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)

```
# 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

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

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
```

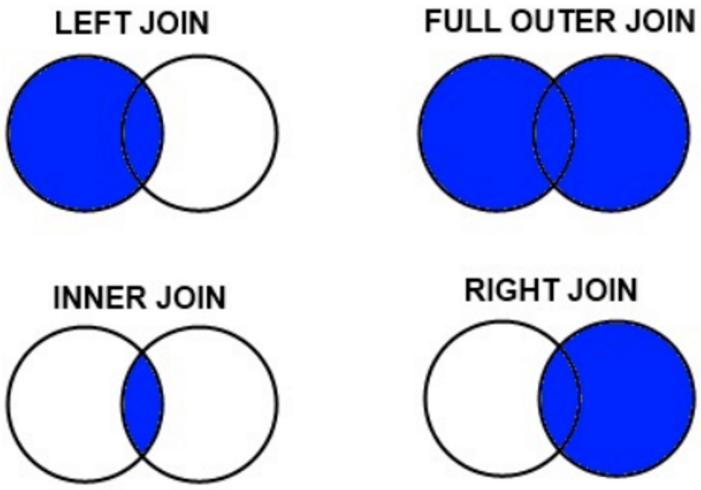
	customer_id	item	cost
0	403	soda	3.0
1	112	chips	4.5
2	543	TV	600.0
3	999	Laptop	900.0
4	654	Cooler	150.0

- **`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

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

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.

Intro to Bugfixing

Friday, October 4, 2019 9:29 AM

Open 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%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%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%
```

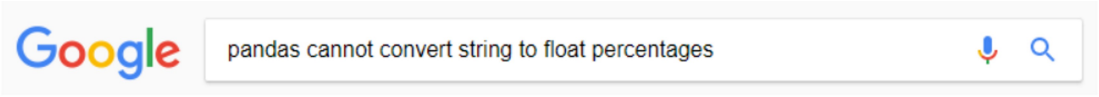
- The first step in fixing a bug is to keep calm.
    - Bugs happen all the time and they are rarely the end of the world. In fact, most bugs that programmers run across are simple enough to solve so long as they know how and where to look for the solution.
- The second step to bugfixing an application 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

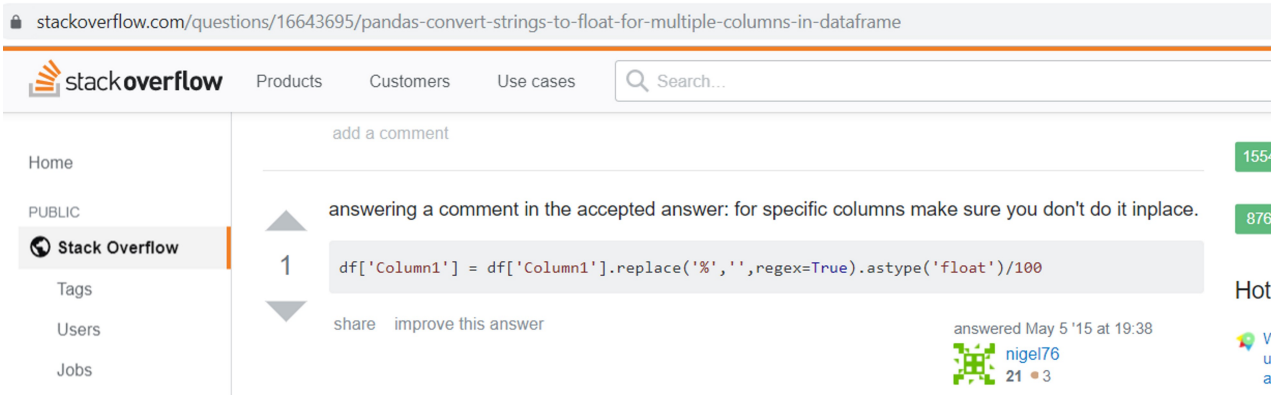
lets the programmer know that Pandas cannot convert the string values in the "Cocoa Percent" column to floats.

```
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%70%70%63%70%63%70%70%6
0%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%70%85%8
```

- 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
```