Welcome to Class

Friday, October 4, 2019 8:33 AM

Overview

Today's lesson is split into two parts. The first part will test the class' Pandas skills by looking through buggy code and fixing the problems so it functions properly. The second part will require students to use all the tools they have learned this week to fully understand the concept of data engineering

Class Objectives

- Understanding 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 in their ability to fix Python/Pandas bugs within Jupyter Notebook.
- Be able to use Google to explore additional Pandas functionality when necessary.

Pandas Recap and Data Types

Sunday, September 29, 2019 8:45 PM

Open PandasRecap.ipynb

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- Count sightings per state
  - Value_counts()

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

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

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

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

	state
ca	8683
fl	3754
wa	3707
tx	3398
ny	2915

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

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

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

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

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

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

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

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

Pandas Grouping

Sunday, September 29, 2019 8:41 PM

Open GroupBy.ipynb

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

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

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

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

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000213EE41CE10>
```

state	datetime	city	country	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
ak	311	311	311	311	311	311	311	311	311	311
al	629	629	629	629	629	629	629	629	629	629
ar	578	578	578	578	578	578	578	578	578	578

- It is possible to create new DataFrames using purely GroupBy data. This can by done by taking the **`pd.DataFrame()`** method and passing the GroupBy data desired in as the parameter.
- A DataFrame can also be created by selecting a single series from a GroupBy object and passing it in as the values for a specified column.

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

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

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

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

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

list parameter

```
# It is also possible to group a DataFrame by multiple columns
# This returns an object with multiple indexes, however, which can be harder to deal with
grouped_international_data = clean_ufo_df.groupby(['country','state'])
grouped_international_data.count().head(20)
```

		datetime	city	shape	duration (seconds)	duration (hours/min)	comments	date posted	latitude	longitude
country	state									
au	al	1	1	1	1	1	1	1	1	1
	dc	1	1	1	1	1	1	1	1	1
	nt	2	2	2	2	2	2	2	2	2
	oh	1	1	1	1	1	1	1	1	1
	sa	2	2	2	2	2	2	2	2	2
	wa	2	2	2	2	2	2	2	2	2
	yt	1	1	1	1	1	1	1	1	1

- A new DataFrame can be created from a GroupBy object

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

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

TEAM UP: Building a PokeDex**

Sunday, September 29, 2019 8:42 PM

Overview

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

Files

pokemon.ipynb
pokemon.csv

Instructions

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

Review: Building a PokeDex

Sunday, September 29, 2019 8:44 PM

Keys

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

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

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

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

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

pokemon_comparison = pokemon_group.mean()
pokemon_comparison
```

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

- Calculate the total power level of each type of Pokemon by summing all of

the previous stats together and place the results into a new column.

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

pokemon_comparison["Total"]
```

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

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

Sorting Made Easy

Sunday, September 29, 2019 8:44 PM

Open Sorting.ipynb

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

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

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

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

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

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

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

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

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

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

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

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

Exercise: Search for the Worst

Sunday, September 29, 2019 8:45 PM

Overview

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

Files

SearchForTheWorst.ipynb
Soccer2018Data.csv

Instructions

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

Review: Search for the Worst

Monday, September 30, 2019 11:24 PM

Keys

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

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

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

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

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

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

5 rows × 33 columns

- Sort the DataFrame based upon a player's skill in that position.
- Reset the index for the DataFrame so that the index is in order.

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

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

strikers_2018_df.head()
```

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

5 rows × 33 columns

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

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

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

Merging Dataframes

Friday, October 4, 2019 9:21 AM

Sometimes, the data an analyst is provided with is split into multiple parts. Obviously, it is far more preferable to work with a single dataset than it is to work with a bunch of different datasets.

This is where the concept of merging comes into play, as Pandas allows its users to combine separate DataFrames on similar values using the **pd.merge()** method

Open Merging.ipynb

- Creating two DataFrames which contain information on customers and the purchases they have made.
- These two DataFrames share the "customer_id" column

```
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
```

	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

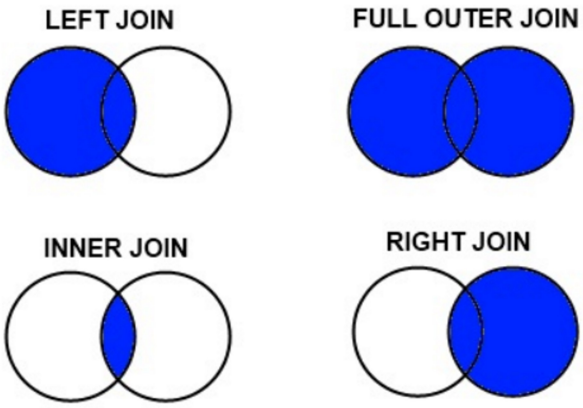
```
# Create DataFrames
raw_data_items = {
    "customer_id": [403, 112, 543, 999, 654],
    "item": ["soda", "chips", "TV", "Laptop", "Cooler"],
    "cost": [3.00, 4.50, 600, 900, 150]
}
items_pd = pd.DataFrame(raw_data_items, columns=[
                        "customer_id", "item", "cost"])
items_pd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

	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

- **`pd.merge()`** method is used and three parameters are passed into it: references to both of the DataFrames and the value `on="customer_id"`.
- This combines the two DataFrames together so that, whenever the "customer_id" column matches, the rows containing the matching data are joined.

```
# Merge two dataframes using an inner join
merge_table = pd.merge(info_pd, items_pd, on="customer_id")
merge_table
```

	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



- How='Outer'

```
# Merge two dataframes using an outer join
merge_table = pd.merge(info_pd, items_pd, on="customer_id", how="outer")
merge_table
```

	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

- How='left'

```
# Merge two dataframes using a left join
merge_table = pd.merge(info_pd, items_pd, on="customer_id", how="left")
merge_table
```

	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

- How='right'

```
# Merge two dataframes using a right join
merge_table = pd.merge(info_pd, items_pd, on="customer_id", how="right")
merge_table
```

	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

Exercise: Cryptocurrency Merging

Friday, October 4, 2019 9:24 AM

Overview

Students will take some time to merge together two datasets on cryptocurrencies, one on Bitcoin and the other on Dash. They will then clean up the combined DataFrame to make it more presentable.

Files

Cryptocurrency.ipynb
Bitcoin_cash_price.csv
Dash_price.csv

Instructions

- Read in both of the CSV files and print out their DataFrames.
- Perform an inner merge that combines both DataFrames on the "Date" column.
- Rename the columns within the newly merged DataFrame so that the headers are more descriptive.
- Create a summary table that includes the following information: Best Bitcoin Open, Best Dash Open, Best Bitcoin Close, Best Dash Close, Total Bitcoin Volume, Total Dash Volume.
- Total Bitcoin Volume and Total Dash Volume should be calculated to have units of "millions" and be rounded to two decimal places.

Review: Cryptocurrency Merging

Friday, October 4, 2019 9:25 AM

Keys

- Read in both of the CSV files and print out their DataFrames.

```
# 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()
```

	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

```
dash_df.head()
```

	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

- Perform an inner merge that combines both DataFrames on the "Date" column.
- There is an issue with columns headers when first merge our DataFrames. This is because the columns within the first DataFrame match those within the second and Pandas feels the need to differentiate them somehow.

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

	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

- Rename the columns within our newly merged DataFrame so that the headers are more descriptive.

```
# 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 Mark

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()
```

	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

- Alternatively, you can set your suffix using the suffixes parameter in the **merge( )** function

```
# 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()
```

	Date	Open_Bitcoin	High_Bitcoin	Low_Bitcoin	Close_Bitcoin	Volume_Bitcoin	Market Cap_Bitcoin	Open_Dash	High_Dash	Low_Dash	Close_Dash	Volume
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	3808
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	4370
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	7269
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	3501
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	2832

- Creating a summary table that includes : Best Bitcoin Open, Best Dash Open, Best Bitcoin Close, Best Dash Close, Total Bitcoin Volume, Total Dash

Volume.

- To find the total volumes for Bitcoin and Dash, our code divides the sum of their columns by one million and then rounds the returned value to the nearest two decimal places.
- The values for our summary DataFrame are held within brackets because, without them, Pandas would have difficulties understanding that all of these values should be held within the same row.

```
# 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
```

	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

Binning Data

Friday, October 4, 2019 9:25 AM

Pandas has a built-in "binning" method that allows its users to place values into groups so as to allow for more vigorous customization of datasets

Open Binning.ipynb

- When using the **`pd.cut()`** method, three parameters must be passed in.
    - Series that is going to be cut.
    - List of the bins that the Series will be sliced into.
    - List of the names/values that will be given to the bins.
- When creating the list for bins, Pandas will automatically determine the range between values.
    - For example, when given the list `[0, 59, 69, 79, 89, 100]`, Pandas will create bins with ranges between those values in the list.
- The labels for the **`pd.cut()`** method must have an equal length to the number of bins.

```
# 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"]
```

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

	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

- After creating and applying these bins, the DataFrame can be grouped according to those values

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

	Test Score
Test Score Summary	
F	59
D	60
C	72
B	88
A	94

Exercise: Binning TED

Friday, October 4, 2019 9:25 AM

Overview

The class will now put their binning skills to the test by creating bins for TED Talks based upon their viewership. After creating the bins, they will then group the DataFrame based upon those bins and perform some analysis on them

Files

BinningTed.ipynb
Ted_talks.csv

Instructions

- Read in the CSV file provided and print it to the screen.
- Find the minimum "views" and maximum "views".
- Using the minimum and maximum "views" as a reference, create 10 bins in which to slice the data.
- Create a new column called "View Group" and fill it with the values collected through your slicing.
- Group the DataFrame based upon the values within "View Group".
- Find out how many rows fall into each group before finding the averages for "comments", "duration", and "languages".

Review: Binning TED

Friday, October 4, 2019 9:26 AM

Keys

- Read in the CSV file provided and print it to the screen.

```
# 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()
```

	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

- Find the minimum "views" and maximum "views".

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

```
47227110
50443
```

- Using the minimum and maximum "views" as a reference, create 10 bins in which to slice the data.

```
# 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"]
```

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

```
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 5
mil < 5mil to 50mil]
```

- Create a new column called "View Group" and fill it with the values collected through your slicing.

```
# 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()
```

	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

- Group the DataFrame based upon the values within "View Group".
- Find out how many rows fall into each group before finding the averages for "comments", "duration", and "languages".

```
# 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
```

View Group	comments	duration	languages
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

Mapping

Friday, October 4, 2019 9:26 AM

Recall how Excel's number formats allows its users to change the styling of columns. Pandas also includes this functionality through its **df.map()** method, which allows users to style columns

Open Mapping.ipynb

- **df[<COLUMN>].map(<FORMAT STRING>.format)** is the method by which users can modify the styling of an entire column.
- The formatting syntax used for mapping is, in a word, confusing. It uses strings containing curly brackets in order to determine how to style columns and this can make it rather difficult to understand at first glance.
- A somewhat easy way to understand mapping strings is that it is almost akin to concatenating strings. Whatever is outside of the curly brackets is added before/after the initial value which is modified by whatever is contained within the curly brackets.
- So, to convert values into a typical dollar format, one would use "**${:.2f}**". This places a dollar sign before the value which has been rounded to two decimal points.
- Using "**{:,}**" will split a number up so that it uses comma notation. For example: the value 2000 would become 2,000 using this format string.

```
# 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()
```

	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

- Format mapping only really works once and will return errors if the same code is run multiple times without restarting the kernel. Because of this, formatting is usually applied near the end of an application.
- Format mapping also can change the datatype of a column. As such, all calculations should be handled before modifying the formatting.

Exercise: Cleaning Kickstarter

Friday, October 4, 2019 9:26 AM

Overview

The class will now spend the remainder of the lesson flexing their Pandas muscles by taking a dataset similar to that of their first homework, cleaning it up, and formatting it in far less time than it would take in Excel

Files

KickStarterClean.ipynb
KickstarterData.csv

Instructions

```
import pandas as pd
```

```
# The path to our CSV file

# Read our Kickstarter data into pandas
```

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

```
# Extract "name", "goal", "pledged", "state", "country", "staff_pick",
# "backers_count", and "spotlight"
```

```
# Remove projects that made no money at all
```

```
# Collect only those projects that were hosted in the US
# Create a list of the columns
# Create a new df for "US" with the columns above.
```

```
# Create a new column that finds the average amount pledged to a project
```

```
# 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
```

```
# Calculate the total number of backers for all US projects
```

```
# Calculate the average number of backers for all US projects
```

```
# Collect only those US campaigns that have been picked as a "Staff Pick"
```

```
# Group by the state of the campaigns and see if staff picks matter (Seems to matter quite a bit)
```

Review: Cleaning Kickstarter

Friday, October 4, 2019 9:28 AM

Intro to Bugfixing

Friday, October 4, 2019 9:29 AM

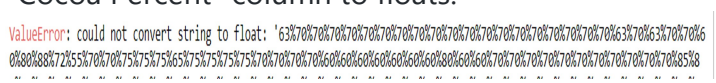
Ope IntroToBugFixing_Unsolved

In this code an error is being returned as the application attempts to collect the average value within the "Cocoa Percent" column

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
F:\Program Files (x86)\Anaconda\envs\PythonData\lib\site-packages\pandas\core\nanops.py in _ensu
re_numeric(x)
    767         try:
--> 768             x = float(x)
    769         except Exception:

ValueError: could not convert string to float: '63%70%70%70%70%70%70%70%70%70%70%70%70%70%70%
70%70%70%70%63%70%63%70%70%60%80%88%72%55%70%70%75%75%75%65%75%75%75%75%70%70%70%70%60%60%60%60%
60%60%60%80%60%60%70%70%70%70%70%70%70%70%70%70%85%85%72%73%64%66%75%63%70%68%70%70%75%70%70%
70%70%70%70%70%63%70%66%75%85%50%75%60%75%75%75%72%75%75%70%70%73%70%70%70%70%70%70%70%70%73%
70%68%70%70%70%70%75%70%75%72%72%72%72%100%72%72%72%72%75%72%72%80%75%72%72%72%68%72%70%77%75%70
%80%70%70%70%70%70%70%70%70%70%70%80%65%70%65%73%72%80%70%70%90%64%64%64%71%70%70%70%83%78%83
%74%74%74%73%72%72%55%64%88%72%72%70%74%64%72%76%76%78%86%72%75%70%65%70%78%75%65%75%65%71%75%68
%70%70%70%70%70%82%72%82%75%75%75%70%70%75%75%75%65%75%75%75%75%75%75%75%75%75%75%75%75%75%75%75
%75%75%75%75%75%100%75%75%77%100%70%70%70%70%68%70%70%72%70%75%85%60%80%70%80%80%60%70%72%70%72%
68%70%68%72%72%72%72%60%70%75%75%75%75%65%70%75%72%66%77%75%75%74%75%70%74%71%74%72%64%70%69%70%
```

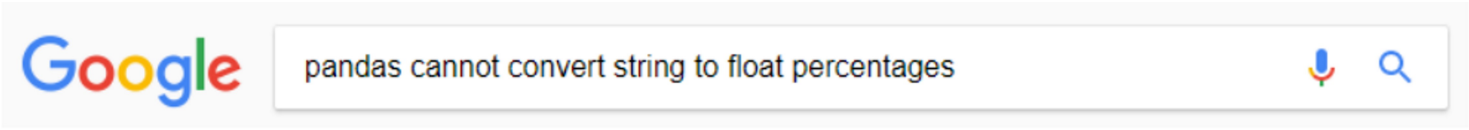
- The first step in fixing a bug is to keep calm.
  - Bugs happen all the time and they are rarely the end of the world. In fact, most bugs that programmers run across are simple enough to solve so long as they know how and where to look for the solution.
- The second step to bugfixing an application is to figure out what the bug is and where it is located.
  - Since the class is using Jupyter Notebook at this point in time, it is quite easy to find the erroneous block of code since the error will always be returned in the space beneath the erroneous cell.
  - Unfortunately , Pandas it often returns large blocks of text that is complex and confusing to those who do not know the library's underlying code.
  - Looking for the line following `KeyError:` is generally a good starting point.
    - For example, the text following `TypeError:` within the current code lets the programmer know that Pandas cannot convert the string values in the "Cocoa Percent" column to floats.

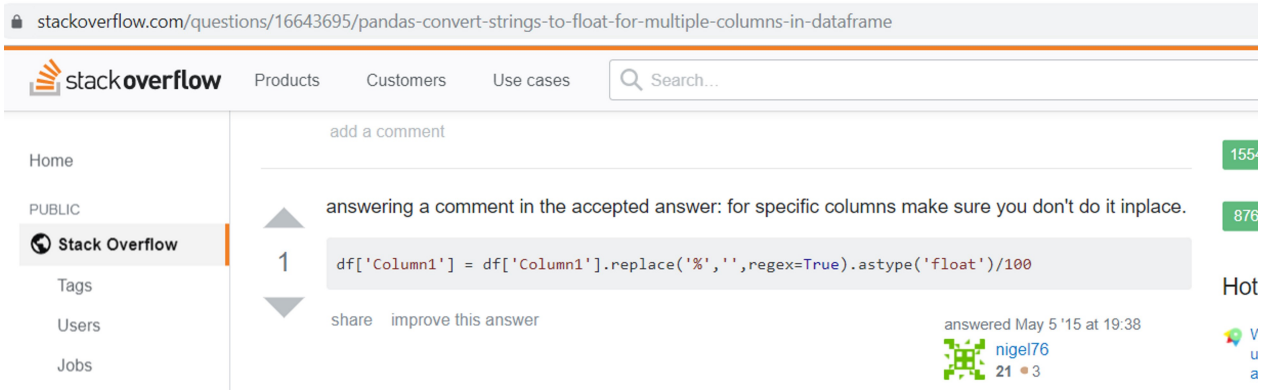
      

    - If the error text is not entirely clear, it is oftentimes helpful to print out variables/columns to the console in order to uncover where the bug is. For

example, printing out the "Cocoa Percent" series lets the programmer know that the `dtype` of this series is an object and not a float.

- The third step is to look up the error online and search for solutions that other programmers have uncovered.
    - The key part to this step is coming up with an accurate way to describe the bug. This may take multiple tries and is a skill that will develop over time.
    - Google is the programmer's best friend as typing in a description of the bug being faced will often bring up links to some possible solutions. If not, simply change the search up a little bit until a solution is discovered.
    - This particular problem requires the code to drop the percentages within the "Cocoa Percent" column, so the search should be a bit more specific






```
# 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()
```

```
71.6983286908078
```

Exercise: Bugfixing Bonanza!

Friday, October 4, 2019 9:30 AM

Overview

The class will now be provided with a Pandas project containing TONS of bugs inside of it. Their job will be to take the application and fix it up so that it works properly. This will both put their Pandas skills to the test while also teaching them how to teach themselves

Files

BugFixBonanza.ipynb
EclipseBugs.csv

Instructions

- Dig through the Jupyter Notebook provided and attempt to fix as many bugs as possible. There are a lot of them and the bugs get harder to deal with as the code progresses.
- Once you have finished bugfixing, perform some additional analysis on the dataset provided. See what interesting trends are buried deep within these bug logs for the Eclipse IDE. So long as you challenge yourself, bugs will pop up and you will get even more bugfixing practice

Review: Bugfixing Bonanza

Friday, October 4, 2019 9:32 AM