Python

Pandas at one click !

Save for later

CLICK ON COLAB LINK AT THE BELOW OF EVERY PAGE TO SEE CODE

Step 1. Go to https://www.kaggle.com/openfoodfacts/world-food-facts/data

Getting and knowing your data

Step 2. Download the dataset to your computer and unzip it.

```
import pandas as pd
import numpy as np
```

Step 3. Use the tsv file and assign it to a dataframe called food

```
food = pd.read_csv('~/Desktop/en.openfoodfacts.org.products.tsv', sep='\t')
```

```
//anaconda/lib/python2.7/site-packages/IPython/core/interactiveshell.py:2717: DtypeWa
  interactivity=interactivity, compiler=compiler, result=result)
```

Step 4. See the first 5 entries

```
food.head()
```

	code	url	creator	created_t	created_datetime
0	3087	http://world-en.openfoodfacts.org/product/0000...	openfoodfacts-contributors	1474103866	2016-09-17T09:17:46Z
1	4530	http://world-en.openfoodfacts.org/product/0000...	usda-ndb-import	1489069957	2017-03-09T14:32:37Z
2	4559	http://world-en.openfoodfacts.org/product/0000...	usda-ndb-import	1489069957	2017-03-09T14:32:37Z
3	16087	http://world-en.openfoodfacts.org/product/0000...	usda-ndb-import	1489055731	2017-03-09T10:35:31Z
4	16094	http://world-en.openfoodfacts.org/product/0000...	usda-ndb-import	1489055653	2017-03-09T10:34:13Z

5 rows × 163 columns

Step 5. What is the number of observations in the dataset?

```
food.shape #will give you both (observations/rows, columns)
```

```
(356027, 163)
```

```
food.shape[0] #will give you only the observations/rows number
```

```
356027
```

Step 6. What is the number of columns in the dataset?

```
print(food.shape) #will give you both (observations/rows, columns)
print(food.shape[1]) #will give you only the columns number

#OR

food.info() #Columns: 163 entries
```

```
(356027, 163)
163
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 356027 entries, 0 to 356026
Columns: 163 entries, code to water-hardness_100g
dtypes: float64(107), object(56)
memory usage: 442.8+ MB
```

Step 7. Print the name of all the columns.

```
food.columns
```

```
Index([u'code', u'url', u'creator', u'created_t', u'created_datetime',
       u'last_modified_t', u'last_modified_datetime', u'product_name',
       u'generic_name', u'quantity',
       ...
       u'fruits-vegetables-nuts_100g', u'fruits-vegetables-nuts-estimate_100g',
       u'collagen-meat-protein-ratio_100g', u'cocoa_100g', u'chlorophyl_100g',
       u'carbon-footprint_100g', u'nutrition-score-fr_100g',
       u'nutrition-score-uk_100g', u'glycemic-index_100g',
       u'water-hardness_100g'],
      dtype='object', length=163)
```

Step 8. What is the name of 105th column?

```
food.columns[104]
```

```
'-glucose_100g'
```

Step 9. What is the type of the observations of the 105th column?

```
food.dtypes['-glucose_100g']
```

```
dtype('float64')
```

Step 10. How is the dataset indexed?

```
food.index
```

```
RangeIndex(start=0, stop=356027, step=1)
```

Step 11. What is the product name of the 19th observation?

```
food.values[18][7]
```

```
'Lotus Organic Brown Jasmine Rice'
```

This time we are going to pull data directly from the internet. Special thanks to: https://github.com/justmarkham for sharing the dataset and materials.

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

```
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'

chipo = pd.read_csv(url, sep = '\t')
```

Step 4. How many products cost more than $10.00?

```
# clean the item_price column and transform it in a float
prices = [float(value[1 : -1]) for value in chipo.item_price]

# reassign the column with the cleaned prices
chipo.item_price = prices

# delete the duplicates in item_name and quantity
chipo_filtered = chipo.drop_duplicates(['item_name','quantity','choice_description'])

# chipo_filtered

# select only the products with quantity equals to 1
chipo_one_prod = chipo_filtered[chipo_filtered.quantity == 1]
chipo_one_prod

# chipo_one_prod[chipo_one_prod['item_price']>10].item_name.nunique()
# chipo_one_prod[chipo_one_prod['item_price']>10]



chipo.query('price_per_item > 10').item_name.nunique()
```

123	54	1	Chicken Bowl	[Fresh Tomato Salsa, [Guacamole, Cheese, Sour ...	11.25
130	57	1	Barbacoa Burrito	[Roasted Chili Corn Salsa, [Rice, Pinto Beans,...	11.75
134	59	1	Chicken Burrito	[Roasted Chili Corn Salsa (Medium), [Rice, Bla...	10.98
135	60	2	Chicken Salad Bowl	[Tomatillo Green Chili Salsa, [Sour Cream, Che...	22.50
136	61	1	Barbacoa Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.75
138	62	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Fajita Vegetables,...	11.25
140	63	1	Chicken Bowl	[Tomatillo Green Chili Salsa, [Rice, Sour Crea...	11.25
142	64	1	Chicken Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
143	64	1	Veggie Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.25
144	65	1	Barbacoa Burrito	[Tomatillo Red Chili Salsa, [Rice, Sour Cream,...	11.75
147	66	1	Steak Burrito	[Tomatillo-Green Chili Salsa (Medium), [Rice, ...	11.48
...	...	...	...	...	...
4504	1791	1	Carnitas Bowl	[Fresh Tomato Salsa, [Rice, Sour Cream, Guacam...	11.75
4506	1792	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Cheese, Sour Cream...	11.25
4510	1793	1	Barbacoa Bowl	[Guacamole]	11.49
4518	1796	1	Steak Bowl	[Tomatillo Red Chili Salsa, [Rice, Black Beans...	11.75
4521	1798	1	Chicken Burrito	[Roasted Chili Corn Salsa, [Guacamole, Lettuce...	11.25
4523	1798	1	Steak Crispy Tacos	[Tomatillo Green Chili Salsa, [Cheese, Sour Cr...	11.75
4533	1802	1	Chicken Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.25
4540	1805	1	Chicken Bowl	[Tomatillo Green Chili Salsa, [Fajita Vegetabl...	11.25
4553	1810	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Black Beans, Sour ...	11.25

Step 5. What is the price of each item?

print a data frame with only two columns item_name and item_price

```
# delete the duplicates in item_name and quantity
# chipo_filtered = chipo.drop_duplicates(['item_name','quantity'])
chipo[(chipo['item_name'] == 'Chicken Bowl') & (chipo['quantity'] == 1)]

# select only the products with quantity equals to 1
# chipo_one_prod = chipo_filtered[chipo_filtered.quantity == 1]

# select only the item_name and item_price columns
```

```
# price_per_item = chipo_one_prod[['item_name', 'item_price']]

# sort the values from the most to less expensive
# price_per_item.sort_values(by = "item_price", ascending = False).head(20)
```

	order_id	quantity	item_name	choice_description	item_price
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	10.98
13	7	1	Chicken Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
19	10	1	Chicken Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	8.75
26	13	1	Chicken Bowl	[Roasted Chili Corn Salsa (Medium), [Pinto Bea...	8.49
42	20	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Rice, Black Beans,...	11.25
76	34	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	8.75
78	34	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	8.75
99	44	1	Chicken Bowl	[Tomatillo Red Chili Salsa, [Rice, Fajita Vege...	8.75
110	49	1	Chicken Bowl	[Tomatillo Red Chili Salsa, [Rice, Black Beans...	8.75
123	54	1	Chicken Bowl	[Fresh Tomato Salsa, [Guacamole, Cheese, Sour ...	11.25
138	62	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Fajita Vegetables,...	11.25
140	63	1	Chicken Bowl	[Tomatillo Green Chili Salsa, [Rice, Sour Crea...	11.25
142	64	1	Chicken Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
160	73	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Black Beans, Rice...	8.49
176	79	1	Chicken Bowl	[Roasted Chili Corn Salsa (Medium), [Black Bea...	10.98
182	82	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Sour ...	8.75
193	86	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Cheese, Sour Cream]]	8.75
199	89	1	Chicken Bowl	[[Roasted Chili Corn Salsa (Medium), Tomatillo...	10.98
206	92	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Cheese, Lettuce]]	8.75
209	93	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	11.25

Step 6. Sort by the name of the item

```
chipo.item_name.sort_values()

# OR

chipo.sort_values(by = "item_name")
```

Ex - GroupBy

Introduction:

GroupBy can be summarized as Split-Apply-Combine.

Special thanks to: https://github.com/justmarkham for sharing the dataset and materials.

Check out this Diagram

Step 1. Import the necessary libraries

```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called drinks.

```
drinks = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/drink
drinks.head()
```

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_a
0	Afghanistan	0	0	0	
1	Albania	89	132	54	
2	Algeria	25	0	14	
3	Andorra	245	138	312	
4	Angola	217	57	45	

Step 4. Which continent drinks more beer on average?

```
drinks.groupby('continent').beer_servings.mean()
```

```
continent
AF     61.471698
AS     37.045455
EU    193.777778
OC     89.687500
SA    175.083333
Name: beer_servings, dtype: float64
```

Step 5. For each continent print the statistics for wine consumption.

```
drinks.groupby('continent').wine_servings.describe()
```

```
continent
AF         count     53.000000
           mean      16.264151
           std       38.846419
           min        0.000000
           25%        1.000000
           50%        2.000000
           75%       13.000000
           max      233.000000
AS         count     44.000000
           mean       9.068182
           std       21.667034
           min        0.000000
           25%        0.000000
           50%        1.000000
           75%        8.000000
           max      123.000000
EU         count     45.000000
           mean     142.222222
           std       97.421738
           min        0.000000
           25%       59.000000
           50%      128.000000
           75%      195.000000
           max      370.000000
OC         count     16.000000
           mean      35.625000
           std       64.555790
           min        0.000000
           25%        1.000000
           50%        8.500000
           75%       23.250000
           max      212.000000
SA         count     12.000000
           mean      62.416667
           std       88.620189
           min        1.000000
           25%        3.000000
           50%       12.000000
           75%       98.500000
           max      221.000000
dtype: float64
```

Step 6. Print the mean alcohol consumption per continent for every column

```
drinks.groupby('continent').mean()
```

continent	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcoh
AF	61.471698	16.339623	16.264151	3.0075
AS	37.045455	60.840909	9.068182	2.1704
EU	193.777778	132.555556	142.222222	8.6177

Step 7. Print the median alcohol consumption per continent for every column

```
drinks.groupby('continent').median()
```

continent	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcoh
AF	32.0	3.0	2.0	2.
AS	17.5	16.0	1.0	1.
EU	219.0	122.0	128.0	10.
OC	52.5	37.0	8.5	1.
SA	162.5	108.5	12.0	6.

Step 8. Print the mean, min and max values for spirit consumption.

This time output a DataFrame

```
drinks.groupby('continent').spirit_servings.agg(['mean', 'min', 'max'])
```

continent	mean	min	max
AF	16.339623	0	152
AS	60.840909	0	326
EU	132.555556	0	373
OC	58.437500	0	254
SA	114.750000	25	302

Student Alcohol Consumption

Introduction:

This time you will download a dataset from the UCI.

Step 1. Import the necessary libraries

```
import pandas as pd
import numpy
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called df.

```
csv_url = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/04_Apply/S
df = pd.read_csv(csv_url)
df.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	fa
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	
3	GP	F	15	U	GT3	T	4	2	health	services	...	
4	GP	F	16	U	GT3	T	3	3	other	other	...	

5 rows × 33 columns

Step 4. For the purpose of this exercise slice the dataframe from 'school' until the 'guardian' column

```
stud_alcoh = df.loc[: , "school":"guardian"]
stud_alcoh.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	course
1	GP	F	17	U	GT3	T	1	1	at_home	other	course

Step 5. Create a lambda function that will capitalize strings.

```
capitalizer = lambda x: x.capitalize()
```

Step 6. Capitalize both Mjob and Fjob

```
stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'].apply(capitalizer)
```

```
0       Teacher
1         Other
2         Other
3      Services
4         Other
5         Other
6         Other
7       Teacher
8         Other
9         Other
10       Health
11        Other
12     Services
13        Other
14        Other
15        Other
16     Services
17        Other
18     Services
19        Other
20        Other
21       Health
22        Other
23        Other
24       Health
25     Services
26        Other
27     Services
28        Other
29      Teacher
         ...
365       Other
366    Services
367    Services
368    Services
369     Teacher
370    Services
371    Services
372     At_home
373       Other
```

```
374       Other
375       Other
376       Other
377    Services
378       Other
379       Other
380     Teacher
381       Other
382    Services
383    Services
384       Other
385       Other
386     At_home
387       Other
388    Services
389       Other
390    Services
391    Services
392       Other
```

Step 7. Print the last elements of the data set.

```
stud_alcoh.tail()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reaso
390	MS	M	20	U	LE3	A	2	2	services	services	cours
391	MS	M	17	U	LE3	T	3	1	services	services	cours
392	MS	M	21	R	GT3	T	1	1	other	other	cours
393	MS	M	18	R	LE3	T	3	2	services	other	cours
394	MS	M	19	U	LE3	T	1	1	other	at_home	cours

Step 8. Did you notice the original dataframe is still lowercase? Why is that? Fix it and capitalize Mjob and Fjob.

```
stud_alcoh['Mjob'] = stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'] = stud_alcoh['Fjob'].apply(capitalizer)
stud_alcoh.tail()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reasc
390	MS	M	20	U	LE3	A	2	2	Services	Services	cours
391	MS	M	17	U	LE3	T	3	1	Services	Services	cours
392	MS	M	21	R	GT3	T	1	1	Other	Other	cours
393	MS	M	18	R	LE3	T	3	2	Services	Other	cours
394	MS	M	19	U	LE3	T	1	1	Other	At_home	cours

Step 9. Create a function called majority that returns a boolean value to a new column called legal_drinker (Consider majority as older than 17 years old)

```
def majority(x):
    if x > 17:
        return True
    else:
        return False
```

```
stud_alcoh['legal_drinker'] = stud_alcoh['age'].apply(majority)
stud_alcoh.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason
0	GP	F	18	U	GT3	A	4	4	At_home	Teacher	course
1	GP	F	17	U	GT3	T	1	1	At_home	Other	course
2	GP	F	15	U	LE3	T	1	1	At_home	Other	other
3	GP	F	15	U	GT3	T	4	2	Health	Services	home
4	GP	F	16	U	GT3	T	3	3	Other	Other	home

Step 10. Multiply every number of the dataset by 10.

I know this makes no sense, don't forget it is just an exercise

```
def times10(x):
    if type(x) is int:
        return 10 * x
    return x
```

```
stud_alcoh.applymap(times10).head(10)
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reas
0	GP	F	180	U	GT3	A	40	40	At_home	Teacher	cour
1	GP	F	170	U	GT3	T	10	10	At_home	Other	cour
2	GP	F	150	U	LE3	T	10	10	At_home	Other	oth
3	GP	F	150	U	GT3	T	40	20	Health	Services	hon
4	GP	F	160	U	GT3	T	30	30	Other	Other	hon
5	GP	M	160	U	LE3	T	40	30	Services	Other	reputati
6	GP	M	160	U	LE3	T	20	20	Other	Other	hon
7	GP	F	170	U	GT3	A	40	40	Other	Teacher	hon
8	GP	M	150	U	LE3	A	30	20	Services	Other	hon
9	GP	M	150	U	GT3	T	30	40	Other	Other	hon

Housing Market

Introduction:

This time we will create our own dataset with fictional numbers to describe a house market. As we are going to create random data don't try to reason of the numbers.

Step 1. Import the necessary libraries

```
import pandas as pd
import numpy as np
```

Step 2. Create 3 differents Series, each of length 100, as follows:

1. The first a random number from 1 to 4
2. The second a random number from 1 to 3
3. The third a random number from 10,000 to 30,000

```
s1 = pd.Series(np.random.randint(1, high=5, size=100, dtype='l'))
s2 = pd.Series(np.random.randint(1, high=4, size=100, dtype='l'))
s3 = pd.Series(np.random.randint(10000, high=30001, size=100, dtype='l'))

print(s1, s2, s3)
```

```
     0     2
     1     2
     2     4
     3     2
     4     1
     5     1
     6     2
     7     3
     8     3
     9     2
     10    1
     11    2
     12    4
     13    1
     14    2
     15    3
     16    4
     17    4
     18    4
     19    3
     20    2
     21    1
     22    4
     23    1
```

```
24    3
25    2
26    3
27    1
28    3
29    4
     ..
70    4
71    2
72    2
73    4
74    2
75    1
76    2
77    4
78    3
79    2
80    2
81    2
82    4
83    2
84    2
85    2
86    1
87    3
88    1
89    1
90    1
91    3
92    1
93    2
94    3
95    4
96    4
97    2
```

Step 3. Let's create a DataFrame by joinning the Series by column

```
housemkt = pd.concat([s1, s2, s3], axis=1)
housemkt.head()
```

	0	1	2
0	2	2	16957
1	2	3	24571
2	4	2	28303
3	2	3	14153
4	1	3	23445

Step 4. Change the name of the columns to bedrs, bathrs, price_sqr_meter

```
housemkt.rename(columns = {0: 'bedrs', 1: 'bathrs', 2: 'price_sqr_meter'}, inplace=True)
```

```
housemkt.head()
```

	bedrs	bathrs	price_sqr_meter
0	2	2	16957
1	2	3	24571
2	4	2	28303
3	2	3	14153
4	1	3	23445

Step 5. Create a one column DataFrame with the values of the 3 Series and assign it to 'bigcolumn'

```
# join concat the values
bigcolumn = pd.concat([s1, s2, s3], axis=0)

# it is still a Series, so we need to transform it to a DataFrame
bigcolumn = bigcolumn.to_frame()
print(type(bigcolumn))

bigcolumn
```

```
<class 'pandas.core.frame.DataFrame'>
```

	0
0	2
1	2
2	4
3	2
4	1
5	1
6	2
7	3
8	3
9	2
10	1
11	2
12	4
13	1
14	2
15	3
16	4
17	4
18	4
19	3
20	2
21	1
22	4
23	1
24	3
25	2
26	3
27	1
28	3
29	4

Step 6. Oops, it seems it is going only until index 99. Is it true?

```
# no the index are kept but the length of the DataFrame is 300
len(bigcolumn)
```

```
300
```

Step 7. Reindex the DataFrame so it goes from 0 to 299

```
bigcolumn.reset_index(drop=True, inplace=True)
bigcolumn
```

	0
0	2
1	2
2	4
3	2
4	1
5	1
6	2
7	3
8	3
9	2
10	1
11	2
12	4
13	1
14	2
15	3
16	4
17	4
18	4
19	3
20	2
21	1
22	4
23	1
24	3
25	2
26	3
27	1
28	3
29	4
...	...

Introduction:

The data have been modified to contain some missing values, identified by NaN.
Using pandas should make this exercise easier, in particular for the bonus question.

You should be able to perform all of these operations without using a for loop or other looping construct.

1. The data in 'wind.data' has the following format:

```
"""
Yr Mo Dy   RPT   VAL   ROS   KIL   SHA   BIR   DUB   CLA   MUL   CLO   BEL   MAL
61  1  1 15.04 14.96 13.17  9.29   NaN  9.87 13.67 10.25 10.83 12.58 18.50 15.04
61  1  2 14.71   NaN 10.83  6.50 12.62  7.67 11.50 10.04  9.79  9.67 17.54 13.83
61  1  3 18.50 16.88 12.33 10.13 11.17  6.17 11.25   NaN  8.50  7.67 12.75 12.71
"""
```

```
'\nYr Mo Dy   RPT   VAL   ROS   KIL   SHA   BIR   DUB   CLA   MUL   CLO   BEL   MAL\r
```

The first three columns are year, month and day. The remaining 12 columns are average windspeeds in knots at 12 locations in Ireland on that day.

More information about the dataset go here.

Step 1. Import the necessary libraries

```
import pandas as pd
import datetime
```

Step 2. Import the dataset from this address

Step 3. Assign it to a variable called data and replace the first 3 columns by a proper datetime index.

```
# parse_dates gets 0, 1, 2 columns and parses them as the index
data_url = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/06_Stats/
data = pd.read_csv(data_url, sep = "\s+", parse_dates = [[0,1,2]])
data.head()
```

	Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	C
0	2061-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.
1	2061-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.

Step 4. Year 2061? Do we really have data from this year? Create a function to fix it and apply it.

```
# The problem is that the dates are 2061 and so on...

# function that uses datetime
def fix_century(x):
  year = x.year - 100 if x.year > 1989 else x.year
  return datetime.date(year, x.month, x.day)

# apply the function fix_century on the column and replace the values to the right ones
data['Yr_Mo_Dy'] = data['Yr_Mo_Dy'].apply(fix_century)

# data.info()
data.head()
```

	Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	C
0	1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.
1	1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.
2	1961-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.

Step 5. Set the right dates as the index. Pay attention at the data type, it should be datetime64[ns].

```
# transform Yr_Mo_Dy it to date type datetime64
data["Yr_Mo_Dy"] = pd.to_datetime(data["Yr_Mo_Dy"])

# set 'Yr_Mo_Dy' as the index
data = data.set_index('Yr_Mo_Dy')

data.head()
# data.info()
```

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO
1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58
1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67

Step 6. Compute how many values are missing for each location over the entire record.

They should be ignored in all calculations below.

```
# "Number of non-missing values for each location: "
data.isnull().sum()
```

```
RPT    6
VAL    3
ROS    2
KIL    5
SHA    2
BIR    0
DUB    3
CLA    2
MUL    3
CLO    1
BEL    0
MAL    4
dtype: int64
```

Step 7. Compute how many non-missing values there are in total.

```
#number of columns minus the number of missing values for each location
data.shape[0] - data.isnull().sum()

#or

data.notnull().sum()
```

```
RPT    6568
VAL    6571
ROS    6572
KIL    6569
SHA    6572
BIR    6574
DUB    6571
CLA    6572
MUL    6571
CLO    6573
BEL    6574
MAL    6570
dtype: int64
```

Step 8. Calculate the mean windspeeds of the windspeeds over all the locations and all the times.

A single number for the entire dataset.

```
data.sum().sum() / data.notna().sum().sum()
```

```
10.227883764282167
```

Step 9. Create a DataFrame called loc_stats and calculate the min, max and mean windspeeds and standard deviations of the windspeeds at each location over all the days

A different set of numbers for each location.

```
data.describe(percentiles=[])
```

	RPT	VAL	ROS	KIL	SHA	
count	6568.000000	6571.000000	6572.000000	6569.000000	6572.000000	657
mean	12.362987	10.644314	11.660526	6.306468	10.455834	
std	5.618413	5.267356	5.008450	3.605811	4.936125	
min	0.670000	0.210000	1.500000	0.000000	0.130000	
50%	11.710000	10.170000	10.920000	5.750000	9.960000	
max	35.800000	33.370000	33.840000	28.460000	37.540000	

Step 10. Create a DataFrame called day_stats and calculate the min, max and mean windspeed and standard deviations of the windspeeds across all the locations at each day.

A different set of numbers for each day.

```
# create the dataframe
day_stats = pd.DataFrame()

# this time we determine axis equals to one so it gets each row.
day_stats['min'] = data.min(axis = 1) # min
day_stats['max'] = data.max(axis = 1) # max
day_stats['mean'] = data.mean(axis = 1) # mean
day_stats['std'] = data.std(axis = 1) # standard deviations
```

```
day_stats.head()
```

Yr_Mo_Dy	min	max	mean	std
1961-01-01	9.29	18.50	13.018182	2.808875
1961-01-02	6.50	17.54	11.336364	3.188994
1961-01-03	6.17	18.50	11.641818	3.681912
1961-01-04	1.79	11.75	6.619167	3.198126
1961-01-05	6.17	13.33	10.630000	2.445356

Step 11. Find the average windspeed in January for each location.

Treat January 1961 and January 1962 both as January.

```
data.loc[data.index.month == 1].mean()
```

```
RPT    14.847325
VAL    12.914560
ROS    13.299624
KIL     7.199498
SHA    11.667734
BIR     8.054839
DUB    11.819355
CLA     9.512047
MUL     9.543208
CLO    10.053566
BEL    14.550520
MAL    18.028763
dtype: float64
```

Step 12. Downsample the record to a yearly frequency for each location.

```
data.groupby(data.index.to_period('A')).mean()
```

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	
1961	12.299583	10.351796	11.362369	6.958227	10.881763	7.729726	9
1962	12.246923	10.110438	11.732712	6.960440	10.657918	7.393068	11
1963	12.813452	10.836986	12.541151	7.330055	11.724110	8.434712	11
1964	12.363661	10.920164	12.104372	6.787787	11.454481	7.570874	10
1965	12.451370	11.075534	11.848767	6.858466	11.024795	7.478110	10
1966	13.461973	11.557205	12.020630	7.345726	11.805041	7.793671	10
1967	12.737151	10.990986	11.739397	7.143425	11.630740	7.368164	10
1968	11.835628	10.468197	11.409754	6.477678	10.760765	6.067322	8
1969	11.166356	9.723699	10.902000	5.767973	9.873918	6.189973	8
1972	12.463962	10.561311	12.058333	5.929699	9.430410	6.358825	9

Step 13. Downsample the record to a monthly frequency for each location.

```
data.groupby(data.index.to_period('M')).mean()
```

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	
1961-01	14.841333	11.988333	13.431613	7.736774	11.072759	8.588065	1
1961-02	16.269286	14.975357	14.441481	9.230741	13.852143	10.937500	1
1961-03	10.890000	11.296452	10.752903	7.284000	10.509355	8.866774	
1961-04	10.722667	9.427667	9.998000	5.830667	8.435000	6.495000	
1961-05	9.860968	8.850000	10.818065	5.905333	9.490323	6.574839	
1961-06	9.904138	8.520333	8.867000	6.083000	10.824000	6.707333	
1961-07	10.614194	8.221613	9.110323	6.340968	10.532581	6.198387	
1961-08	12.035000	10.133871	10.335806	6.845806	12.715161	8.441935	1
1961-09	12.531000	9.656897	10.776897	7.155517	11.003333	7.234000	
1961-10	14.289667	10.915806	12.236452	8.154839	11.865484	8.333871	1
1961-11	10.896333	8.592667	11.850333	6.045667	9.123667	6.250667	1
1961-12	14.973548	11.903871	13.980323	7.073871	11.323548	8.302258	1
1962-01	14.783871	13.160323	12.591935	7.538065	11.779677	8.720000	1
1962-02	15.844643	12.041429	15.178929	9.262963	13.821429	9.726786	1
1962-03	11.634333	8.602258	12.110645	6.403226	10.352258	6.732258	1
1962-04	12.160667	9.676667	12.088333	7.163000	10.544000	7.558000	1
1962-05	12.745806	10.865484	11.874839	7.471935	11.285806	7.209032	1
1962-06	10.305667	9.677000	9.996333	6.846667	10.711333	7.441333	1
1962-07	9.981935	8.370645	9.753548	6.093226	9.112903	5.877097	
1962-08	10.964194	9.694194	10.184516	6.701290	10.465161	7.009032	1
1962-09	11.176333	9.507000	11.640000	6.164333	9.722333	6.214000	
1962-10	9.699355	8.063548	9.357097	4.818065	8.432258	5.730000	
1962-11	11.071333	7.984000	12.035667	5.740000	8.135667	6.338333	
1962-12	16.785484	13.753548	14.276452	9.557419	13.724839	10.321613	1
1963-01	14.868387	11.112903	15.121613	6.635806	11.080645	7.835484	1
1963-02	14.418929	11.876429	15.697500	8.611786	12.887857	9.600357	1
1963-03	14.853871	12.271290	14.295806	9.268387	13.112903	10.088065	1

Step 14. Downsample the record to a weekly frequency for each location.

```
data.groupby(data.index.to_period('W')).mean()
```

1961-07-17/1961-07-23	4.202857	4.255714	6.738571	3.300000	6.112857	2.715714	3.964286
...	...	...	...	...	...	...	...
1978-06-05/1978-06-11	12.022857	9.154286	9.488571	5.971429	10.637143	8.030000	8.678571
1978-06-12/1978-06-18	9.410000	8.770000	14.135714	6.457143	8.564286	6.898571	7.297143
1978-06-19/1978-06-25	12.707143	10.244286	8.912857	5.878571	10.372857	6.852857	7.648571
1978-06-26/1978-07-02	12.208571	9.640000	10.482857	7.011429	12.772857	9.005714	11.055714
1978-07-03/1978-07-09	18.052857	12.630000	11.984286	9.220000	13.414286	10.762857	11.368571
1978-07-10/1978-07-16	5.882857	3.244286	5.358571	2.250000	4.618571	2.631429	2.494286
1978-07-17/1978-07-23	13.654286	10.007143	9.915714	6.577143	10.757143	8.282857	8.147143
1978-07-24/1978-07-30	12.172857	11.854286	11.094286	6.631429	9.918571	8.707143	7.458571
1978-07-31/1978-08-06	12.475714	9.488571	10.584286	5.457143	8.724286	5.855714	7.065714
1978-08-07/1978-08-13	10.114286	9.600000	7.635714	4.790000	8.101429	6.702857	5.452857
1978-08-14/1978-08-20	11.100000	11.237143	10.505714	5.697143	9.910000	8.034286	7.267143
1978-08-21/1978-08-27	6.208571	5.060000	8.565714	3.121429	4.638571	4.077143	3.291429
1978-08-28/1978-09-03	8.232857	4.888571	7.767143	3.588571	3.892857	5.090000	6.184286
1978-09-04/1978-09-10	11.487143	12.742857	11.124286	5.702857	10.721429	10.927143	9.157143

This time we are going to pull data directly from the internet. Special thanks to: https://github.com/justmarkham for sharing the dataset and materials.

Step 1. Import the necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter

# set this so the
%matplotlib inline
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

```
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv'

chipo = pd.read_csv(url, sep = '\t')
```

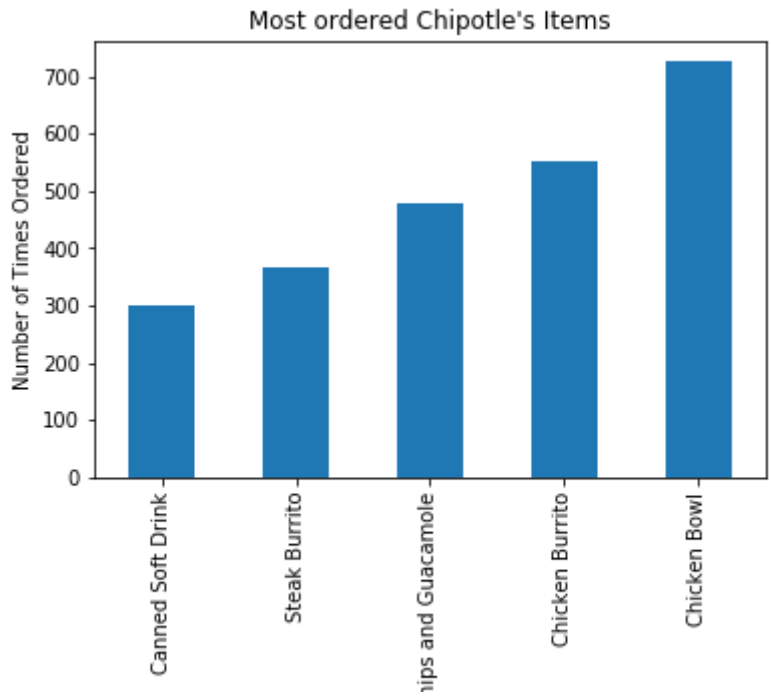
Step 4. See the first 10 entries

```
chipo.head(10)
```

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98

Step 5. Create a histogram of the top 5 items bought

```
# get the Series of the names
x = chipo.item_name

# use the Counter class from collections to create a dictionary with keys(text) and freque
letter_counts = Counter(x)

# convert the dictionary to a DataFrame
df = pd.DataFrame.from_dict(letter_counts, orient='index')

# sort the values from the top to the least value and slice the first 5 items
df = df[0].sort_values(ascending = True)[45:50]

# create the plot
df.plot(kind='bar')

# Set the title and labels
plt.xlabel('Items')
plt.ylabel('Number of Times Ordered')
plt.title('Most ordered Chipotle\'s Items')

# show the plot
plt.show()
```



Step 6. Create a scatterplot with the number of items orderered per order price

Hint: Price should be in the X-axis and Items ordered in the Y-axis

```
# create a list of prices
```

```
chipo.item_price = [float(value[1:-1]) for value in chipo.item_price] # strip the dollar s

# then groupby the orders and sum
orders = chipo.groupby('order_id').sum()

# creates the scatterplot
# plt.scatter(orders.quantity, orders.item_price, s = 50, c = 'green')
plt.scatter(x = orders.item_price, y = orders.quantity, s = 50, c = 'green')

# Set the title and labels
plt.xlabel('Order Price')
plt.ylabel('Items ordered')
plt.title('Number of items ordered per order price')
plt.ylim(0)
```

```
(0, 36.7178857951459)
```

BONUS: Create a question and a graph to answer your own question.

Visualizing the Titanic Disaster

Introduction:

This exercise is based on the titanic Disaster dataset avaiable at Kaggle.
To know more about the variables check here

Step 1. Import the necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
```

Step 2. Import the dataset from this address

Step 3. Assign it to a variable titanic

```
url = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/Visualization/

titanic = pd.read_csv(url)

titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs	female	38.0	1	0	PC 17599	71.

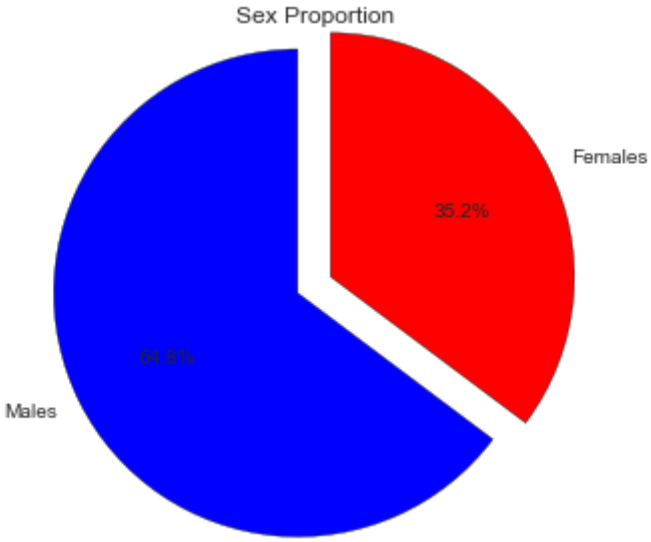
Step 4. Set PassengerId as the index

```
titanic.set_index('PassengerId').head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
			Cumings, Mrs. John Bradley						

Step 5. Create a pie chart presenting the male/female proportion

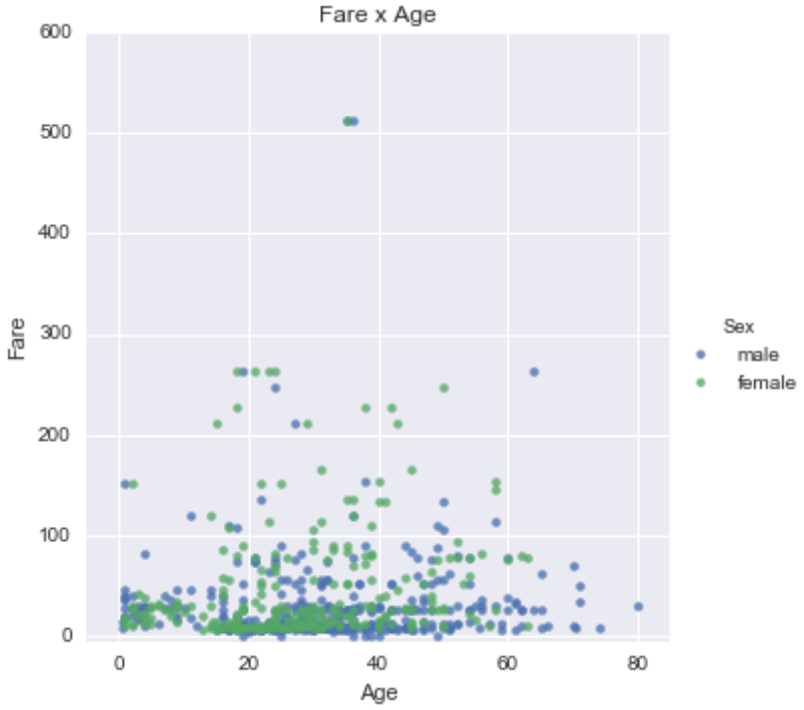
```
# sum the instances of males and females
males = (titanic['Sex'] == 'male').sum()
females = (titanic['Sex'] == 'female').sum()

# put them into a list called proportions
proportions = [males, females]

# Create a pie chart
plt.pie(
    # using proportions
    proportions,

    # with the labels being officer names
    labels = ['Males', 'Females'],

    # with no shadows
    shadow = False,

    # with colors
    colors = ['blue','red'],

    # with one slide exploded out
    explode = (0.15 , 0),

    # with the start angle at 90%
    startangle = 90,

    # with the percent listed as a fraction
    autopct = '%1.1f%%'
    )

# View the plot drop above
plt.axis('equal')

# Set labels
plt.title("Sex Proportion")

# View the plot
plt.tight_layout()
plt.show()
```



Step 6. Create a scatterplot with the Fare payed and the Age, differ the plot color by gender

```
# creates the plot using
lm = sns.lmplot(x = 'Age', y = 'Fare', data = titanic, hue = 'Sex', fit_reg=False)

# set title
lm.set(title = 'Fare x Age')

# get the axes object and tweak it
axes = lm.axes
axes[0,0].set_ylim(-5,)
axes[0,0].set_xlim(-5,85)
```

```
(-5, 85)
```



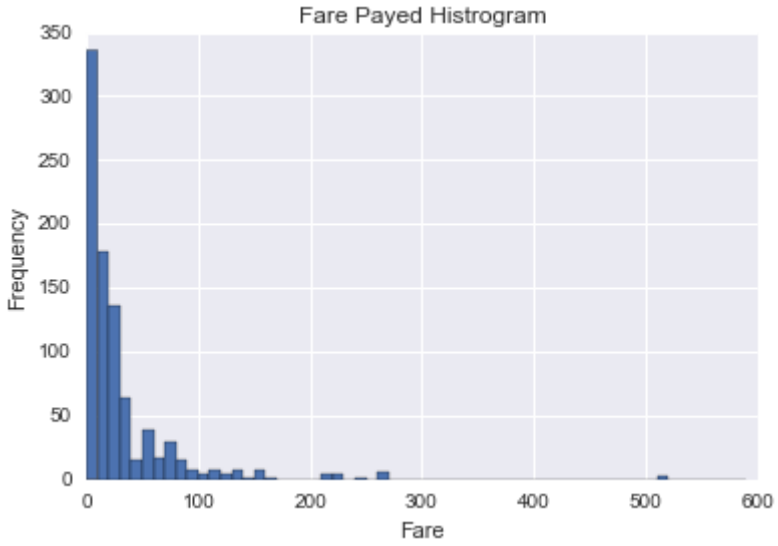
Step 7. How many people survived?

```
titanic.Survived.sum()
```

```
342
```

Step 8. Create a histogram with the Fare payed

```
# sort the values from the top to the least value and slice the first 5 items
df = titanic.Fare.sort_values(ascending = False)
df

# create bins interval using numpy
binsVal = np.arange(0,600,10)
binsVal

# create the plot
plt.hist(df, bins = binsVal)

# Set the title and labels
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.title('Fare Payed Histrogram')

# show the plot
plt.show()
```



BONUS: Create your own question and answer it.