy object based upon "View Group"
ted_group = ted_df.groupby("View Group")

# Find how many rows fall into each bin
print(ted_group["comments"].count())

# Get the average of each column within the GroupBy object
ted_group[["comments", "duration", "languages"]].mean()
```

```
View Group
0 to 199k         32
200k to 399k     135
400k to 599k     234
600k to 799k     307
800k to 999k     339
1mil to 2mil    1004
2mil to 3mil     239
3mil to 4mil      93
4mil to 5mil      68
5mil to 50mil     99
Name: comments, dtype: int64
```

Out[7]:

	comments	duration	languages
View Group			
0 to 199k	76.937500	898.187500	4.062500
200k to 399k	81.992593	832.192593	18.785185
400k to 599k	107.162393	870.517094	22.940171
600k to 799k	118.912052	829.039088	24.400651
800k to 999k	119.628319	798.772861	25.678466
1mil to 2mil	168.136454	809.899402	27.899402
2mil to 3mil	299.481172	832.430962	32.807531
3mil to 4mil	360.870968	809.505376	34.258065
4mil to 5mil	507.088235	920.514706	35.720588
5mil to 50mil	650.393939	884.282828	40.252525

In [ ]:

In [1]:
```python
import pandas as pd
```

In [2]:
```python
# Mapping lets you format an entire DataFrame
file = "Resources/sample_data.csv"
file_df = pd.read_csv(file)
file_df.head()
```

Out[2]:

	id	city	avg_cost	population	other
0	1	Houxiang	55.121518	609458	-15.66171
1	2	Leribe	95.782967	601963	-23.79499
2	3	Hengshan	57.867827	589509	1.31471
3	4	Sogcho	59.220634	948491	-11.38280
4	5	Kohlu	23.092232	92206	7.66861

In [3]:
```python
# Use Map to format all the columns
file_df["avg_cost"] = file_df["avg_cost"].map("${:.2f}".format)
file_df["population"] = file_df["population"].map("{:,}".format)
file_df["other"] = file_df["other"].map("{:.2f}".format)
file_df.head()
```

Out[3]:

	id	city	avg_cost	population	other
0	1	Houxiang	$55.12	609,458	-15.66
1	2	Leribe	$95.78	601,963	-23.79
2	3	Hengshan	$57.87	589,509	1.31
3	4	Sogcho	$59.22	948,491	-11.38
4	5	Kohlu	$23.09	92,206	7.67

In [4]:
```python
# Mapping has changed the datatypes of the columns to strings
file_df.dtypes
```

Out[4]:
```
id             int64
city          object
avg_cost      object
population    object
other         object
dtype: object
```

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

In [2]:
```
# The path to our CSV file
file = "Resources/KickstarterData.csv"

# Read our Kickstarter data into pandas
df = pd.read_csv(file)
df.head()
```

Out[2]:

	id	photo	name	blurb	goal	pledged	state	slug	disable_communication	country	...	location	
0	1645666704	{"small":"https://ksr-ugc.imgix.net/assets/012...	The Class Act Players Theatre Company Presents...	The Class Act Players put on another one of th...	1500.0	2925.0	successful	the-class-act-players-theatre-company-presents...	False	US	...	{"country":"US","urls":{"web":{"discover":"htt...	{"d
1	874638240	{"small":"https://ksr-ugc.imgix.net/assets/012...	MR INCREDIBLE by Camilla Whitehill - VAULT Fes...	A brand new play about love and entitlement, b...	2500.0	2936.0	successful	mr-incredible-by-camilla-whitehill-vault-festival	False	GB	...	{"country":"GB","urls":{"web":{"discover":"htt...	{"d
2	247074984	{"small":"https://ksr-ugc.imgix.net/assets/012...	RUN	Yonni's pissed off in a world filled with scho...	1000.0	1200.0	successful	run-10	False	GB	...	{"country":"GB","urls":{"web":{"discover":"htt...	{"d
3	1941196813	{"small":"https://ksr-ugc.imgix.net/assets/012...	9th International Meeting of Youth Theatre sap...	27. April bis 1. Mai 2016 in Brixen/Südtirol/I...	2000.0	2135.0	successful	9th-international-meeting-of-youth-theatre-sap...	False	IT	...	{"country":"IT","urls":{"web":{"discover":"htt...	{"d
4	421961595	{"small":"https://ksr-ugc.imgix.net/assets/012...	Get Conti to the Ed Fringe!	The Italia Conti 2nd years are going to Ed Fri...	1000.0	1250.0	successful	get-conti-to-the-ed-fringe	False	GB	...	{"country":"GB","urls":{"web":{"discover":"htt...	{"d

5 rows × 33 columns

In [3]:
```
# Get a list of all of our columns for easy reference
df.columns
```

Out[3]:
```
Index(['id', 'photo', 'name', 'blurb', 'goal', 'pledged', 'state', 'slug',
       'disable_communication', 'country', 'currency', 'currency_symbol',
       'currency_trailing_code', 'deadline', 'state_changed_at', 'created_at',
       'launched_at', 'staff_pick', 'is_starrable', 'backers_count',
       'static_usd_rate', 'usd_pledged', 'creator', 'location', 'category',
       'profile', 'spotlight', 'urls', 'source_url', 'friends', 'is_starred',
       'is_backing', 'permissions'],
      dtype='object')
```

In [4]:
```
# Extract "name", "goal", "pledged", "state", "country", "staff_pick",
# "backers_count", and "spotlight"
reduced_kickstarter_df = df.loc[:, ["name", "goal", "pledged",
                                    "state", "country", "staff_pick", "backers_count", "spotlight"]]
reduced_kickstarter_df
```

Out[4]:

	name	goal	pledged	state	country	staff_pick	backers_count	spotlight
0	The Class Act Players Theatre Company Presents...	1500.0	2925.00	successful	US	False	17	True
1	MR INCREDIBLE by Camilla Whitehill - VAULT Fes...	2500.0	2936.00	successful	GB	True	15	True
2	RUN	1000.0	1200.00	successful	GB	False	30	True
3	9th International Meeting of Youth Theatre sap...	2000.0	2135.00	successful	IT	False	24	True
4	Get Conti to the Ed Fringe!	1000.0	1250.00	successful	GB	False	28	True
5	Somebody Out There Loves Me @ Edinburgh Fringe...	1100.0	1100.00	successful	GB	False	41	True
6	Stand Up & Slam! at the Edinburgh Fringe	1500.0	1605.00	successful	GB	False	49	True
7	Edinburgh Fringe Tour of Two Modern Classics	2550.0	2552.00	successful	GB	False	17	True
8	Forefront Festival 2015	7200.0	7230.00	successful	US	False	68	True
9	Tangerine Theatre presents GODDESS by Serena H...	1200.0	1625.00	successful	GB	False	34	True
10	I Would... get us to Edinburgh!	700.0	745.00	successful	GB	False	27	True
11	Hamlet the Hip-Hopera	9747.0	10103.00	successful	US	True	132	True

In [5]:
```
# Remove projects that made no money at all
reduced_kickstarter_df = reduced_kickstarter_df.loc[(
    reduced_kickstarter_df["pledged"] > 0)]
reduced_kickstarter_df.head()
```

Out[5]:

	name	goal	pledged	state	country	staff_pick	backers_count	spotlight
0	The Class Act Players Theatre Company Presents...	1500.0	2925.0	successful	US	False	17	True
1	MR INCREDIBLE by Camilla Whitehill - VAULT Fes...	2500.0	2936.0	successful	GB	True	15	True
2	RUN	1000.0	1200.0	successful	GB	False	30	True
3	9th International Meeting of Youth Theatre sap...	2000.0	2135.0	successful	IT	False	24	True
4	Get Conti to the Ed Fringe!	1000.0	1250.0	successful	GB	False	28	True

In [6]:

```
# Collect only those projects that were hosted in the US.

# Create a list of the columns
columns = [
    "name", "goal", "pledged", "state",
    "country", "staff_pick", "backers_count", "spotlight"]

#  Create a new df for "US" with the columns.
hosted_in_us = reduced_kickstarter_df.loc[reduced_kickstarter_df["country"] == "US",  columns]
hosted_in_us.head()
```

Out[6]:

	name	goal	pledged	state	country	staff_pick	backers_count	spotlight
0	The Class Act Players Theatre Company Presents...	1500.0	2925.0	successful	US	False	17	True
8	Forefront Festival 2015	7200.0	7230.0	successful	US	False	68	True
11	Hamlet the Hip-Hopera	9747.0	10103.0	successful	US	True	132	True
14	Pride Con	15000.0	15110.0	successful	US	False	60	True
15	En Garde Arts Emerging Artists Festival BOSSS	10000.0	10306.0	successful	US	True	80	True

In [7]:

```
# Create a new column that finds the average amount pledged to a project
average_donation = hosted_in_us['pledged'] / hosted_in_us['backers_count']
average_donation
```

Out[7]:

```
0       172.058824
8       106.323529
11       76.537879
14      251.833333
15      128.825000
17       66.578947
19       76.470588
20       61.666667
31      105.000000
32      143.181818
33       69.117647
39      167.117647
42       62.714286
43       81.975000
44       54.241935
45       65.216667
46       42.036765
47       99.131579
48       54.347826
50       93.147059
52       37.016129
54       90.130952
59      119.047619
64      229.772727
67       59.701493
73       27.133333
74       50.500000
75      106.358491
77       54.961165
79       44.426230
           ...
4076    107.002075
4077     50.981818
4078     52.551020
4080     47.878788
4081    150.416667
4082     96.666667
4083    133.333333
4084     51.222222
4086     93.813433
4087    176.086957
4088     45.478261
4091     97.916667
4092     77.186047
4093     87.961538
4094    122.536585
4095     98.200000
4097     88.739130
4099     63.574074
4100    106.797030
4101    111.892857
4103     58.064516
4104     60.300892
4105     70.884956
4106     68.353846
4108    178.571429
4109     51.219512
4110    100.171429
4115    119.192488
4117     66.111111
4118     52.000000
Length: 2129, dtype: float64
```

In [8]:
```
# Create a new column that finds the average amount pledged to a project
hosted_in_us["average_donation"] = hosted_in_us['pledged'] / \
    hosted_in_us['backers_count']
```

In [9]:
```
# First convert "average_donation", "goal", and "pledged" columns to float
# Then Format to go to two decimal places, include a dollar sign, and use comma notation

hosted_in_us["average_donation"] = hosted_in_us["average_donation"].astype(float).map(
    "${:,.2f}".format)
hosted_in_us["goal"] = hosted_in_us["goal"].astype(float).map("${:,.2f}".format)
hosted_in_us["pledged"] = hosted_in_us["pledged"].astype(float).map("${:,.2f}".format)
```

In [10]:
```
hosted_in_us.head()
```

Out[10]:

	name	goal	pledged	state	country	staff_pick	backers_count	spotlight	average_donation
0	The Class Act Players Theatre Company Presents...	$1,500.00	$2,925.00	successful	US	False	17	True	$172.06
8	Forefront Festival 2015	$7,200.00	$7,230.00	successful	US	False	68	True	$106.32
11	Hamlet the Hip-Hopera	$9,747.00	$10,103.00	successful	US	True	132	True	$76.54
14	Pride Con	$15,000.00	$15,110.00	successful	US	False	60	True	$251.83
15	En Garde Arts Emerging Artists Festival BOSSS	$10,000.00	$10,306.00	successful	US	True	80	True	$128.82

In [11]:
```
# Calculate the total number of backers for all US projects
hosted_in_us["backers_count"].sum()
```

Out[11]: 89273

In [12]:
```
# Calculate the average number of backers for all US projects
hosted_in_us["backers_count"].mean()
```

Out[12]: 41.931892907468296

In [13]:
```
# Collect only those US campaigns that have been picked as a "Staff Pick"
picked_by_staff = hosted_in_us.loc[hosted_in_us["staff_pick"] == True]
picked_by_staff
```

Out[13]:

	name	goal	pledged	state	country	staff_pick	backers_count	spotlight	average_donation
11	Hamlet the Hip-Hopera	$9,747.00	$10,103.00	successful	US	True	132	True	$76.54
15	En Garde Arts Emerging Artists Festival BOSSS	$10,000.00	$10,306.00	successful	US	True	80	True	$128.82
39	"Poor People" at FringeNYC 2015	$5,500.00	$5,682.00	successful	US	True	34	True	$167.12
44	Queen Mab's Steampunk and Fairie Street Festival	$1,300.00	$3,363.00	successful	US	True	62	True	$54.24
45	RAFT: a new play by Emily Kitchens	$7,500.00	$7,826.00	successful	US	True	120	True	$65.22
47	The Spinning Wheel: a son remixes a father's r...	$20,000.00	$22,602.00	successful	US	True	228	True	$99.13
50	Bloomers Presents: LaughtHERfest	$8,000.00	$9,501.00	successful	US	True	102	True	$93.15
54	Natasha Noman's Noman's Land | Aug 5-15th Edi...	$7,000.00	$7,571.00	successful	US	True	84	True	$90.13
107	Peter/Wendy goes to the 2015 Edinburgh Fringe ...	$10,000.00	$12,003.00	successful	US	True	95	True	$126.35
115	La Lune de Femme goes to New Orleans Fringe	$5,000.00	$5,519.00	successful	US	True	79	True	$69.86
117	Secular Solstice 2014	$7,500.00	$8,157.01	successful	US	True	164	True	$49.74
122	Southern Shakespeare Festival	$7,500.00	$8,202.00	successful	US	True	63	True	$130.19

In [14]:
```
# Group by the state of the campaigns and see if staff picks matter (Seems to matter quite a bit)
state_groups = picked_by_staff.groupby("state")
state_groups["name"].count()
```

Out[14]:
```
state
canceled        6
failed         21
live            2
successful    145
Name: name, dtype: int64
```

In [ ]:

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

In [2]:
```
# Reference to CSV and reading CSV into Pandas DataFrame
csv_path = "Resources/flavors_of_cacao.csv"
chocolate_ratings_df = pd.read_csv(csv_path)
chocolate_ratings_df.head(10)
```

Out[2]:

	Company (Maker-if known)	Specific Bean Origin or Bar Name	REF	Review Date	Cocoa Percent	Company Location	Rating	Bean Type	Broad Bean Origin
0	A. Morin	Agua Grande	1876	2016	63%	France	3.75		Sao Tome
1	A. Morin	Kpime	1676	2015	70%	France	2.75		Togo
2	A. Morin	Atsane	1676	2015	70%	France	3.00		Togo
3	A. Morin	Akata	1680	2015	70%	France	3.50		Togo
4	A. Morin	Quilla	1704	2015	70%	France	3.50		Peru
5	A. Morin	Carenero	1315	2014	70%	France	2.75	Criollo	Venezuela
6	A. Morin	Cuba	1315	2014	70%	France	3.50		Cuba
7	A. Morin	Sur del Lago	1315	2014	70%	France	3.50	Criollo	Venezuela
8	A. Morin	Puerto Cabello	1319	2014	70%	France	3.75	Criollo	Venezuela
9	A. Morin	Pablino	1319	2014	70%	France	4.00		Peru

In [3]:
```
chocolate_ratings_df.columns
```

Out[3]:
```
Index(['Company (Maker-if known)', 'Specific Bean Origin or Bar Name', 'REF',
       'Review Date', 'Cocoa Percent', 'Company Location', 'Rating',
       'Bean Type', 'Broad Bean Origin'],
      dtype='object')
```

In [4]:
```
# Converting the "Cocoa Percent" column to floats
chocolate_ratings_df["Cocoa Percent"] = chocolate_ratings_df["Cocoa Percent"].replace(
    '%', '', regex=True).astype('float')

# Finding the average cocoa percent
chocolate_ratings_df["Cocoa Percent"].mean()
```

Out[4]:
```
71.6983286908078
```

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

In [2]:
```
# Create a reference to the CSV and import it into a Pandas DataFrame
csv_path = "Resources/EclipseBugs.csv"
eclipse_df = pd.read_csv(csv_path)
```

In [3]:
```
eclipse_df = eclipse_df.rename(columns={"Bug\nID": "Bug ID",
                                        "Assignee\nReal\nName": "Assignee Real Name",
                                        "Number of\nComments": "Number of Comments",
                                        "Reporter\nReal\nName": "Reporter Real Name",
                                        "Target\nMilestone": "Target Milestone"})
eclipse_df.columns
```

Out[3]:
```
Index(['Bug ID', 'Product', 'Component', 'Assignee', 'Status', 'Resolution',
       'Summary', 'Changed', 'Assignee Real Name', 'Classification',
       'Hardware', 'Number of Comments', 'Opened', 'OS', 'Priority',
       'Reporter', 'Reporter Real Name', 'Severity', 'Target Milestone',
       'Version', 'Votes'],
      dtype='object')
```

In [4]:
```
# Finding the average number of comments per bug
average_comments = eclipse_df["Number of Comments"].mean()
average_comments
```

Out[4]:
```
8.75
```

In [5]:
```
# Grouping the DataFrame by "Assignee"
assignee_group = eclipse_df.groupby("Assignee")

# Count how many of each component Assignees worked on and create DataFrame
assignee_work = pd.DataFrame(assignee_group["Component"].value_counts())
assignee_work.head()
```

Out[5]:

		Component
Assignee	**Component**	
Aaron_Ferguson	**UI**	10
Adam_Schlegel	**UI**	7
ChrisAustin	**User Assistance**	3
Claude_Knaus	**UI**	31
	Text	7

In [6]:

```
# Rename the "Component" column to "Component Bug Count"
assignee_work = assignee_work.rename(
    columns={"Component": "Component Bug Count"})
assignee_work.head()
```

Out[6]:

		Component Bug Count
Assignee	**Component**	
Aaron_Ferguson	**UI**	10
Adam_Schlegel	**UI**	7
ChrisAustin	**User Assistance**	3
Claude_Knaus	**UI**	31
	Text	7

In [7]:

```
# Find the percentage of bugs overall fixed by each Assignee
total_bugs = len(eclipse_df)
bugs_per_user = assignee_group["Assignee"].count()

user_bug_percent = pd.DataFrame((bugs_per_user/total_bugs)*100)
user_bug_percent.head()
```

Out[7]:

	Assignee
Assignee	
Aaron_Ferguson	0.10
Adam_Schlegel	0.07
ChrisAustin	0.03
Claude_Knaus	0.38
Curtis_Windatt	0.06

In [8]:

```
# Rename the "Assignee" column to "Percent of Total Bugs Assigned"
user_bug_percent = user_bug_percent.rename(
    columns={"Assignee": "Percent of Total Bugs Assigned"})

# Reset the index for this DataFrame so "Assignee" is a column
user_bug_percent = user_bug_percent.reset_index()
user_bug_percent.head()
```

Out[8]:

	Assignee	Percent of Total Bugs Assigned
0	Aaron_Ferguson	0.10
1	Adam_Schlegel	0.07
2	ChrisAustin	0.03
3	Claude_Knaus	0.38
4	Curtis_Windatt	0.06

In [9]:

```
# Reset the index of "assignee_group" so that "Assignee" and "Component" are columns
assignee_work = assignee_work.reset_index()
assignee_work.head()
```

Out[9]:

	Assignee	Component	Component Bug Count
0	Aaron_Ferguson	UI	10
1	Adam_Schlegel	UI	7
2	ChrisAustin	User Assistance	3
3	Claude_Knaus	UI	31
4	Claude_Knaus	Text	7

In [10]:

```
# Merge the "Percent of Total Bugs Assigned" into the DataFrame
assignee_work = assignee_work.merge(user_bug_percent, on="Assignee")
assignee_work.head()
```

Out[10]:

	Assignee	Component	Component Bug Count	Percent of Total Bugs Assigned
0	Aaron_Ferguson	UI	10	0.10
1	Adam_Schlegel	UI	7	0.07
2	ChrisAustin	User Assistance	3	0.03
3	Claude_Knaus	UI	31	0.38
4	Claude_Knaus	Text	7	0.38

[04-Pandas Homework Instructions](#)
Mark Flynn authored 2 weeks ago

10d66a0f

pandas_grading_rubric.pdf 180 KB

Instructions:

Evaluate the homework against the outlined criteria in the below rubric, assigning a rating to each criterion. Add points earned across all criteria and convert the total points to a letter grade, assigning a "+" or "-" letter grade designation at your discretion.

A (+/-)	90+	C (+/-)	40-64	F (+/-)	<15
B (+/-)	65-89	D (+/-)	15-39		

Notes:

The deployed assignment utilizes the **Pandas** library to analyze 1 of 2 challenges. Only one assignment will be accepted for grading. The source code should also be deployed to **Github** or **Gitlab**.

Rubric for Heroes Of PyMoli:

	Mastery **20 points**	**Approaching Mastery** **15 points**	**Progressing** **10 points**	**Emerging** **5-0 points**	**Incomplete**
Expected output displayed	Output for Pymoli contains all: ✓ Total Players ✓ Purchase Analysis (Total) ✓ Gender Demographics ✓ Purchase Analysis (Gender) ✓ Age Demographics ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items ✓ Most profitable Items	Output for Pymoli contains at least 7: ✓ Total Players ✓ Purchase Analysis (Total) ✓ Gender Demographics ✓ Purchase Analysis (Gender) ✓ Age Demographics ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items ✓ Most profitable Items	Output for Pymoli contains at least 5: ✓ Total Players ✓ Purchase Analysis (Total) ✓ Gender Demographics ✓ Purchase Analysis (Gender) ✓ Age Demographics ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items ✓ Most profitable Items	Output for Pymoli contains 2 or fewer: ✓ Total Players ✓ Purchase Analysis (Total) ✓ Gender Demographics ✓ Purchase Analysis (Gender) ✓ Age Demographics ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items ✓ Most profitable Items	No submission was received -OR- Submission was empty or blank -OR- Submission contains evidence of academic dishonesty
Functions used on DataFrames	The following functions are used on DataFrames and produce correct results: ✓ Mean ✓ Sum ✓ Count	The following functions are used on DataFrames and produce varying results: ✓ Mean ✓ Sum ✓ Count	Two of the following functions are used on DataFrames to produce varying results: ✓ Mean ✓ Sum ✓ Count	One or fewer of the following functions are used on DataFrames to produce varying results: ✓ Mean ✓ Sum ✓ Count	
GroupBy used	GroupBy is used in Pymoli in determining the following: ✓ Purchase Analysis (Gender) ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items	GroupBy is used for Pymoli in determining at least 3 of the following: ✓ Purchase Analysis (Gender) ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items	GroupBy is used for Pymoli in determining at least 2 of the following: ✓ Purchase Analysis (Gender) ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items	GroupBy is used for Pymoli in determining 1 or fewer of the following: ✓ Purchase Analysis (Gender) ✓ Purchasing Analysis (Age) ✓ Top Spenders ✓ Most Popular Items	
Cut method used to create new series of binned data	Pymoli data was cut and binned for both correctly: ✓ Age Demographics ✓ Purchasing Analysis (Age)	Pymoli data was cut and binned for one correctly: ✓ Age Demographics ✓ Purchasing Analysis (Age)	Pymoli data attempted to cut and binned for one with errors: ✓ Age Demographics ✓ Purchasing Analysis (Age)	Pymoli data was either not attempted or was attempted to cut and bin but produces no results: ✓ Age Demographics ✓ Purchasing Analysis (Age)	
Written Report	Presents a cohesive written analysis that: ✓ Draws three correct conclusions from the data for Pymoli	Presents a cohesive written analysis that: ✓ Draws at least two correct conclusions from the data for Pymoli	Presents a cohesive written analysis that: ✓ Draws at least one correct and one incomplete conclusion from the data for Pymoli	Presents a limited written analysis or no written analysis that: ✓ Incorrect and incomplete conclusion from the data for Pymoli	

Instructions:

Evaluate the homework against the outlined criteria in the below rubric, assigning a rating to each criterion. Add points earned across all criteria and convert the total points to a letter grade, assigning a "+" or "-" letter grade designation at your discretion.

A (+/-)	100-90	C (+/-)	79-70	F (+/-)	< 60
B (+/-)	89-80	D (+/-)	69-60		

Rubric for Academy of Py:

	Mastery 20 points	Approaching Mastery 15 points	Progressing 10 points	Emerging 5-0 points	Incomplete
Expected output displayed	✓ Output for Pyschool contains all: ✓ `Direct Summary` ✓ `School Summary` ✓ `Top Performing Schools (By Passing Rate)` ✓ `Bottom Performing Schools (By Passing Rate)` ✓ `Math Score by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending` ✓ `Scores by School Size` ✓ `Scores by School Type`	✓ Output for Pyschool contains at least 7: ✓ `Direct Summary` ✓ `School Summary` ✓ `Top Performing Schools (By Passing Rate)` ✓ `Bottom Performing Schools (By Passing Rate)` ✓ `Math Score by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending` ✓ `Scores by School Size` ✓ `Scores by School Type`	✓ Output for Pyschool contains at least 5: ✓ `Direct Summary` ✓ `School Summary` ✓ `Top Performing Schools (By Passing Rate)` ✓ `Bottom Performing Schools (By Passing Rate)` ✓ `Math Score by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending`	✓ Output for Pyschool contains 2 or fewer: ✓ `Direct Summary` ✓ `School Summary` ✓ `Top Performing Schools (By Passing Rate)` ✓ `Bottom Performing Schools (By Passing Rate)` ✓ `Math Score by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending`	No submission was received -OR- Submission was empty or blank -OR- Submission contains evidence of academic dishonesty
Functions used on DataFrames	The following functions are used on DataFrames and produce correct results: ✓ `Mean` ✓ `Sum` ✓ `Count`	The following functions are used on DataFrames and produce varying results: ✓ `Mean` ✓ `Sum` ✓ `Count`	Two of the following functions are used on DataFrames to produce varying results: ✓ `Mean` ✓ `Sum` ✓ `Count`	One or fewer of the following functions are used on DataFrames to produce varying results: ✓ `Mean` ✓ `Sum` ✓ `Count`	
GroupBy used	GroupBy is used in Pyschools in determining the following: ✓ `School Summary` ✓ `Math Scores by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending` ✓ `Scores by School Size` ✓ `Scores by School Type`	GroupBy is used for Pyschools in determining at least 4 of the following: ✓ `School Summary` ✓ `Math Scores by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending` ✓ `Scores by School Size` ✓ `Scores by School Type`	GroupBy is used for Pyschools in determining at least 3 of the following: ✓ `School Summary` ✓ `Math Scores by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending` ✓ `Scores by School Size` ✓ `Scores by School Type`	GroupBy is used for Pyschools in determining 1 or fewer of the following: ✓ `School Summary` ✓ `Math Scores by Grade` ✓ `Reading Score by Grade` ✓ `Scores by School Spending` ✓ `Scores by School Size` ✓ `Scores by School Type`	
Cut method used to create new series of binned data	Pyschools data was cut and binned for both correctly: ✓ `Scores by School Spending` ✓ `Scores by School Size`	Pyschools data was cut and binned for one correctly: ✓ `Scores by School Spending` ✓ `Scores by School Size`	Pyschools data was cut and binned for one with errors: ✓ `Scores by School Spending` ✓ `Scores by School Size`	Pyschools data was either not attempted or was attempted to cut and bin but produces no results: ✓ `Scores by School Spending` ✓ `Scores by School Size`	
Written Report	Presents a cohesive written analysis that: ✓ Draws two correct conclusions from the data for Pyschools	Presents a cohesive written analysis that: ✓ Draws at least one correct conclusion from the data for Pyschools	Presents a cohesive written analysis that: ✓ Draws at least one complete but incorrect conclusion from the data for Pyschools	Presents a limited written analysis or no written analysis that: ✓ Incorrect and incomplete conclusion form the data for Pyschools	

04-Pandas Homework Instructions
Mark Flynn authored 2 weeks ago

`10d66a0f`

README.md 8.47 KB

You're not allowed to edit files in this project directly. Please fork this project, make your changes there, and submit a merge request.

Pandas Homework - Pandas, Pandas, Pandas

Background

The data dive continues!

Now, it's time to take what you've learned about Python Pandas and apply it to new situations. For this assignment, you'll need to complete **one of two** (not both) Data Challenges. Once again, which challenge you take on is your choice. Just be sure to give it your all -- as the skills you hone will become powerful tools in your data analytics tool belt.

Before You Begin

1. Create a new repository for this project called `pandas-challenge`. **Do not add this homework to an existing repository**.

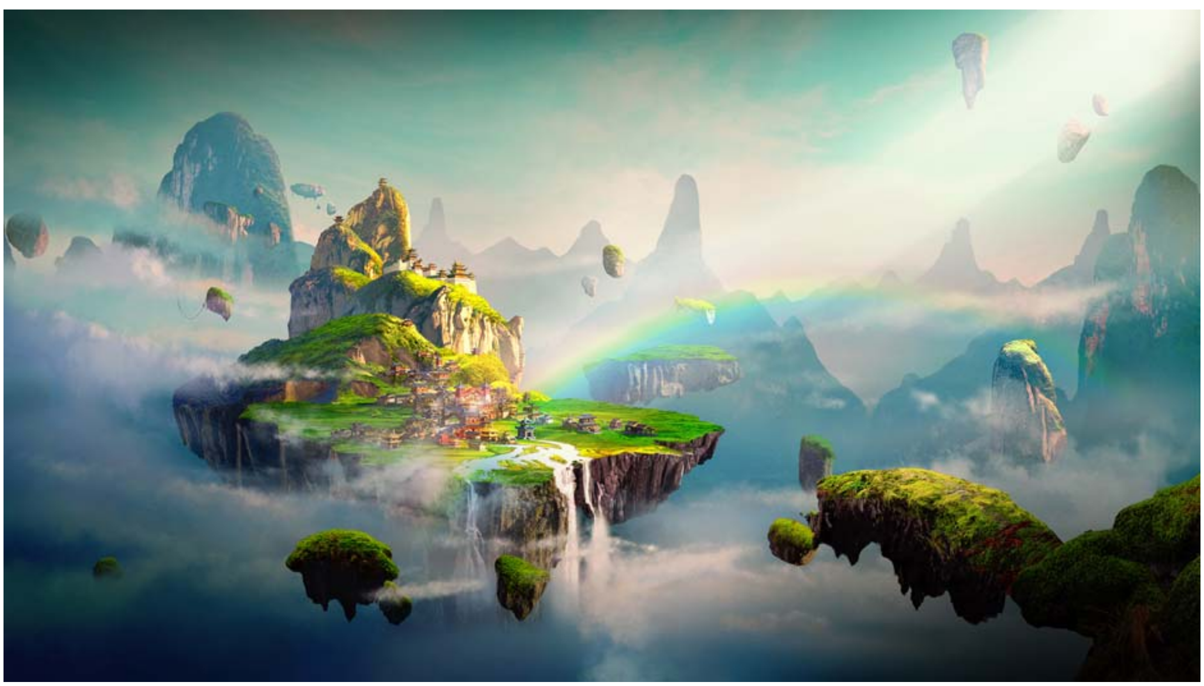
2. Clone the new repository to your computer.

3. Inside your local git repository, create a directory for the Pandas Challenge you choose. Use folder names corresponding to the challenges: **HeroesOfPymoli** or **PyCitySchools**.

4. Add your Jupyter notebook to this folder. This will be the main script to run for analysis.

5. Push the above changes to GitHub or GitLab.

Option 1: Heroes of Pymoli



Congratulations! After a lot of hard work in the data munging mines, you've landed a job as Lead Analyst for an independent gaming company. You've been assigned the task of analyzing the data for their most recent fantasy game Heroes of Pymoli.

Like many others in its genre, the game is free-to-play, but players are encouraged to purchase optional items that enhance their playing experience. As a first task, the company would like you to generate a report that breaks down the game's purchasing data into meaningful insights.

Your final report should include each of the following:

Player Count

- Total Number of Players

Purchasing Analysis (Total)

- Number of Unique Items
- Average Purchase Price
- Total Number of Purchases
- Total Revenue

Gender Demographics

- Percentage and Count of Male Players
- Percentage and Count of Female Players
- Percentage and Count of Other / Non-Disclosed

Purchasing Analysis (Gender)

- The below each broken by gender
  - Purchase Count
  - Average Purchase Price
  - Total Purchase Value
  - Average Purchase Total per Person by Gender

Age Demographics

- The below each broken into bins of 4 years (i.e. <10, 10-14, 15-19, etc.)
  - Purchase Count
  - Average Purchase Price
  - Total Purchase Value
  - Average Purchase Total per Person by Age Group

Top Spenders

- Identify the the top 5 spenders in the game by total purchase value, then list (in a table):
  - SN
  - Purchase Count
  - Average Purchase Price
  - Total Purchase Value

Most Popular Items

- Identify the 5 most popular items by purchase count, then list (in a table):
  - Item ID
  - Item Name
  - Purchase Count
  - Item Price
  - Total Purchase Value

Most Profitable Items

- Identify the 5 most profitable items by total purchase value, then list (in a table):
  - Item ID
  - Item Name
  - Purchase Count
  - Item Price
  - Total Purchase Value

As final considerations:

- You must use the Pandas Library and the Jupyter Notebook.
- You must submit a link to your Jupyter Notebook with the viewable Data Frames.
- You must include a written description of three observable trends based on the data.
- See [Example Solution] for a reference on expected format.

Option 2: Academy of Py

Well done! Having spent years analyzing financial records for big banks, you've finally scratched your idealistic itch and joined the education sector. In your latest role, you've become the Chief Data Scientist for your city's school district. In this capacity, you'll be helping the school board and mayor make strategic decisions regarding future school budgets and priorities.

As a first task, you've been asked to analyze the district-wide standardized test results. You'll be given access to every student's math and reading scores, as well as various information on the schools they attend. Your responsibility is to aggregate the data to and showcase obvious trends in school performance.

Your final report should include each of the following:

District Summary

- Create a high level snapshot (in table form) of the district's key metrics, including:
  - Total Schools
  - Total Students
  - Total Budget
  - Average Math Score
  - Average Reading Score
  - % Passing Math
  - % Passing Reading
  - Overall Passing Rate (Average of the above two)

School Summary

- Create an overview table that summarizes key metrics about each school, including:
  - School Name
  - School Type
  - Total Students
  - Total School Budget
  - Per Student Budget
  - Average Math Score
  - Average Reading Score
  - % Passing Math
  - % Passing Reading
  - Overall Passing Rate (Average of the above two)

Top Performing Schools (By Passing Rate)

- Create a table that highlights the top 5 performing schools based on Overall Passing Rate. Include:
  - School Name
  - School Type
  - Total Students
  - Total School Budget
  - Per Student Budget
  - Average Math Score
  - Average Reading Score
  - % Passing Math
  - % Passing Reading
  - Overall Passing Rate (Average of the above two)

Bottom Performing Schools (By Passing Rate)

- Create a table that highlights the bottom 5 performing schools based on Overall Passing Rate. Include all of the same metrics as above.

Math Scores by Grade**

- Create a table that lists the average Math Score for students of each grade level (9th, 10th, 11th, 12th) at each school.

Reading Scores by Grade

- Create a table that lists the average Reading Score for students of each grade level (9th, 10th, 11th, 12th) at each school.

Scores by School Spending

- Create a table that breaks down school performances based on average Spending Ranges (Per Student). Use 4 reasonable bins to group school spending. Include in the table each of the following:
  - Average Math Score
  - Average Reading Score
  - % Passing Math
  - % Passing Reading
  - Overall Passing Rate (Average of the above two)

Scores by School Size

- Repeat the above breakdown, but this time group schools based on a reasonable approximation of school size (Small, Medium, Large).

Scores by School Type

- Repeat the above breakdown, but this time group schools based on school type (Charter vs. District).

As final considerations:

- Use the pandas library and Jupyter Notebook.
- You must submit a link to your Jupyter Notebook with the viewable Data Frames.
- You must include a written description of at least two observable trends based on the data.
- See [Example Solution] for a reference on the expected format.

Hints and Considerations

- These are challenging activities for a number of reasons. For one, these activities will require you to analyze thousands of records. Hacking through the data to look for obvious trends in Excel is just not a feasible option. The size of the data may seem daunting, but pandas will allow you to efficiently parse through it.

- Second, these activities will also challenge you by requiring you to learn on your feet. Don't fool yourself into thinking: "I need to study pandas more closely before diving in." Get the basic gist of the library and then *immediately* get to work. When facing a daunting task, it's easy to think: "I'm just not ready to tackle it yet." But that's the surest way to never succeed. Learning to program requires one to constantly tinker, experiment, and learn on the fly. You are doing exactly the *right* thing, if you find yourself constantly practicing Google-Fu and diving into documentation. There is just no way (or reason) to try and memorize it all. Online references are available for you to use when you need them. So use them!

- Take each of these tasks one at a time. Begin your work, answering the basic questions: "How do I import the data?" "How do I convert the data into a DataFrame?" "How do I build the first table?" Don't get intimidated by the number of asks. Many of them are repetitive in nature with just a few tweaks. Be persistent and creative!

- Expect these exercises to take time! Don't get discouraged if you find yourself spending hours initially with little progress. Force yourself to deal with the discomfort of not knowing and forge ahead. Consider these hours an investment in your future!

- As always, feel encouraged to work in groups and get help from your TAs and Instructor. Just remember, true success comes from mastery and *not* a completed homework assignment. So challenge yourself to truly succeed!

Copyright