Welcome to the Data Manipulation Lesson.

Notebook 1

In this lesson, we will be using this workbook in tandum with the reading assignments.

The workbook has been broken up into three sections. Each section has reading assignments and is followed by questions and prompts for you to work through.

In your Canvas, you will find the reading quiz.

In [2]:
```
import pandas as pd
import numpy as np

data= pd.read_csv("titanic.csv")
```

Before You Get Started

We are going to be using the Titanic Dataset. Make sure to run a head() before you start working with manipulation methods.

In [45]:
```
# Run the head of your data set here:
data.head()
```

Out[45]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN

In [53]:
```
# ??
```

In [12]:
```
# if there are, go ahead and drop them:
data.describe()
```

Out[12]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Cleaning Note:

While the columns are not the "prettiest", don't adjust any of them yet. We are going to update some values and add some values as we workthrough this notebook. Applologies for the extra visual "noise" on your screen. You will be given the option to tidy up the columns at the end of this notebook.

Running Tables Note:

If your tables don't appear to have accepted your changes, try the "Run All" option in the "Cell" section of the menu bar.

A. Aggregation

1. Please read the following:
   - Python | Pandas dataframe.aggregate()
   - Python | Pandas dataframe.groupby()
2. Answer the Check Your Understanding Questions in your Canvas account.
3. Work through the section Exercises.
   - There are 4 sections in part A:
     - Groupby
     - Aggregation Methods
     - Groupby and Basic Math
     - Groupby and Multiple Aggregations

Creating Variables.

As we begin to manipulate our data, create new variables to store your work in. This will keep your original data in tact. Having the original dataset available will save you time with each manipulation. You can also create variable names that inform you of the purpose of the manipulation.

1: Groupby

Groupby "embark_town"

1. Using the titanic data set, groupby "embark_town".
2. Create a variable that will represent the grouping of data.
3. Intitalize it using the groupby() function and pass it the column.

In [46]:

```
# Code your groupby "embark_town" here:
embt = data.groupby("embark_town")
```

In [47]:

```
# To view the grouped data as a table, use the variable_name.first():
embt.first()
```

Out[47]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_
embark_town											
Cherbourg	1	1	female	38.0	1	0	71.2833	C	First	woman	
Queenstown	0	3	male	2.0	0	0	8.4583	Q	Third	man	
Southampton	0	3	male	22.0	1	0	7.2500	S	Third	man	

Groupby "survived"

Did you know that you can also chain on some of our exploratory methods to the groupby method?

1. Create & initalize a new variable to hold a table that will groupby "survived"
2. Use method chaining to tack on the describe method

In [48]:

```
# Code your groupby "survived" table here:
survived = data.groupby('survived')

# run your table below:

survived.first()
```

Out[48]:

	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	er
survived												
0	3	male	22.0	1	0	7.2500	S	Third	man	True	E	S
1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	

In [49]:

```
# run your table with describe
survived.describe()
```

Out[49]:

	pclass								age		...		fa
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**...**	**75%**	**m**
survived													
0	549.0	2.531876	0.735805	1.0	2.0	3.0	3.0	3.0	424.0	30.626179	...	26.0	263.00
1	342.0	1.950292	0.863321	1.0	1.0	2.0	3.0	3.0	290.0	28.343690	...	57.0	512.32

2 rows × 48 columns

In [50]:

```
# How is this table organized?  Why are there 40 columns now?

#There are 40 columns because besides survived, there are 5 numeric columns in "data,'
#calculating 8 values for each of those 5 columns. They are added as columns because g
#by two rows for survived or not.
```

2. Aggregation Methods

Note: **agg()** and **aggregate()** are identical source

Method Chaining

1. Create a variable to method chain **head()** and **agg()** togehter.
2. Pass one of the following statistical values to **agg()**
   - "mean", "median", "mode", "min", "max", "std", "var", "first", "last", "sum"

In [13]:

```
# Code your method chain here:
# "together" is misspelled up there ^^^
data.agg(['mean','min','max']).head()
```

```
C:\Users\Brown\AppData\Local\Temp\ipykernel_61052\327450567.py:3: FutureWarning: ['se
x', 'embarked', 'class', 'who', 'deck', 'embark_town', 'alive'] did not aggregate suc
cessfully. If any error is raised this will raise in a future version of pandas. Drop
these columns/ops to avoid this warning.
  data.agg(['mean','min','max']).head()
```

Out[13]:

	survived	pclass	sex	age	sibsp	parch	fare	class	who	adult_ma
mean	0.383838	2.308642	NaN	29.699118	0.523008	0.381594	32.204208	NaN	NaN	0.6026
min	0.000000	1.000000	female	0.420000	0.000000	0.000000	0.000000	First	child	Fa
max	1.000000	3.000000	male	80.000000	8.000000	6.000000	512.329200	Third	woman	Tr

In [14]:

```
# Create a variable to method chain head() with agg("sum")
data.agg('sum').head
# run your table:
```

```
C:\Users\Brown\AppData\Local\Temp\ipykernel_61052\2046074924.py:2: FutureWarning: Dro
pping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is depre
cated; in a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  data.agg('sum').head
```

Out[14]:

```
<bound method NDFrame.head of survived
342
pclass                                                      2057
sex           malefemalefemalefemalemalemalemalemalefemalefe...
age                                                     21205.17
sibsp                                                        466
parch                                                        340
fare                                                  28693.9493
class         ThirdFirstThirdFirstThirdThirdFirstThirdThirdS...
who           manwomanwomanwomanmanmanmanchildwomanchildchil...
adult_male                                                   537
alive         noyesyesyesnononononoyesyesyesyesnononoyesnoyesn...
alone                                                        537
dtype: object>
```

In [62]:

```
# Explain the sum table.  What is going on with the "sex", "class", and "alive" columr

# sum is running on strings so it is just concatenating them into one big disaster str
```

Using a Dictionary

A dictionary is a Python collection type.

Is a collection type that stores **key-value pairs**. A key-value pair is an orgainzation system that is made up of a single *key* that has one or more *values* paired with it.
Think of it like your contacts list. The contacts list is the dictionary object.
Each contact is organized by a key, usually name. And attached to each name is contact information, or the values. Some contacts might have email address, phone number, home or work address, etc. Other contacts may just be a name and phone number. This is a very simple example, but understanding this orgainzational structure will be helpful as you learn to manipulate tables.

Here is a dictionary example with 3 keys:

> **contacts_dictionary = {"name1": ["email", 555-5552, "work info"], "name2": ["email", 555-5554], "name3": 555-5555}**

Here is a dictionary example with a single key-value pair **study_group_dictionary = {"name1": 555-5557}**

It has a single key, and a list of values. The organization of this structure is called a "Key-Value Pair". Using the contact list example, the key would be the name of the person and the values would be their contact information. The key is a single item (the person's name) and the values can be a single item (an email address) or mulitple items (email, phone number, address, work info, etc). Keys and values can be any data type, but must use correct data type syntax. The keys do not have to be strings, but they do need to be a single value.

For more information, you can read more on dictionary objects here.

Aggregation across muliple columns using dictionary functionality

Syntax Example:

age_dictionary={"age":["sum", "max"]}

We are creating a new dictionary (**age_dictionary**). The key is **age** and the values we want are **"sum""** and **"max"**. This dictionary object has now become a tempate for the aggregations we want to preform. However, on it's own, it does nothing. Once passed to the **agg()** method, it will pick out the specific location of data we want to examine. Making a subset table.

The code is contained in the box below. Run it and see what happens.

For syntax examples, review this webpage.

- -

In [ ]:

```
In the above paragraph, "However, on it's own" should be "However, on its own" becaus
"However, on it is own." No apostrophe is the possessive form of "it." An apostrophe w
```

In [16]:

```
# Predict the table output before you uncomment the code below.

# it will print a table showing those two calculations for the age column

age_dictionary={"age":["sum", "max"]}
dictionary_agg =data.agg(age_dictionary)
dictionary_agg
```

Out[16]:

	age
sum	21205.17
max	80.00

1. What if we want to look at more than one column at a time? We pass more dictionaries to the agg function.
2. Create a variable to hold at least 3 columns. Use the syntax from the "Syntax Example" as a guide.
    - Aggregate the following: survived: "sum" & "count"; age: "std" & "min", and sibsp: "count" & "sum"

In [18]:

```
# Code your dictionary here:
mult_dictionary = {"survived":['sum', 'count'], "age":['std', 'min'], "sibsp":['count'
# passing the variable didn't work? but directly inputting the dictionary does...
dict_agg = data.agg({"survived":['sum', 'count'], "age":['std', 'min'], "sibsp":['cour
dict_agg
```

Out[18]:

	survived	age	sibsp
sum	342.0	NaN	466.0
count	891.0	NaN	891.0
std	NaN	14.526497	NaN
min	NaN	0.420000	NaN

3. Groupby and Basic Math

1. Groupby "pclass". Make sure you use a variable to hold your grouped data.

In [21]:

```
# Code your groupby here:

pclass_gb = data.groupby('pclass')

# Run your table using first() here instead of head():
pclass_gb.first()
```

Out[21]:

	survived	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
pclass											
1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	female	14.0	1	0	30.0708	C	Second	child	False	D
3	0	male	22.0	1	0	7.2500	S	Third	man	True	G

4. Groupby and Multiple Aggregations

Group with a List

1. We want to do muliple aggregation functions to our newly grouped data set. We created a variable to hold a list of functions we want to perform. These functions are part of the agg method. When we pass our list to the method, the method will iterate through each item and perform that function for the entire table.

In [22]:

```
# our list of functions
agg_func_list = ['sum', 'mean', 'median', 'min', 'max', 'std', 'var', 'first', 'last',


#Apply the agg method to our passenger_class variable (made in the Groupby Basic Math
# Pass our list to the function and run your table.
pclass_gb.agg(agg_func_list)
```

```
C:\Users\Brown\AppData\Local\Temp\ipykernel_61052\2407042617.py:7: FutureWarning: ['s
ex', 'embarked', 'class', 'who', 'deck', 'embark_town', 'alive'] did not aggregate su
ccessfully. If any error is raised this will raise in a future version of pandas. Dro
p these columns/ops to avoid this warning.
  pclass_gb.agg(agg_func_list)
```

Out[22]:

	survived										...		
	sum	mean	median	min	max	std	var	first	last	count	...	sum	mean
pclass													
1	136	0.629630	1.0	0	1	0.484026	0.234281	1	1	216	...	109	0.504630
2	87	0.472826	0.0	0	1	0.500623	0.250624	1	0	184	...	104	0.565217
3	119	0.242363	0.0	0	1	0.428949	0.183998	0	0	491	...	324	0.659878

3 rows × 70 columns

In [ ]:

Group with a Dictionary

Using only a list provides us with the entire table. What if we only want to look at age vs pclass?

we can create a dictionary to hold the age column for us. The *key* would be the name of our column, and the values our list of functions to preform on that column. The code would look like this:

In [ ]:

```
# it's spelled perform, not preform =)
```

In [25]:

```
agg_func_dict = {
    'age':
    ['sum', 'mean', 'median', 'min', 'max', 'std', 'var', 'first', 'last', 'count']
}
# We would run our table like this:
pclass_gb.agg(agg_func_dict)
```

Out[25]:

	age									
	sum	mean	median	min	max	std	var	first	last	count
pclass										
1	7111.42	38.233441	37.0	0.92	80.0	14.802856	219.124543	38.0	26.0	186
2	5168.83	29.877630	29.0	0.67	70.0	14.001077	196.030152	14.0	27.0	173
3	8924.92	25.140620	24.0	0.42	74.0	12.495398	156.134976	22.0	32.0	355

Looking at the *age_func_dict* syntax, create a dictionary variable for the "survived" column and pass it to **passenger_class.agg()** in the box below.

In [26]:

```
# Code it here:

agg_func_dict_svd = {
    'survived':
    ['sum', 'mean', 'median', 'min', 'max', 'std', 'var', 'first', 'last', 'count']
}
pclass_gb.agg(agg_func_dict_svd)
```

Out[26]:

	survived									
	sum	mean	median	min	max	std	var	first	last	count
pclass										
1	136	0.629630	1.0	0	1	0.484026	0.234281	1	1	216
2	87	0.472826	0.0	0	1	0.500623	0.250624	1	0	184
3	119	0.242363	0.0	0	1	0.428949	0.183998	0	0	491

B. Recoding and Creating New Values and Variables

1. Please read the following:
   A. How to create new columns derived from existing columns? 1.Recode Data
2. Answer the Check Your Understanding questions in your Canvas Account.
3. Work through the Part B, there are 2 sections

Suggested Reading:

- How to manipulate textual data?

Create a New Column

As questions arrise during your data exploring and cleaning, you might want to test them out. In this instance, we want to make sure the values we want to manipulate remain untouched. One thing we can do is to add a new column that will contain our manipulations.

In the box below:

1. Create a new column by manipulating the values of different column. Specifically, create a new column, "fare_2021" that allows us to compare the cost of fare in pounds back in 1912 to 2021. This website can help you find the 2021 fare amount.

In [28]:

```
# Code your new "fare_2021" column here:
data['fare_2021'] = data['fare']*134.99
# Run the head of your table to see your new column:
data.head()
```

Out[28]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN

Replacing Values

Replace the values in the "alive" coloum from string "yes" or "no" to bools, where "yes" becomes True and "no" becomes False.

In [34]:

```
# "coloum" is misspelled here
# Code your updated values here:
def alive_rename(series):
    if series == 'yes':
        return True
    elif series == 'no':
        return False
    else:
        return series

data['alive'] = data['alive'].apply(alive_rename)

data.head()
```

Out[34]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN

We can also use functions to update values.

1. Create a function that will set the alive values as bools. Apply it to your table and run your table here:

In [71]:

```
# Code your function here:
```

Using a function to create a new column

Sometimes you might want to create a new column based on combining multiple columns together.

1. create an "age_group" column that breaks years up as 0-19, 20-29, 30-39, etc until all given ages are covered. Make sure you check to see where you can stop counting by 10s.

In [36]:

```
# Write your max age check here:
max_age = {
    'age':
    ['max']}
data.agg(max_age)
```

Out[36]:

	age
max	80.0

In [39]:

```
# Code the new "age_group" column function here:

def age_groups(series):
    if series < 20:
        return "0-19 yrs"
    elif 20 <= series < 30:
        return "20-29 yrs"
    elif 30 <= series < 40:
        return "30-39 yrs"
    elif 40 <= series < 50:
        return "40-49 yrs"
    elif 50 <= series < 60:
        return "50-59 yrs"
    elif 60 <= series < 70:
        return "60-69 yrs"
    elif 70 <= series <= 80:
        return "70-80 yrs"


data['Age Group'] = data['age'].apply(age_groups)
data.head()
```

Out[39]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN

C. Reshaping Tables

1. Please read the following:
   A. How to reshape the layout of tables?
2. Answer the Check Your Understanding in your Canvas account
3. Work through Part C, there are 4 sections

Suggested Reading:

1. pandas.pivot_table
2. pandas.melt
3. pandas.pivot

Sort_values

Use **sort_values()** to answer the following question:

> What is the age of the person who paid the highest fare?

Hint: We want to see the highest fare value first. What order would we want? ascending or descending? Check the documentation for the syntax.

In [51]:
```
# Code your sort_values here:
data.sort_values("fare", axis=0, ascending=False, inplace=False, kind='quicksort', na_

# Run your table here:

# age of person with highest fare is 35 years old
```

Out[51]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
258	1	1	female	35.0	0	0	512.3292	C	First	woman	False
737	1	1	male	35.0	0	0	512.3292	C	First	man	True
679	1	1	male	36.0	0	1	512.3292	C	First	man	True
88	1	1	female	23.0	3	2	263.0000	S	First	woman	False
27	0	1	male	19.0	3	2	263.0000	S	First	man	True
...	...	...	...	...	...	...	...	...	...	...	...
633	0	1	male	NaN	0	0	0.0000	S	First	man	True
413	0	2	male	NaN	0	0	0.0000	S	Second	man	True
822	0	1	male	38.0	0	0	0.0000	S	First	man	True
732	0	2	male	NaN	0	0	0.0000	S	Second	man	True
674	0	2	male	NaN	0	0	0.0000	S	Second	man	True

891 rows × 17 columns

pivot_table

1. pivot the table of the summed data where the values are "fare", index is "who" and "age_group", and the columns are "survived"

Hint: set the aggfunc parameter to np.sum

In [54]:
```
# Code your pivot_table here:
pd.pivot_table(data, 'fare', ['who', 'Age Group'], 'survived')

# Run your table here:
```

Out[54]:

	survived	0	1
who	Age Group		
child	0-19 yrs	32.633703	32.891329
man	0-19 yrs	22.236930	29.106660
	20-29 yrs	19.798641	25.878168
	30-39 yrs	17.931892	74.823735
	40-49 yrs	24.680831	38.889942
	50-59 yrs	32.819392	56.550000
	60-69 yrs	45.114423	44.850000
	70-80 yrs	30.197233	30.000000
woman	0-19 yrs	18.019643	49.720836
	20-29 yrs	20.237500	48.354406
	30-39 yrs	16.490420	67.017582
	40-49 yrs	24.125420	71.074250
	50-59 yrs	19.606250	73.880206
	60-69 yrs	NaN	60.698950

Wide to Long

1. Create a table where the columns are "who" and the values are "pclass"
2. Answer the question: How does this table differ from the pivot_table above? Specifically, how is "who" different?

In [56]:

```
# Code your table here:
pd.wide_to_long(data, ["who", "pclass"], i="who", j="pclass")


# Run your table here:


# Answer the question here:
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_61052\1831683979.py in <module>
      1 # Code your table here:
----> 2 pd.wide_to_long(data, ["who", "pclass"], i="who", j="pclass")
      3 
      4 
      5 # Run your table here:

~\anaconda3\lib\site-packages\pandas\core\reshape\melt.py in wide_to_long(df, stubnam
es, i, j, sep, suffix)
    521 
    522     if any(col in stubnames for col in df.columns):
--> 523         raise ValueError("stubname can't be identical to a column name")
    524 
    525     if not is_list_like(i):

ValueError: stubname can't be identical to a column name
```

Melt -

1. What does **melt** to the data?

In [77]:
```
# What does melt do?
```

1. Melt to your data. Be sure to store the output in a new variable. What is the new shape of your table?

In [78]:
```
# Create your default melt table here with the following syntax:  new_name = pd.melt(

# Run your table here:

# Check the shape of your new table.
```

1. Create a melt table where the index variables are "embarked", and the values are "fare" and "deck"

In [79]:
```
# Create your melt table here:

# Run your table here:

# Check the shape
```

Optonal Challenges:

1. Clean and Explore the table.
   A. How would you handle any missing data?
   B. Would you keep all of the columns?

C. Would you want to manipulate any data?

In [ ]: