Intro to **Pandas** – week 4 – also started using **Jupyter Lab**

Python = programming language &/or kernel

Pandas = branch/module of python

Anaconda = conda = miniconda = manages python versions

Pandas is a Python module that contains structures and functions useful for data exploration and analysis.

The two main data structures Pandas introduces are **Series** and **DataFrames**.

IN SUPPLEMENTAL FOLDER… intro\_to\_pandas.ipynb GOT ABOVE AND SEE MORE…

Series & data frames notes below taken from here too

**HW notes**

Total\_Purchase\_Value\_Top\_Series = Total\_Purchase\_Value\_Top.groupby('SN').sum()**["Price"]**

**["Price"] makes it return a series vs. without = df**

**Allows it to go in descending order, or .head( ) as well**

1.1 Jupyter Intro

# Running the basic "Hello World" code

hello = "Hello World"

print(hello)

# Doing simple math

4 + 4

# Storing results in variables

a = 5

# Using those variables elsewhere in the code

a

# Variables will hold the value most recently run

# This means that, if we run the code above, it will now print 2

a = 2

1.2 Netflix Remix

# Modules

import os

import csv

from pathlib import Path

# Prompt user for video lookup

video = input("What show or movie are you looking for? ")

# Set path for file

csvpath = Path("Resources/netflix\_ratings.csv")

# Bonus

# Set variable to check if we found the video

found = False

# 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[6])

# Set variable to confirm we have found the video

found = True

# If the video is never found, alert the user

if found is False:

print("Sorry about this, we don't seem to have what you are looking for!")

1.3 Intro to Pandas

**Pandas Series**

* 1-D data structure (similar to Python lists, or an Excel column)
* Can contain multiple data types, but usually should contain data of one type
* Create a Pandas Series by passing in a **list** to **pd.Series()**
* By default, a Pandas Series will have an index that starts at 0; can access specific values using this index
* Learn more: <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html>

# Dependencies

import pandas as pd

# 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

# 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

# Convert a single dictionary containing lists into a dataframe

df = pd.DataFrame({"Dynasty": ["Early Dynastic Period", "Old

Kingdom"], "Pharoh": ["Thinis", "Memphis"]})

Df

**Pandas DataFrames**

* 2-D data structure with labeled rows and columns (similar to tables in Excel)
  + For example: if we were looking at traffic violations data for NYC, each row could represent a violation instance, and each column could represent a specific attribution of a violation (date, amount of fine, location, etc.)
* Create a Pandas Dataframe by using **pd.DataFrame()**, and passing in either a **list of dictionaries**, or a **dictionary with lists**
* A lot of data in the real world will be provided in tabular format which can be easily translated into DataFrames
* Learn more: <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html>

\*\*\* Key Activities

\*\*\* 1.4 Data Frame Shop

# Import Dependencies

import pandas as pd

# Create a DataFrame of frames using a dictionary of lists

frame\_df = pd.DataFrame({

"Frame": ["Ornate", "Classical", "Modern", "Wood", "Cardboard"],

"Price": [15.00, 12.50, 10.00, 5.00, 1.00],

"Sales": [100, 200, 150, 300, "N/A"]})frame\_df

EASIER...LESS WRITING ABOVE vs. BELOW

# Create a DataFrame of paintings using a list of dictionaries

painting\_df = pd.DataFrame([

{"Painting": "Mona Lisa (Knockoff)", "Price": 25,

"Popularity": "Very Popular"},

{"Painting": "Van Gogh (Knockoff)", "Price": 20, "Popularity": "Popular"},

{"Painting": "Starving Artist", "Price": 10, "Popularity": "Average"},

{"Painting": "Toddler Drawing", "Price": 1, "Popularity": "Not Popular"}])painting\_df

1.5 Data Functions

# Dependencies

import pandas as pd

# Save path to data set in a variable

data\_file = "Resources/dataSet.csv"

# Use Pandas to read data

data\_file\_pd = pd.read\_csv(data\_file)

data\_file\_pd.head()

# Display a statistical overview of the DataFrame

data\_file\_pd.describe()

# Reference a single column within a DataFrame

data\_file\_pd["Amount"].head()

# Reference multiple columns within a DataFrame

data\_file\_pd[["Amount", "Gender"]].head()

# The mean method averages the series

average = data\_file\_pd["Amount"].mean()

average

# The sum method adds every entry in the series

total = data\_file\_pd["Amount"].sum()

total

# The unique method shows every element of the series that appears only once

unique = data\_file\_pd["Last Name"].unique()

unique

# The value\_counts method counts unique values in a column

count = data\_file\_pd["Gender"].value\_counts()

count

# Calculations can also be performed on Series and added into DataFrames as new columns

thousands\_of\_dollars = data\_file\_pd["Amount"]/1000

data\_file\_pd["Thousands of Dollars"] = thousands\_of\_dollars

data\_file\_pd.head()

\*\*\* 1.6 Training Grounds

# Import Dependencies

import pandas as pd

import random

# A seriously gigantic DataFrame of individuals' names, their trainers, their weight, and their days as gym members

training\_data = pd.DataFrame({

"Name":[ ...] ...

})training\_data.head()

# Collecting a summary of all numeric data

training\_data.describe()

# Finding the names of the trainers

training\_data["Trainer"].unique()

# Finding how many students each trainer has

training\_data["Trainer"].value\_counts()

# Finding the average weight of all students

training\_data["Weight"].mean()

# Finding the combined weight of all students

training\_data["Weight"].sum()

# 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()

1.7 Column Manipulation

# Import Dependencies

import pandas as pd

# A gigantic DataFrame of individuals' names, their trainers, their weight, and their days as gym members

training\_data = pd.DataFrame({

"Name":["Gino Walker", ...

,59,13]})training\_data.head(10)

# Collecting a list of all columns within the DataFrame

training\_data.columns

# Reorganizing the columns using double brackets

organized\_df = training\_data[["Name","Trainer","Weight","Membership(Days)"]]

organized\_df.head()

# 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()

1.8 Hey Arnold

# Dependencies

import pandas as pd

# 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

# NOTE THAT PANDAS MAKES IT LOOK NICER THAN "PRINT (hey\_arnold)"

# 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

# 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

1.9 Reading.Writing CSV

# Dependencies

import pandas as pd

# Store filepath in a variable

file\_one = "Resources/DataOne.csv"

# Read our Data file with the pandas library

# Not every CSV requires an encoding, but be aware this can come up

file\_one\_df = pd.read\_csv(file\_one, encoding="ISO-8859-1")

# Show just the header

file\_one\_df.head()

# Show a single column

file\_one\_df["first\_name"].head()

# Show mulitple specific columns--note the extra brackets

file\_one\_df[["first\_name", "email"]].head()

# NOTE: Head does not change the DataFrame--it only displays it

# 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)

\*\*\* 1.10 Good Reads

# Import Dependencies

import pandas as pd

# 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()

# 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()

# 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()

# Push the remade DataFrame to a new CSV file

renamed\_df.to\_csv("Output/books\_clean.csv", encoding="utf-8", index=False, header=True)

\*\*\* 1.11 Good Reads Summary

# Import Dependencies

import pandas as pd

# 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()

# 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

total\_reviews = goodreads\_df["One Star Reviews"].sum() + goodreads\_df["Two Star Reviews"].sum(

) + goodreads\_df["Three Star Reviews"].sum() + goodreads\_df["Four Star Reviews"].sum() + goodreads\_df["Five Star Reviews"].sum()

# 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

From intro\_to\_pandas.ipynb in Supplemental folder:

Two functions exist to make life easier when trying to slice and dice any DataFrame:

**.iloc[ ]** and **.loc[ ]**

**.iloc[ ]** uses the numeric indexes of a dataframe's rows and columns to return specific values

There are several possible ways to use **.iloc[ ]**

The general structure is: **.iloc[*rows you want*, *columns you want*]**

* Use single values to just get one row/column
* Use a colon (:) to get all rows/columns
* Use a list to get specific rows/columns
* Use a range(x:y) to get a range of rows/columns

# To return ALL ROWS and COLUMN 2

purchase\_df.iloc[ : , 1]

# The colon before the comma in .iloc[] means we want ALL rows

# The 1 after the comma means we want the column at index 1

# To return ROWS 1 THROUGH 4 (including 4), and ALL COLUMNS

purchase\_df.iloc[0:4, : ]

# To return ROWS 2, 3, AND 5, and COLUMNS 2 THROUGH 4 (including 4)

purchase\_df.iloc[[1, 2, 4], 1:4]

**.loc[ ]** uses the named indexes of a dataframe's rows and columns to return specific values.

The general structure for .loc[] is the same as that for .iloc[], except named indexes are used instead of numeric indexes   
A column's named index is simply its **column name**   
By default, when we create a dataframe, a row's index is numeric and starts at 0. You can set a named index for a dataframe's rows by using **.set\_index()**

# Set the row index to be the first\_name

credit\_df = credit\_df.set\_index("first\_name")

credit\_df

# Now, we can filter this dataframe using .loc[]

# Return data for James' and Tyler's rows, ALL COLUMNS included

credit\_df.loc[["James", "Tyler"], : ]

# Return rows from Bill to Matt (including Matt), and only the age and credit\_score columns

credit\_df.loc["Bill":"Matt", ["age", "credit\_score"]]

# Return all rows, only the credit\_score column

credit\_df.loc[ : , "credit\_score"]

2.1 Loc & iloc

import pandas as pd

file = "Resources/sampleData.csv"

df\_original = pd.read\_csv(file)

df\_original.head()

# Set new index to last\_name

df = df\_original.set\_index("last\_name")

df.head()

# 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)

# 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)

# The following will select all rows for columns `first\_name` and `Phone Number`

df.loc[:, ["first\_name", "Phone Number"]].head()

# the following logic test/conditional statement returns a series of boolean values

named\_billy = df["first\_name"] == "Billy"

named\_billy.head()

# 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)

2.2 Good movies

# Dependency

import pandas as pd

# Load in file

movie\_file = "Resources/movie\_scores.csv"

# Read and display the CSV with Pandas

movie\_file\_pd = pd.read\_csv(movie\_file)

movie\_file\_pd.head()

# List all the columns in the table

movie\_file\_pd.columns

# 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()

# 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()

# 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()

# 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)

2.3 Cleaning Data

# Dependencies

import pandas as pd

import numpy as np

# Name of the CSV file

file = 'Resources/donors2008.csv'

# The correct encoding must be used to read the CSV in pandas

df = pd.read\_csv(file, encoding="ISO-8859-1")

# Preview of the DataFrame

# Note that FIELD8 is likely a meaningless column

df.head()

# Delete extraneous column

del df['FIELD8']

df.head()

# Identify incomplete rows

df.count()

# Drop all rows with missing information

df = df.dropna(how='any')

# Verify dropped rows

df.count()

# The Amount column is the wrong data type. It should be numeric.

df.dtypes

# Use pd.to\_numeric() method to convert the datatype of the Amount column

df['Amount'] = pd.to\_numeric(df['Amount'])

# Verify that the Amount column datatype has been made numeric

df['Amount'].dtype

# Display an overview of the Employers column

df['Employer'].value\_counts()

# Clean up Employer category. Replace 'Self Employed' and 'Self' with 'Self-Employed'

df['Employer'] = df['Employer'].replace(

{'Self Employed': 'Self-Employed', 'Self': 'Self-Employed'})

# Verify clean-up.

df['Employer'].value\_counts()

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

df['Employer'].value\_counts()

# Display a statistical overview

# We can infer the maximum allowable individual contribution from 'max'

df.describe()

2.4 Portland Crime

# Import Dependencies

import pandas as pd

# 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.head()

# look for missing values

crime\_df.count()

# drop null rows

no\_null\_crime\_df = crime\_df.dropna(how='any')

# verify counts

no\_null\_crime\_df.count()

# Check to see if there are any values with mispelled or similar values in "Offense Type"

no\_null\_crime\_df["Offense Type"].value\_counts()

# Combining similar offenses together

no\_null\_crime\_df["Offense Type"] = no\_null\_crime\_df["Offense Type"].replace(

{"Commercial Sex Acts": "Prostitution", "Assisting or Promoting Prostitution": "Prostitution"})

no\_null\_crime\_df

# 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

2.5 Pandas Recap – UFOs

# Import the Pandas library

import pandas as pd

# Create a reference the CSV file desired

csv\_path = "Resources/ufoSightings.csv"

# Read the CSV into a Pandas DataFrame

ufo\_df = pd.read\_csv(csv\_path)

# Print the first five rows of data to the screen

ufo\_df.head()

# Check to see if there are any rows with missing data

ufo\_df.count()

# Remove the rows with missing data

clean\_ufo\_df = ufo\_df.dropna(how="any")

clean\_ufo\_df.count()

# 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", :]

usa\_ufo\_df

# Count how many sightings have occured within each state

state\_counts = usa\_ufo\_df["state"].value\_counts()

state\_counts

# Convert the state\_counts Series into a DataFrame

state\_ufo\_counts\_df = pd.DataFrame(state\_counts)

state\_ufo\_counts\_df.head()

# 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()

# 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

# Using to\_numeric() to convert a column's data into floats

usa\_ufo\_df["duration (seconds)"] = pd.to\_numeric(

usa\_ufo\_df["duration (seconds)"])

usa\_ufo\_df.dtypes

# Now it is possible to find the sum of seconds

usa\_ufo\_df["duration (seconds)"].sum()

\*\*\* 2.6 Group By

SAME START AS 2.5…AND A LITTLE MORE CROSSOVER, BUT DIFFERENT…

# Remove the rows with missing data

clean\_ufo\_df = ufo\_df.dropna(how="any")

clean\_ufo\_df.count()

# Converting the "duration (seconds)" column's values to numeric

clean\_ufo\_df["duration (seconds)"] = pd.to\_numeric(

clean\_ufo\_df["duration (seconds)"])

# 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", :]

usa\_ufo\_df.head()

# Count how many sightings have occured within each state

state\_counts = usa\_ufo\_df["state"].value\_counts()

state\_counts.head()

# 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)

# 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()

# 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()

# It is also possible to group a DataFrame by multiple columns

# This returns an object with multiple indexes, however, which can be harder to deal with

grouped\_international\_data = clean\_ufo\_df.groupby(['country', 'state'])

grouped\_international\_data.count().head(20)

# Converting a GroupBy object into a DataFrame

international\_duration = pd.DataFrame(

grouped\_international\_data["duration (seconds)"].sum())

international\_duration.head(10)

\*\*\* 2.7 Par\_Pokemon

# Dependencies

import pandas as pd

import numpy as np

# Save file path to variable

pokemon = "Resources/Pokemon.csv"

# Read with Pandas

pokemon\_pd = pd.read\_csv(pokemon)

pokemon\_pd.head()

# Create new table

pokemon\_type = pokemon\_pd[["Type 1", "HP", "Attack",

"Defense", "Sp. Atk", "Sp. Def", "Speed"]]

pokemon\_type.head()

# Create the GroupBy object based on the "Type 1" column

pokemon\_group = pokemon\_type.groupby(["Type 1"])

# Calculate averages for combat stats using the .mean() method

pokemon\_comparison = pokemon\_group.mean()

pokemon\_comparison

# Total number of points

pokemon\_comparison["Total"] = pokemon\_comparison["HP"] + pokemon\_comparison["Attack"] + pokemon\_comparison["Defense"] + \

pokemon\_comparison["Sp. Atk"] + \

pokemon\_comparison["Sp. Def"] + pokemon\_comparison["Speed"]

pokemon\_comparison["Total"]

# Sort by strongest Pokemon, and reset index

strongest\_pokemon = pokemon\_comparison.sort\_values(["Total"], ascending=False)

strongest\_pokemon.reset\_index(inplace=True)

strongest\_pokemon

# Save output to Excel

pokemon\_comparison.to\_excel("output/pokemon\_rankings.xlsx", index=False)

2.8 Sorting

# Import Dependencies

import pandas as pd

csv\_path = "Resources/Happiness\_2017.csv"

happiness\_df = pd.read\_csv(csv\_path)

happiness\_df.head()

# 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()

# To sort from highest to lowest, ascending=False must be passed in

freedom\_df = happiness\_df.sort\_values("Freedom", ascending=False)

freedom\_df.head()

# It is possible to sort based upon multiple columns

family\_and\_generosity = happiness\_df.sort\_values(

["Family", "Generosity"], ascending=False)

family\_and\_generosity.head()

# 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()

\*\*\* 2.9 Search for the worst

# Import Dependencies

import pandas as pd

import numpy as np

# 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.head ()

# Collect a list of all the unique values in "Preferred Position"

soccer\_2018\_df["Preferred Position"].unique()

# Looking only at strikers (ST) to start

strikers\_2018\_df = soccer\_2018\_df.loc[soccer\_2018\_df["Preferred Position"] == "ST", :]

strikers\_2018\_df.head()

# 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()

# Save all of the information collected on the worst striker

worst\_striker = strikers\_2018\_df.loc[0, :]

worst\_striker

3.1 Merging

# Dependencies

import pandas as pd

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

# 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

# Merge two dataframes using an INNER (default...now how =) join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id")

merge\_table

# Merge two dataframes using an OUTER join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id", how="outer")

merge\_table

# Merge two dataframes using a LEFT join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id", how="left")

merge\_table

# Merge two dataframes using a RIGHT join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id", how="right")

merge\_table

\*\*\* 3.2 Cryptocurrency

# Import Dependencies

import pandas as pd

bitcoin\_csv = "Resources/bitcoin\_cash\_price.csv"

dash\_csv = "Resources/dash\_price.csv"

bitcoin\_df = pd.read\_csv(bitcoin\_csv)

dash\_df = pd.read\_csv(dash\_csv)

bitcoin\_df.head()

dash\_df.head()

# Merge the two DataFrames together based on the Dates they share

crypto\_df = pd.merge(bitcoin\_df, dash\_df, on="Date")

crypto\_df.head()

# 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()

# 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()

# 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)

# 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

3.3 Binning

# Import Dependencies

import pandas as pd

raw\_data = {'Class': ['Oct', 'Oct', 'Jan', 'Jan', 'Oct', 'Jan'],

'Name': ["Cyndy", "Logan", "Laci", "Elmer", "Crystle", "Emmie"],

'Test Score': [90, 56, 72, 88, 98, 67]}

df = pd.DataFrame(raw\_data)

df

# Create the bins in which Data will be held

# Bins are 0, 60, 70, 80, 90, 100

bins = [0, 59, 69, 79, 89, 100]

# Create the names for the four bins

group\_names = ["F", "D", "C", "B", "A"]

df["Test Score Summary"] = pd.cut(df["Test Score"], bins, labels=group\_names)

df

# Creating a group based off of the bins

df = df.groupby("Test Score Summary")

df.max()

3.4 Ted Talks

# Import Dependencies

import pandas as pd

# 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()

# Figure out the minimum and maximum views for a TED Talk

print(ted\_df["views"].max())

print(ted\_df["views"].min())

# Create bins in which to place values based upon TED Talk views

bins = [0, 200000, 400000, 600000, 800000, 1000000,

2000000, 3000000, 4000000, 5000000, 50000000]

# Create labels for these bins

group\_labels = ["0 to 200k", "200k to 400k", "400k to 600k", "600k to 800k", "800k to 1mil", "1mil to 2mil", "2mil to 3mil", "3mil to 4mil", "4mil to 5mil", "5mil to 50mil"]

# Slice the data and place it into bins

pd.cut(ted\_df["views"], bins, labels=group\_labels).head()

# 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()

# 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()

3.5 Mapping / Formatting

import pandas as pd

# Mapping lets you format an entire DataFrame

file = "Resources/sample\_data.csv"

file\_df = pd.read\_csv(file)

file\_df.head()

# 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()

# Mapping has changed the datatypes of the columns to strings

file\_df.dtypes

\*\*\* 3.6 Cleaning Kickstarter

import pandas as pd

# The path to our CSV file &

# Read our Kickstarter data into pandas

file = "Resources/KickstarterData.csv"

df = pd.read\_csv(file)

df.head()

# Get a list of all of our columns for easy reference

df.columns

# 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.head()

# Remove projects that made no money at all

reduced\_kickstarter\_df = reduced\_kickstarter\_df.loc[(reduced\_kickstarter\_df["pledged"] > 0)]

reduced\_kickstarter\_df.head()

# Collect only those projects that were hosted in the US

hosted\_in\_us = reduced\_kickstarter\_df.loc[reduced\_kickstarter\_df["country"] == "US"]

hosted\_in\_us.head()

# 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']

# 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)

hosted\_in\_us.head()

# Calculate the total number of backers for all US projects

hosted\_in\_us["backers\_count"].sum()

# Calculate the average number of backers for all US projects

hosted\_in\_us["backers\_count"].mean()

# 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.head()

# 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()

3.7 Intro. to Bug Fixing

# Import dependencies

import pandas as pd

# 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)

chocolate\_ratings\_df.columns

# 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()

3.8 Bug Fixing Bonanza

# Import Dependencies

import pandas as pd

# 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)

eclipse\_df.head()

# Get a reference to the column names

eclipse\_df.columns

# Removing the newlines from column headers

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

# Finding the average number of comments per bug

average\_comments = eclipse\_df["Number of Comments"].mean()

average\_comments

# Grouping the DataFrame by "Assignee"

assignee\_group = eclipse\_df.groupby("Assignee")

# Count how many of each component Assignees worked on and create DataFrame of the data

assignee\_work = pd.DataFrame(assignee\_group["Component"].value\_counts())

assignee\_work.head()

# Rename the "Component" column to "Component Bug Count"

assignee\_work = assignee\_work.rename(columns={"Component": "Component Bug Count"})

assignee\_work.head()

# Find the percentage of bugs overall fixed by each Assignee

total\_bugs = eclipse\_df["Assignee"].count()

bugs\_per\_user = assignee\_group["Assignee"].count()

user\_bug\_percent = pd.DataFrame((bugs\_per\_user/total\_bugs)\*100)

user\_bug\_percent.head()

# 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()

# Reset the index of "assignee\_group" so that "Assignee" and "Component" are columns

assignee\_work = assignee\_work.reset\_index()

assignee\_work.head()

# Merge the "Percent of Total Bugs Assigned" into the DataFrame

assignee\_work = assignee\_work.merge(user\_bug\_percent, on="Assignee")

# Remove the extra columns

assignee\_work = assignee\_work[["Assignee", "Percent of Total Bugs Assigned",

"Component", "Component Bug Count"]]

assignee\_work.head()

## Objectives FROM “StudentGuide.md”

\* Be able to serve Jupyter notebook files from local directories and connect to their development environment.

\* Be able to create Pandas DataFrames from scratch.

\* Understand how to run functions on Pandas DataFrames.

\* Know how to read/write DataFrames from/to CSV files using Pandas.

\* Understand how to navigate through DataFrames using Loc and Iloc.

\* Understand how to filter and slice Pandas DataFrames.

\* Understand how to create and access Pandas GroupBy objects.

\* Understand how to sort DataFrames.

\* Know how to merge DataFrames together whilst understanding the differences between inner, outer, left, and right merges.

\* Be able to slice data using the `cut()` method and create new values based upon a series of bins.

\* Feel more confident with fixing Python/Pandas bugs within Jupyter Notebook.

\* Be able to use Google to explore additional Pandas functionality when necessary.

## Helpful Links & ## Additional Course Resources

\* [Formatting](https://pyformat.info/)

\* [Pandas Tutorials](https://chrisalbon.com/)

**\*\*\* [Pandas Documentation](http://pandas.pydata.org/)**

\* [Visual Guide to Joins](https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/)

\* [Pandas Merging](https://pandas.pydata.org/pandas-docs/stable/merging.html)

\* [Pandas Summary Notebooks](Supplemental/)

\* [Pandas Cheatseet](https://www.dataquest.io/blog/pandas-cheat-sheet/)