Basic Exercises on Data Importing - Understanding - Manipulating - Analysis - Visualization

Section-1: The pupose of the below exercises (1-7) is to create dictionary and convert into dataframes, how to diplay etc...

The below exercises required to create data

1. Import the necessary libraries (pandas, numpy, datetime, re etc)

In [3]:

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

# set the graphs to show in the jupyter notebook
%matplotlib inline

# set seabor graphs to a better style
sns.set(style="ticks")
```

2. Run the below line of code to create a dictionary and this will be used for below exercises

In [2]:
```
raw_data = {"name": ['Bulbasaur', 'Charmander','Squirtle','Caterpie'],
            "evolution": ['Ivysaur','Charmeleon','Wartortle','Metapod'],
            "type": ['grass', 'fire', 'water', 'bug'],
            "hp": [45, 39, 44, 45],
            "pokedex": ['yes', 'no','yes','no']
            }
```

3. Assign it to a object called pokemon and it should be a pandas DataFrame

In [3]:
```
pokemon=pd.DataFrame(raw_data)
```

4. If the DataFrame columns are in alphabetical order, change the order of the columns as name, type, hp, evolution, pokedex

In [10]:
```
pokemon=pokemon[['name','type','hp','evolution','pokedex']]
pokemon
```

Out[10]:

	name	type	hp	evolution	pokedex
0	Bulbasaur	grass	45	Ivysaur	yes
1	Charmander	fire	39	Charmeleon	no
2	Squirtle	water	44	Wartortle	yes
3	Caterpie	bug	45	Metapod	no

In [17]:
```
pokemon.shape[1]
```

Out[17]: 5

5. Add another column called place, and insert places (lakes, parks, hills, forest etc) of your choice.

In [22]:
```
pokemon['place']='lakes','parks','hills','forest'
pokemon
```

Out[22]:

	name	type	hp	evolution	pokedex	place
0	Bulbasaur	grass	45	Ivysaur	yes	lakes
1	Charmander	fire	39	Charmeleon	no	parks
2	Squirtle	water	44	Wartortle	yes	hills
3	Caterpie	bug	45	Metapod	no	forest

6. Display the data type of each column

In [25]:
```
pokemon.dtypes
```

Out[25]:
```
name          object
type          object
hp             int64
evolution     object
pokedex       object
place         object
dtype: object
```

7. Display the info of dataframe

In [26]:
```
pokemon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   name       4 non-null      object
 1   type       4 non-null      object
 2   hp         4 non-null      int64 
 3   evolution  4 non-null      object
 4   pokedex    4 non-null      object
 5   place      4 non-null      object
dtypes: int64(1), object(5)
memory usage: 320.0+ bytes
```

Section-2: The pupose of the below exercise (8-20) is to understand deleting data with pandas.

The below exercises required to use wine.data

8. Import the dataset *wine.txt* from the folder and assign it to a object called wine

Please note that the original data text file doesn't contain any header. Please ensure that when you import the data, you should use a suitable argument so as to avoid data getting imported as header.

In [58]:
```python
wine=pd.read_table('wine.txt',',',header=None)
wine
```

Out[58]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

9. Delete the first, fourth, seventh, nineth, eleventh, thirteenth and fourteenth columns

In [59]:
```
#wine.drop(columns='0')
wine.drop(columns=wine.iloc[:,[0,3,6,8,10,12,13]],inplace=True)
wine
```

Out[59]:

	1	2	4	5	7	9	11
0	14.23	1.71	15.6	127	3.06	2.29	1.04
1	13.20	1.78	11.2	100	2.76	1.28	1.05
2	13.16	2.36	18.6	101	3.24	2.81	1.03
3	14.37	1.95	16.8	113	3.49	2.18	0.86
4	13.24	2.59	21.0	118	2.69	1.82	1.04
...	...	...	...	...	...	...	...
173	13.71	5.65	20.5	95	0.61	1.06	0.64
174	13.40	3.91	23.0	102	0.75	1.41	0.70
175	13.27	4.28	20.0	120	0.69	1.35	0.59
176	13.17	2.59	20.0	120	0.68	1.46	0.60
177	14.13	4.10	24.5	96	0.76	1.35	0.61

178 rows × 7 columns

10. Assign the columns as below:

The attributes are (dontated by Riccardo Leardi, riclea '@' anchem.unige.it):

1) alcohol
2) malic_acid
3) alcalinity_of_ash
4) magnesium

5) flavanoids
6) proanthocyanins
7) hue

In [60]:
```python
wine.columns=['alcohol','malic_acid','alcalinity_of_ash','magnesium','flavanoids','flavanoids','hue']
wine
```

Out[60]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	flavanoids	hue
0	14.23	1.71	15.6	127	3.06	2.29	1.04
1	13.20	1.78	11.2	100	2.76	1.28	1.05
2	13.16	2.36	18.6	101	3.24	2.81	1.03
3	14.37	1.95	16.8	113	3.49	2.18	0.86
4	13.24	2.59	21.0	118	2.69	1.82	1.04
...	...	...	...	...	...	...	...
173	13.71	5.65	20.5	95	0.61	1.06	0.64
174	13.40	3.91	23.0	102	0.75	1.41	0.70
175	13.27	4.28	20.0	120	0.69	1.35	0.59
176	13.17	2.59	20.0	120	0.68	1.46	0.60
177	14.13	4.10	24.5	96	0.76	1.35	0.61

178 rows × 7 columns

11. Set the values of the first 3 values from alcohol column as NaN

In [61]:
```
wine.iloc[[0,1,2],[0]]=float("NaN")
wine
```

Out[61]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	flavanoids	hue
0	NaN	1.71	15.6	127	3.06	2.29	1.04
1	NaN	1.78	11.2	100	2.76	1.28	1.05
2	NaN	2.36	18.6	101	3.24	2.81	1.03
3	14.37	1.95	16.8	113	3.49	2.18	0.86
4	13.24	2.59	21.0	118	2.69	1.82	1.04
...	...	...	...	...	...	...	...
173	13.71	5.65	20.5	95	0.61	1.06	0.64
174	13.40	3.91	23.0	102	0.75	1.41	0.70
175	13.27	4.28	20.0	120	0.69	1.35	0.59
176	13.17	2.59	20.0	120	0.68	1.46	0.60
177	14.13	4.10	24.5	96	0.76	1.35	0.61

178 rows × 7 columns

12. Now set the value of the rows 3 and 4 of magnesium as NaN

In [62]:
```
wine.iloc[[2,3],[3]]=float("NaN")
wine
```

Out[62]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	flavanoids	hue
0	NaN	1.71	15.6	127.0	3.06	2.29	1.04
1	NaN	1.78	11.2	100.0	2.76	1.28	1.05
2	NaN	2.36	18.6	NaN	3.24	2.81	1.03
3	14.37	1.95	16.8	NaN	3.49	2.18	0.86
4	13.24	2.59	21.0	118.0	2.69	1.82	1.04
...	...	...	...	...	...	...	...
173	13.71	5.65	20.5	95.0	0.61	1.06	0.64
174	13.40	3.91	23.0	102.0	0.75	1.41	0.70
175	13.27	4.28	20.0	120.0	0.69	1.35	0.59
176	13.17	2.59	20.0	120.0	0.68	1.46	0.60
177	14.13	4.10	24.5	96.0	0.76	1.35	0.61

178 rows × 7 columns

13. Fill the value of NaN with the number 10 in alcohol and 100 in magnesium

In [63]:
```
wine.alcohol=wine.alcohol.fillna(10)
```

In [65]:
```
wine.magnesium=wine.magnesium.fillna(100)
```

14. Count the number of missing values in all columns.

In [66]:
```python
wine.shape[0]-wine.count()
```

Out[66]:
```
alcohol              0
malic_acid           0
alcalinity_of_ash    0
magnesium            0
flavanoids           0
flavanoids           0
hue                  0
dtype: int64
```

15. Create an array of 10 random numbers up until 10 and save it.

In [72]:
```python
array1=np.random.randint(0,10,10)
array1
```

Out[72]:
```
array([3, 3, 6, 4, 8, 6, 5, 1, 1, 1])
```

In [102]:
```python
list(array1)
```

Out[102]:
```
[3, 3, 6, 4, 8, 6, 5, 1, 1, 1]
```

16. Set the rows corresponding to the random numbers to NaN in the column *alcohol*

In [109]:
```python
wine.iloc[list(array1),0]=float("NaN")
```

17. How many missing values do we have now?

In [110]:
```
wine.shape[0]-wine.count()
```

Out[110]:
```
alcohol              6
malic_acid           0
alcalinity_of_ash    0
magnesium            0
flavanoids           0
flavanoids           0
hue                  0
dtype: int64
```

18. Print only the non-null values in alcohol

In [111]:
```
wine[wine.alcohol.notna()]
```

Out[111]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	flavanoids	hue
0	10.00	1.71	15.6	127.0	3.06	2.29	1.04
2	10.00	2.36	18.6	100.0	3.24	2.81	1.03
7	14.06	2.15	17.6	121.0	2.51	1.25	1.06
9	13.86	1.35	16.0	98.0	3.15	1.85	1.01
10	14.10	2.16	18.0	105.0	3.32	2.38	1.25
...	...	...	...	...	...	...	...
173	13.71	5.65	20.5	95.0	0.61	1.06	0.64
174	13.40	3.91	23.0	102.0	0.75	1.41	0.70
175	13.27	4.28	20.0	120.0	0.69	1.35	0.59
176	13.17	2.59	20.0	120.0	0.68	1.46	0.60
177	14.13	4.10	24.5	96.0	0.76	1.35	0.61

172 rows × 7 columns

19. Delete the rows that contain missing values

In [112]:
```
wine.dropna(axis=0, how='any')
```

Out[112]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	flavanoids	hue
0	10.00	1.71	15.6	127.0	3.06	2.29	1.04
2	10.00	2.36	18.6	100.0	3.24	2.81	1.03
7	14.06	2.15	17.6	121.0	2.51	1.25	1.06
9	13.86	1.35	16.0	98.0	3.15	1.85	1.01
10	14.10	2.16	18.0	105.0	3.32	2.38	1.25
...	...	...	...	...	...	...	...
173	13.71	5.65	20.5	95.0	0.61	1.06	0.64
174	13.40	3.91	23.0	102.0	0.75	1.41	0.70
175	13.27	4.28	20.0	120.0	0.69	1.35	0.59
176	13.17	2.59	20.0	120.0	0.68	1.46	0.60
177	14.13	4.10	24.5	96.0	0.76	1.35	0.61

172 rows × 7 columns

20. Reset the index, so it starts with 0 again

In [113]:
```
wine.reset_index()
```

Out[113]:

	index	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	flavanoids	hue
0	0	10.00	1.71	15.6	127.0	3.06	2.29	1.04
1	1	NaN	1.78	11.2	100.0	2.76	1.28	1.05
2	2	10.00	2.36	18.6	100.0	3.24	2.81	1.03
3	3	NaN	1.95	16.8	100.0	3.49	2.18	0.86
4	4	NaN	2.59	21.0	118.0	2.69	1.82	1.04
...	...	...	...	...	...	...	...	...
173	173	13.71	5.65	20.5	95.0	0.61	1.06	0.64
174	174	13.40	3.91	23.0	102.0	0.75	1.41	0.70
175	175	13.27	4.28	20.0	120.0	0.69	1.35	0.59
176	176	13.17	2.59	20.0	120.0	0.68	1.46	0.60
177	177	14.13	4.10	24.5	96.0	0.76	1.35	0.61

178 rows × 8 columns

Section-3: The pupose of the below exercise (21-27) is to understand *filtering & sorting* data from dataframe.

The below exercises required to use chipotle.tsv

This time we are going to pull data directly from the internet.
Import the dataset directly from this link (https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv (https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv)) and create dataframe called chipo

In [234]:
```
chipo=pd.read_csv("https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv",sep='\t', header=0)
chipo.head()
```

Out[234]:

	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

21. How many products cost more than $10.00?

Use `str` attribute to remove the $ sign and convert the column to proper numeric type data before filtering.

In [ ]:
```
chipo['item_price'] = chipo['item_price'].str.replace('$', '')
```

In [236]:
```
chipo['item_price']=pd.to_numeric(chipo['item_price'])
```

In [237]:
```
chipo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   order_id            4622 non-null   int64  
 1   quantity            4622 non-null   int64  
 2   item_name           4622 non-null   object 
 3   choice_description  3376 non-null   object 
 4   item_price          4622 non-null   float64
dtypes: float64(1), int64(2), object(2)
memory usage: 180.7+ KB
```

In [238]:
```
chipo[chipo.item_price > 10].count()
```

Out[238]:
```
order_id              1130
quantity              1130
item_name             1130
choice_description    1123
item_price            1130
dtype: int64
```

In [2]:
```
def remove_dollar(a):
    out=''
    for i in range(1,len(a)):
        out=out+a[i]
    return float(out)
```

In [3]:
```
remove_dollar('$45.23')
```

Out[3]:
```
45.23
```

22. Print the Chipo Dataframe & info about data frame

In [239]:
```
chipo
```

Out[239]:

	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
...	...	...	...	...	...
4617	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Sour ...	11.75
4618	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Sour Cream, Cheese...	11.75
4619	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	11.25
4620	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Lettu...	8.75
4621	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	8.75

4622 rows × 5 columns

In [240]:
```
chipo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   order_id            4622 non-null   int64  
 1   quantity            4622 non-null   int64  
 2   item_name           4622 non-null   object 
 3   choice_description  3376 non-null   object 
 4   item_price          4622 non-null   float64
dtypes: float64(1), int64(2), object(2)
memory usage: 180.7+ KB
```

23. What is the price of each item?

- Delete the duplicates in item_name and quantity
- Print a data frame with only two columns `item_name` and `item_price`
- Sort the values from the most to less expensive

In [241]:
```
chipo.head()
```

Out[241]:

	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

In [242]:
```
chipo2=chipo.drop_duplicates(subset=['item_name','quantity'])
```

In [243]:
```
chipo2.head()
```

Out[243]:

	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

In [244]:
```
chipo2[['item_name','item_price']]
```

Out[244]:

	item_name	item_price
0	Chips and Fresh Tomato Salsa	2.39
1	Izze	3.39
2	Nantucket Nectar	3.39
3	Chips and Tomatillo-Green Chili Salsa	2.39
4	Chicken Bowl	16.98
...	...	...
3890	Carnitas Crispy Tacos	17.98
3973	Canned Soft Drink	5.00
4152	Bottled Water	15.00
4354	Steak Soft Tacos	18.50
4489	Chips and Guacamole	17.80

103 rows × 2 columns

In [245]:
```
chipo2.sort_values(by="item_price", ascending=False)
```

Out[245]:

	order_id	quantity	item_name	choice_description	item_price
3598	1443	15	Chips and Fresh Tomato Salsa	NaN	44.25
3480	1398	3	Carnitas Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	35.25
1254	511	4	Chicken Burrito	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	35.00
3601	1443	3	Veggie Burrito	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	33.75
409	178	3	Chicken Bowl	[[Fresh Tomato Salsa (Mild), Tomatillo-Green C...	32.94
...	...	...	...	...	...
40	19	1	Chips	NaN	2.15
6	3	1	Side of Chips	NaN	1.69
263	114	1	Canned Soft Drink	[Coke]	1.25
34	17	1	Bottled Water	NaN	1.09
28	14	1	Canned Soda	[Dr. Pepper]	1.09

103 rows × 5 columns

24. Sort by the name of the item

In [246]:
```
chipo2.sort_values(by="item_name")
```

Out[246]:

	order_id	quantity	item_name	choice_description	item_price
298	129	1	6 Pack Soft Drink	[Sprite]	6.49
3389	1360	2	6 Pack Soft Drink	[Diet Coke]	12.98
39	19	1	Barbacoa Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	11.75
21	11	1	Barbacoa Burrito	[[Fresh Tomato Salsa (Mild), Tomatillo-Green C...	8.99
1903	768	2	Barbacoa Crispy Tacos	[Fresh Tomato Salsa, [Sour Cream, Cheese, Rice]]	18.50
...	...	...	...	...	...
1653	668	1	Veggie Crispy Tacos	[Fresh Tomato Salsa (Mild), [Pinto Beans, Rice...	8.49
1694	686	1	Veggie Salad	[[Fresh Tomato Salsa (Mild), Roasted Chili Cor...	8.49
186	83	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
3889	1559	2	Veggie Soft Tacos	[Fresh Tomato Salsa (Mild), [Black Beans, Rice...	16.98
738	304	1	Veggie Soft Tacos	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.25

103 rows × 5 columns

25. What was the quantity of the most expensive item ordered?

In [247]:
```
chipo2.item_price.max()
```

Out[247]: 44.25

In [248]:
```
chipo2[chipo2.item_price==chipo2.item_price.max()]
```

Out[248]:

	order_id	quantity	item_name	choice_description	item_price
3598	1443	15	Chips and Fresh Tomato Salsa	NaN	44.25

26. How many times were a Veggie Salad Bowl ordered?

In [249]:
```
chipo2[chipo2.item_name=='Veggie Salad Bowl']
```

Out[249]:

	order_id	quantity	item_name	choice_description	item_price
186	83	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25

In [255]:
```
chipo[chipo.item_name=='Veggie Salad Bowl'].count()
```

Out[255]:
```
order_id              18
quantity              18
item_name             18
choice_description    18
item_price            18
dtype: int64
```

27. How many times people orderd more than one Canned Soda?

In [259]:
```
chipo[(chipo.item_name=='Canned Soda') & (chipo.quantity>1)].count()
```

Out[259]:
```
order_id              20
quantity              20
item_name             20
choice_description    20
item_price            20
dtype: int64
```

Section-4: The purpose of the below exercises is to understand how to perform aggregations of data frame

The below exercises (28-33) required to use occupation.csv

28. Import the dataset occupation.csv and assign object as users

In [4]:
```
users=pd.read_csv('occupation.csv',sep='|')
```

29. Discover what is the mean age per occupation

In [5]:
```
users
```

Out[5]:

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
...	...	...	...	...	...
938	939	26	F	student	33319
939	940	32	M	administrator	02215
940	941	20	M	student	97229
941	942	48	F	librarian	78209
942	943	22	M	student	77841

943 rows × 5 columns

In [6]:
```
users.groupby('occupation').age.mean()
```

Out[6]:
```
occupation
administrator    38.746835
artist           31.392857
doctor           43.571429
educator         42.010526
engineer         36.388060
entertainment    29.222222
executive        38.718750
healthcare       41.562500
homemaker        32.571429
lawyer           36.750000
librarian        40.000000
marketing        37.615385
none             26.555556
other            34.523810
programmer       33.121212
retired          63.071429
salesman         35.666667
scientist        35.548387
student          22.081633
technician       33.148148
writer           36.311111
Name: age, dtype: float64
```

30. Discover the Male ratio per occupation and sort it from the most to the least.

Use numpy.where() to encode gender column.

In [21]:
```
users[users.occupation=='student'].gender.value_counts(normalize=True)
```

Out[21]:
```
M    0.693878
F    0.306122
Name: gender, dtype: float64
```

In [19]:
```python
users[users.gender=='M'].occupation.value_counts(normalize=True)
```

Out[19]:
```
student          0.202985
other            0.102985
educator         0.102985
engineer         0.097015
programmer       0.089552
administrator    0.064179
executive        0.043284
scientist        0.041791
writer           0.038806
technician       0.038806
librarian        0.032836
entertainment    0.023881
marketing        0.023881
artist           0.022388
retired          0.019403
lawyer           0.014925
salesman         0.013433
doctor           0.010448
healthcare       0.007463
none             0.007463
homemaker        0.001493
Name: occupation, dtype: float64
```

In [14]:
```
users.groupby(['occupation','gender'])['user_id'].count().reset_index()
```

Out[14]:

	occupation	gender	user_id
0	administrator	F	36
1	administrator	M	43
2	artist	F	13
3	artist	M	15
4	doctor	M	7
5	educator	F	26
6	educator	M	69
7	engineer	F	2
8	engineer	M	65
9	entertainment	F	2
10	entertainment	M	16
11	executive	F	3
12	executive	M	29
13	healthcare	F	11
14	healthcare	M	5
15	homemaker	F	6
16	homemaker	M	1
17	lawyer	F	2
18	lawyer	M	10
19	librarian	F	29
20	librarian	M	22
21	marketing	F	10
22	marketing	M	16
23	none	F	4

	occupation	gender	user_id
24	none	M	5
25	other	F	36
26	other	M	69
27	programmer	F	6
28	programmer	M	60
29	retired	F	1
30	retired	M	13
31	salesman	F	3
32	salesman	M	9
33	scientist	F	3
34	scientist	M	28
35	student	F	60
36	student	M	136
37	technician	F	1
38	technician	M	26
39	writer	F	19
40	writer	M	26

31. For each occupation, calculate the minimum and maximum ages

In [38]:
```python
users.groupby('occupation').age.agg(['min','max'])
```

Out[38]:

occupation	min	max
administrator	21	70
artist	19	48
doctor	28	64
educator	23	63
engineer	22	70
entertainment	15	50
executive	22	69
healthcare	22	62
homemaker	20	50
lawyer	21	53
librarian	23	69
marketing	24	55
none	11	55
other	13	64
programmer	20	63
retired	51	73
salesman	18	66
scientist	23	55
student	7	42
technician	21	55
writer	18	60

32. For each combination of occupation and gender, calculate the mean age

In [39]:
```
pd.crosstab(index = users.occupation, columns = users.gender, values = users.age, aggfunc='mean')
```

Out[39]:

gender	F	M
occupation		
administrator	40.638889	37.162791
artist	30.307692	32.333333
doctor	NaN	43.571429
educator	39.115385	43.101449
engineer	29.500000	36.600000
entertainment	31.000000	29.000000
executive	44.000000	38.172414
healthcare	39.818182	45.400000
homemaker	34.166667	23.000000
lawyer	39.500000	36.200000
librarian	40.000000	40.000000
marketing	37.200000	37.875000
none	36.500000	18.600000
other	35.472222	34.028986
programmer	32.166667	33.216667
retired	70.000000	62.538462
salesman	27.000000	38.555556
scientist	28.333333	36.321429
student	20.750000	22.669118
technician	38.000000	32.961538
writer	37.631579	35.346154

33. For each occupation present the percentage of women and men

In [43]:
```
pd.crosstab(index = users.occupation, columns = users.gender, values = users.user_id, aggfunc='count',normalize=True)
```

Out[43]:

gender	F	M
occupation		
administrator	0.038176	0.045599
artist	0.013786	0.015907
doctor	0.000000	0.007423
educator	0.027572	0.073171
engineer	0.002121	0.068929
entertainment	0.002121	0.016967
executive	0.003181	0.030753
healthcare	0.011665	0.005302
homemaker	0.006363	0.001060
lawyer	0.002121	0.010604
librarian	0.030753	0.023330
marketing	0.010604	0.016967
none	0.004242	0.005302
other	0.038176	0.073171
programmer	0.006363	0.063627
retired	0.001060	0.013786
salesman	0.003181	0.009544
scientist	0.003181	0.029692
student	0.063627	0.144221
technician	0.001060	0.027572
writer	0.020148	0.027572

Section-6: The purpose of the below exercises is to understand how to use lambda-apply-functions

The below exercises (34-41) required to use student-mat.csv and student-por.csv files

34. Import the datasets *student-mat* and *student-por* and append them and assigned object as df

In [46]:
```
student1=pd.read_csv("student-mat.csv")
```

In [47]:
```
student2=pd.read_csv("student-por.csv")
```

In [48]:

```
student1
```

Out[48]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	MS	M	20	U	LE3	A	2	2	services	services	...	5	5	4	4	5	4	11	9
391	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	3	14
392	MS	M	21	R	GT3	T	1	1	other	other	...	5	5	3	3	3	3	3	10
393	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	0	11
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	3	2	3	3	3	5	5	8

395 rows × 33 columns

In [49]:
```
student2
```

Out[49]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	F	19	R	GT3	T	2	3	services	other	...	5	4	2	1	2	5	4	10
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	4	3	4	1	1	1	4	15
646	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	1	5	6	11
647	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	6	10
648	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	4	10

649 rows × 33 columns

In [50]:
```
df=student1.append(student2)
df
```

Out[50]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	F	19	R	GT3	T	2	3	services	other	...	5	4	2	1	2	5	4	10
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	4	3	4	1	1	1	4	15
646	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	1	5	6	11
647	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	6	10
648	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	4	10

1044 rows × 33 columns

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

In [55]:
```
df2 = df.loc[:,'school':'guardian']
```

36. Create a lambda function that captalize strings (example: if we give at_home as input function and should give At_home as output.

In [44]:
```
lambda x:x.capitalize()
```

Out[44]:
```
<function __main__.<lambda>(x)>
```

37. Capitalize both Mjob and Fjob variables using above lamdba function

In [56]:
```
df2.Mjob.apply(lambda x:x.capitalize())
```

Out[56]:
```
0       At_home
1       At_home
2       At_home
3        Health
4         Other
         ...
644    Services
645     Teacher
646       Other
647    Services
648    Services
Name: Mjob, Length: 1044, dtype: object
```

In [57]:
```
df2.Fjob.apply(lambda x:x.capitalize())
```

Out[57]:
```
0       Teacher
1         Other
2         Other
3      Services
4         Other
         ...
644       Other
645    Services
646       Other
647    Services
648       Other
Name: Fjob, Length: 1044, dtype: object
```

38. Print the last elements of the data set. (Last few records)

In [58]:
```
df2.tail()
```

Out[58]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian
644	MS	F	19	R	GT3	T	2	3	Services	Other	course	mother
645	MS	F	18	U	LE3	T	3	1	Teacher	Services	course	mother
646	MS	F	18	U	GT3	T	1	1	Other	Other	course	mother
647	MS	M	17	U	LE3	T	3	1	Services	Services	course	mother
648	MS	M	18	R	LE3	T	3	2	Services	Other	course	mother

39. Did you notice the original dataframe is still lowercase? Why is that? Fix it and captalize Mjob and Fjob.

In [60]:
```
df2.Mjob=df.Mjob.apply(lambda x:x.capitalize())
df2.Fjob=df.Fjob.apply(lambda x:x.capitalize())
```

40. Create a function called majority that return a boolean value to a new column called legal_drinker

In [63]:
```
df2['majority']=np.where(df2.age<18,False,True)
```

41. Multiply every number of the dataset by 10.

In [64]:
```
df.select_dtypes('int64')*10
```

Out[64]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	180	40	40	20	20	0	40	30	40	10	10	30	60	50	60	60
1	170	10	10	10	20	0	50	30	30	10	10	30	40	50	50	60
2	150	10	10	10	20	30	40	30	20	20	30	30	100	70	80	100
3	150	40	20	10	30	0	30	20	20	10	10	50	20	150	140	150
4	160	30	30	10	20	0	40	30	20	10	20	50	40	60	100	100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	190	20	30	10	30	10	50	40	20	10	20	50	40	100	110	100
645	180	30	10	10	20	0	40	30	40	10	10	10	40	150	150	160
646	180	10	10	20	20	0	10	10	10	10	10	50	60	110	120	90
647	170	30	10	20	10	0	20	40	50	30	40	20	60	100	100	100
648	180	30	20	30	10	0	40	40	10	30	40	50	40	100	110	110

1044 rows × 16 columns

Section-6: The purpose of the below exercises is to understand how to perform simple joins

The below exercises (42-48) required to use cars1.csv and cars2.csv files

42. Import the datasets cars1.csv and cars2.csv and assign names as cars1 and cars2

In [93]:
```
cars1=pd.read_csv('cars1.csv')
cars2=pd.read_csv('cars2.csv')
```

43. Print the information to cars1 by applying below functions

hint: Use different functions/methods like type(), head(), tail(), columns(), info(), dtypes(), index(), shape(), count(), size(), ndim(), axes(), describe(), memory_usage(), sort_values(), value_counts() Also create profile report using pandas_profiling.Profile_Report

In [ ]:
```
cars1.head()
```

In [81]:
```
cars1.describe().T
```

Out[81]:

	count	mean	std	min	25%	50%	75%	max
mpg	198.0	19.719697	5.814254	9.0	15.00	19.0	24.375	35.0
cylinders	198.0	5.898990	1.785417	3.0	4.00	6.0	8.000	8.0
displacement	198.0	223.469697	115.181017	68.0	113.25	228.0	318.000	455.0
weight	198.0	3177.888889	934.783733	1613.0	2302.50	3030.0	4080.750	5140.0
acceleration	198.0	15.005556	2.872382	8.0	13.00	15.0	16.800	23.5
model	198.0	72.818182	1.865332	70.0	71.00	73.0	74.000	76.0
origin	198.0	1.439394	0.708085	1.0	1.00	1.0	2.000	3.0
Unnamed: 9	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Unnamed: 10	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Unnamed: 11	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Unnamed: 12	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Unnamed: 13	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [83]:
```
cars1.columns
```

Out[83]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model', 'origin', 'car', 'Unnamed: 9', 'Unnamed: 10',
       'Unnamed: 11', 'Unnamed: 12', 'Unnamed: 13'],
      dtype='object')
```

44. It seems our first dataset has some unnamed blank columns, fix cars1

In [94]:
```
cars1.dropna(axis=1,how='all', inplace=True)
```

45. What is the number of observations in each dataset?

In [95]:
```
cars1.count()
```

Out[95]:
```
mpg             198
cylinders       198
displacement    198
horsepower      198
weight          198
acceleration    198
model           198
origin          198
car             198
dtype: int64
```

In [99]:
```
cars2.count()
```

Out[99]:
```
mpg             200
cylinders       200
displacement    200
horsepower      200
weight          200
acceleration    200
model           200
origin          200
car             200
dtype: int64
```

46. Join cars1 and cars2 into a single DataFrame called cars

In [118]:
```
cars=cars1.append(cars2).reset_index()
cars
```

Out[118]:

	index	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	car
0	0	18.0	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu
1	1	15.0	8	350	165	3693	11.5	70	1	buick skylark 320
2	2	18.0	8	318	150	3436	11.0	70	1	plymouth satellite
3	3	16.0	8	304	150	3433	12.0	70	1	amc rebel sst
4	4	17.0	8	302	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...	...
393	195	27.0	4	140	86	2790	15.6	82	1	ford mustang gl
394	196	44.0	4	97	52	2130	24.6	82	2	vw pickup
395	197	32.0	4	135	84	2295	11.6	82	1	dodge rampage
396	198	28.0	4	120	79	2625	18.6	82	1	ford ranger
397	199	31.0	4	119	82	2720	19.4	82	1	chevy s-10

398 rows × 10 columns

47. There is a column missing, called owners. Create a random number Series from 15,000 to 73,000.

In [121]:
```
owner=np.random.randint(15000, 73000, 398)
```

48. Add the column owners to cars

In [123]:
```
cars['owners']=owner
cars
```

Out[123]:

	index	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	car	owners
0	0	18.0	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu	16014
1	1	15.0	8	350	165	3693	11.5	70	1	buick skylark 320	51570
2	2	18.0	8	318	150	3436	11.0	70	1	plymouth satellite	35549
3	3	16.0	8	304	150	3433	12.0	70	1	amc rebel sst	62116
4	4	17.0	8	302	140	3449	10.5	70	1	ford torino	50091
...	...	...	...	...	...	...	...	...	...	...	...
393	195	27.0	4	140	86	2790	15.6	82	1	ford mustang gl	38832
394	196	44.0	4	97	52	2130	24.6	82	2	vw pickup	21225
395	197	32.0	4	135	84	2295	11.6	82	1	dodge rampage	24419
396	198	28.0	4	120	79	2625	18.6	82	1	ford ranger	16819
397	199	31.0	4	119	82	2720	19.4	82	1	chevy s-10	49782

398 rows × 11 columns

Section-7: The purpose of the below exercises is to understand how to perform date time operations

49. Write a Python script to display the

- a. Current date and time
- b. Current year
- c. Month of year
- d. Week number of the year

- e. Weekday of the week
- f. Day of year
- g. Day of the month
- h. Day of week

In [66]:
```
pd.Timestamp.now()
```

Out[66]:
```
Timestamp('2022-07-06 17:24:51.745935')
```

50. Write a Python program to convert a string to datetime.

Sample String : Jul 1 2014 2:43PM

Expected Output : 2014-07-01 14:43:00

In [67]:
```
def program1():
    dte=input('enter a date string: ')
    return pd.to_datetime(dte)
```

In [37]:
```
program1()
```

```
enter a date stringJul 1 2014 2:43PM
```

Out[37]:
```
Timestamp('2014-07-01 14:43:00')
```

51. Write a Python program to subtract five days from current date.

Current Date : 2015-06-22

5 days before Current Date : 2015-06-17

In [44]:
```
current_date='2015-06-22'
current_date=pd.to_datetime(current_date)+pd.DateOffset(days = -5)
current_date
```

Out[44]:
```
Timestamp('2015-06-17 00:00:00')
```

52. Write a Python program to convert unix timestamp string to readable date.

Sample Unix timestamp string : 1284105682

Expected Output : 2010-09-10 13:31:22

In [51]:
```
unix_time=1284105682
dt.datetime.fromtimestamp(unix_time).strftime("%Y-%m-%d %H:%M:%S")
```

Out[51]:
```
'2010-09-10 13:31:22'
```

53. Convert the below Series to pandas datetime :

DoB = pd.Series(["07Sep59","01Jan55","15Dec47","11Jul42"])

Make sure that the year is 19XX not 20XX

In [73]:
```
DoB = pd.Series(["07Sep59","01Jan55","15Dec47","11Jul42"])
DoB=pd.to_datetime(DoB)
```

In [74]:
```
DoB
```

Out[74]:
```
0   2059-09-07
1   2055-01-01
2   2047-12-15
3   2042-07-11
dtype: datetime64[ns]
```

In [85]:
```
import datetime
def fix_date(x):
    if x.year > 1989:
        year = x.year - 100
    else:
        year = x.year
    return datetime.date(year,x.month,x.day)
```

In [76]:
```
DoB=DoB.apply(fix_date)
```

In [77]:
```
DoB
```

Out[77]:
```
0    1959-09-07
1    1955-01-01
2    1947-12-15
3    1942-07-11
dtype: object
```

54. Write a Python program to get days between two dates.

In [82]:
```
date1=pd.to_datetime("07Sep59")
date2=pd.to_datetime("01Jan55")

date1-date2
```

Out[82]:
```
Timedelta('1710 days 00:00:00')
```

55. Convert the below date to datetime and then change its display format using the .dt module

Date = "15Dec1989"

Result : "Friday, 15 Dec 98"

In [85]:
```
Date = pd.to_datetime("15Dec1989")
Date.strftime('%A, %d %b %y')
```

Out[85]: 'Friday, 15 Dec 89'

The below exercises (56-66) required to use wind.data file

About wind.data:

The data have been modified to contain some missing values, identified by NaN.

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
"""
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.
```

56. Import the dataset wind.data and assign it to a variable called data and replace the first 3 columns by a proper date time index

In [75]:
```
data=pd.read_csv('wind.data')
```

In [76]:
```
data.tail()
```

Out[76]:

	Yr	Mo	Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
6569	78	12	27	17.58	16.96	17.62	8.08	13.21	11.67	14.46	15.59	14.04	14.00	17.21	40.08
6570	78	12	28	13.21	5.46	13.46	5.00	8.12	9.42	14.33	16.25	15.25	18.05	21.79	41.46
6571	78	12	29	14.00	10.29	14.42	8.71	9.71	10.54	19.17	12.46	14.50	16.42	18.88	29.58
6572	78	12	30	18.50	14.04	21.29	9.13	12.75	9.71	18.08	12.87	12.46	12.12	14.67	28.79
6573	78	12	31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

In [77]:
```
data['DATE']=data['Yr'].astype(str)+'-'+data['Mo'].astype(str)+'-'+data['Dy'].astype(str)
```

In [78]:
```
data.drop(data.columns[[0,1,2]], axis=1, inplace=True)
```

In [298]:
```
data.DATE=pd.to_datetime(data.DATE)
```

In [83]:
```
data
```

Out[83]:

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL	DATE
0	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04	2061-01-01
1	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83	2061-01-02
2	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71	2061-01-03
3	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88	2061-01-04
4	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83	2061-01-05
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6569	17.58	16.96	17.62	8.08	13.21	11.67	14.46	15.59	14.04	14.00	17.21	40.08	1978-12-27
6570	13.21	5.46	13.46	5.00	8.12	9.42	14.33	16.25	15.25	18.05	21.79	41.46	1978-12-28
6571	14.00	10.29	14.42	8.71	9.71	10.54	19.17	12.46	14.50	16.42	18.88	29.58	1978-12-29
6572	18.50	14.04	21.29	9.13	12.75	9.71	18.08	12.87	12.46	12.12	14.67	28.79	1978-12-30
6573	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08	1978-12-31

6574 rows × 13 columns

57. Year 2061 is seemingly imporoper. Convert every year which are < 70 to 19XX instead of 20XX.

In [87]:
```
data['DATE']=data['DATE'].apply(fix_date)
```

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

In [ ]:
```

```

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

They should be ignored in all calculations below.

In [88]:
```
data.shape[0]-data.count()
```

Out[88]:
```
RPT     6
VAL     3
ROS     2
KIL     5
SHA     2
BIR     0
DUB     3
CLA     2
MUL     3
CLO     1
BEL     0
MAL     4
DATE    0
dtype: int64
```

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

In [89]:
```
data.count()
```

Out[89]:
```
RPT     6568
VAL     6571
ROS     6572
KIL     6569
SHA     6572
BIR     6574
DUB     6571
CLA     6572
MUL     6571
CLO     6573
BEL     6574
MAL     6570
DATE    6574
dtype: int64
```

61. Calculate the mean windspeeds over all the locations and all the times.

A single number for the entire dataset.

In [90]:
```
data.mean()
```

Out[90]:
```
RPT    12.362987
VAL    10.644314
ROS    11.660526
KIL     6.306468
SHA    10.455834
BIR     7.092254
DUB     9.797343
CLA     8.495053
MUL     8.493590
CLO     8.707332
BEL    13.121007
MAL    15.599079
dtype: float64
```

In [91]:
```
np.mean(data.mean())
```

Out[91]:
```
10.227982360836924
```

62. 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.

In [274]:
```
def fn_describe2(x):
    n_min = x.min()
    n_max = x.max()
    n_mean = x.mean()
    n_std = x.std()
    return pd.Series([n_min,n_max,n_mean,n_std],index=['Min','Max','Mean','STD'])
```

In [283]:
```
data.columns
data1=data[['RPT','VAL','ROS','KIL','SHA','BIR','DUB','CLA','MUL','CLO','BEL','MAL']]
```

In [285]:
```
loc_stats=data1.apply(fn_describe2)
loc_stats
```

Out[285]:

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
Min	0.670000	0.210000	1.500000	0.000000	0.130000	0.000000	0.000000	0.000000	0.000000	0.040000	0.130000	0.670000
Max	35.800000	33.370000	33.840000	28.460000	37.540000	26.160000	30.370000	31.080000	25.880000	28.210000	42.380000	42.540000
Mean	12.362987	10.644314	11.660526	6.306468	10.455834	7.092254	9.797343	8.495053	8.493590	8.707332	13.121007	15.599079
STD%	5.618413	5.267356	5.008450	3.605811	4.936125	3.968683	4.977555	4.499449	4.166872	4.503954	5.835037	6.699794

63. 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.

In [286]:
```
data2=data
```

In [289]:
```
data2=data2.sort_values(by='DATE')
```

In [293]:
```
data2=data2.set_index('DATE').T
```

In [294]:
```
day_stats=data2.apply(fn_describe2)
day_stats
```

Out[294]:

DATE	1961-01-01	1961-01-02	1961-01-03	1961-01-04	1961-01-05	1961-01-06	1961-01-07	1961-01-08	1961-01-09	1961-01-10	...	1978-12-22	1978-12-23	1978-1
Min	9.290000	6.500000	6.170000	1.790000	6.170000	4.420000	4.960000	5.910000	4.750000	6.54000	...	2.460000	9.500000	4.7900
Max	18.500000	17.540000	18.500000	11.750000	13.330000	13.210000	14.290000	16.620000	15.370000	19.50000	...	13.080000	22.210000	31.7100
Mean	13.018182	11.336364	11.641818	6.619167	10.630000	8.240000	10.385000	10.487500	9.897500	10.47750	...	7.000833	15.613333	10.8233
STD%	2.808875	3.188994	3.681912	3.198126	2.445356	2.998063	3.072114	3.547237	2.905954	3.44261	...	3.237337	3.850840	7.1950

rows × 6574 columns

64. Find the average windspeed in January for each location.

Treat January 1961 and January 1962 both as January.

In [295]:
```
data.head()
```

Out[295]:

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL	DATE
0	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04	1961-01-01
1	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83	1961-01-02
2	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71	1961-01-03
3	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88	1961-01-04
4	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83	1961-01-05

In [301]:
```
data.groupby(data.DATE.dt.month).mean().head(1)
```

Out[301]:

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
DATE												
1	14.847325	12.91456	13.299624	7.199498	11.667734	8.054839	11.819355	9.512047	9.543208	10.053566	14.55052	18.028763

65. Calculate the mean windspeed for each month in the dataset.

Treat January 1961 and January 1962 as *different* months.

(hint: first find a way to create an identifier unique for each month.)

In [ ]:
```

```

66. Calculate the min, max and mean windspeeds and standard deviations of the windspeeds across all locations for each week (assume that the first week starts on January 2 1961) for the first 52 weeks.

In [ ]:
```

```

The below exercises (67-70) required to use appl_1980_2014.csv file

67. Import the file appl_1980_2014.csv and assign it to a variable called 'apple'

In [112]:
```
apple=pd.read_csv('appl_1980_2014.csv')
```

68. Check out the type of the columns

In [142]:
```
apple.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8465 entries, 1980-12-12 to 2014-07-08
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date_Col   8465 non-null   datetime64[ns]
 1   Open       8465 non-null   float64
 2   High       8465 non-null   float64
 3   Low        8465 non-null   float64
 4   Close      8465 non-null   float64
 5   Volume     8465 non-null   int64
 6   Adj Close  8465 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 529.1 KB
```

69. Transform the Date column as a datetime type

In [114]:
```
apple['Date']=pd.to_datetime(apple['Date'])
```

70. Set the date as the index

In [119]:
```
apple=apple.set_index(apple.Date)
apple
```
Out[119]:

Date	Date	Open	High	Low	Close	Volume	Adj Close
2014-07-08	2014-07-08	96.27	96.80	93.92	95.35	65130000	95.35
2014-07-07	2014-07-07	94.14	95.99	94.10	95.97	56305400	95.97
2014-07-03	2014-07-03	93.67	94.10	93.20	94.03	22891800	94.03
2014-07-02	2014-07-02	93.87	94.06	93.09	93.48	28420900	93.48
2014-07-01	2014-07-01	93.52	94.07	93.13	93.52	38170200	93.52
...	...	...	...	...	...	...	...
1980-12-18	1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
1980-12-17	1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
1980-12-16	1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
1980-12-15	1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
1980-12-12	1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45

8465 rows × 7 columns

71. Is there any duplicate dates?

In [ ]:
```
No Duplicate dates.
```

In [120]:
```
apple[apple.Date.duplicated()]
```

Out[120]:

	Date	Open	High	Low	Close	Volume	Adj Close
Date							

72. The index is from the most recent date. Sort the data so that the first entry is the oldest date.

In [162]:
```
apple.rename(columns={'Date':'Date_Col','Adj Close':'Adj_Close'}, inplace=True)
```

In [138]:
```
apple=apple.sort_values(by='Date_Col', ascending=True)
apple
```

Out[138]:

	Date_Col	Open	High	Low	Close	Volume	Adj Close
Date							
1980-12-12	1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45
1980-12-15	1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
1980-12-16	1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
1980-12-17	1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
1980-12-18	1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
...	...	...	...	...	...	...	...
2014-07-01	2014-07-01	93.52	94.07	93.13	93.52	38170200	93.52
2014-07-02	2014-07-02	93.87	94.06	93.09	93.48	28420900	93.48
2014-07-03	2014-07-03	93.67	94.10	93.20	94.03	22891800	94.03
2014-07-07	2014-07-07	94.14	95.99	94.10	95.97	56305400	95.97
2014-07-08	2014-07-08	96.27	96.80	93.92	95.35	65130000	95.35

8465 rows × 7 columns

73. Get the last business day of each month

In [151]:

```
pd.crosstab(index=apple.Date_Col.dt.month, columns=apple.Date_Col.dt.year, values=apple.Date_Col, aggfunc=max)
#pd.crosstab(index = stores.Location, columns = stores.StoreType, values = stores.TotalSales, aggfunc=sum)
```

Out[151]:

Date_Col	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	20
Date_Col																					
1	NaT	1981-01-30	1982-01-29	1983-01-31	1984-01-31	1985-01-31	1986-01-31	1987-01-30	1988-01-29	1989-01-31	...	2005-01-31	2006-01-31	2007-01-31	2008-01-31	2009-01-30	2010-01-29	2011-01-31	2012-01-31	2013-01-31	20 01-
2	NaT	1981-02-27	1982-02-26	1983-02-28	1984-02-29	1985-02-28	1986-02-28	1987-02-27	1988-02-29	1989-02-28	...	2005-02-28	2006-02-28	2007-02-28	2008-02-29	2009-02-27	2010-02-26	2011-02-28	2012-02-29	2013-02-28	20 02-
3	NaT	1981-03-31	1982-03-31	1983-03-31	1984-03-30	1985-03-29	1986-03-31	1987-03-31	1988-03-31	1989-03-31	...	2005-03-31	2006-03-31	2007-03-30	2008-03-31	2009-03-31	2010-03-31	2011-03-31	2012-03-30	2013-03-28	20 03-
4	NaT	1981-04-30	1982-04-30	1983-04-29	1984-04-30	1985-04-30	1986-04-30	1987-04-30	1988-04-29	1989-04-28	...	2005-04-29	2006-04-28	2007-04-30	2008-04-30	2009-04-30	2010-04-30	2011-04-29	2012-04-30	2013-04-30	20 04-
5	NaT	1981-05-29	1982-05-28	1983-05-31	1984-05-31	1985-05-31	1986-05-30	1987-05-29	1988-05-31	1989-05-31	...	2005-05-31	2006-05-31	2007-05-31	2008-05-30	2009-05-29	2010-05-28	2011-05-31	2012-05-31	2013-05-31	20 05-
6	NaT	1981-06-30	1982-06-30	1983-06-30	1984-06-29	1985-06-28	1986-06-30	1987-06-30	1988-06-30	1989-06-30	...	2005-06-30	2006-06-30	2007-06-29	2008-06-30	2009-06-30	2010-06-30	2011-06-30	2012-06-29	2013-06-28	20 06-
7	NaT	1981-07-31	1982-07-30	1983-07-29	1984-07-31	1985-07-31	1986-07-31	1987-07-31	1988-07-29	1989-07-31	...	2005-07-29	2006-07-31	2007-07-31	2008-07-31	2009-07-31	2010-07-30	2011-07-29	2012-07-31	2013-07-31	20 07-
8	NaT	1981-08-31	1982-08-31	1983-08-31	1984-08-31	1985-08-30	1986-08-29	1987-08-31	1988-08-31	1989-08-31	...	2005-08-31	2006-08-31	2007-08-31	2008-08-29	2009-08-31	2010-08-31	2011-08-31	2012-08-31	2013-08-30	N
9	NaT	1981-09-30	1982-09-30	1983-09-30	1984-09-28	1985-09-30	1986-09-30	1987-09-30	1988-09-30	1989-09-29	...	2005-09-30	2006-09-29	2007-09-28	2008-09-30	2009-09-30	2010-09-30	2011-09-30	2012-09-28	2013-09-30	N
10	NaT	1981-10-30	1982-10-29	1983-10-31	1984-10-31	1985-10-31	1986-10-31	1987-10-30	1988-10-31	1989-10-31	...	2005-10-31	2006-10-31	2007-10-31	2008-10-31	2009-10-30	2010-10-29	2011-10-31	2012-10-31	2013-10-31	N
11	NaT	1981-11-30	1982-11-30	1983-11-30	1984-11-30	1985-11-29	1986-11-28	1987-11-30	1988-11-30	1989-11-30	...	2005-11-30	2006-11-30	2007-11-30	2008-11-28	2009-11-30	2010-11-30	2011-11-30	2012-11-30	2013-11-29	N

Date_Col	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	20
Date_Col																					
12	1980-12-31	1981-12-31	1982-12-31	1983-12-30	1984-12-31	1985-12-31	1986-12-31	1987-12-31	1988-12-30	1989-12-29	...	2005-12-30	2006-12-29	2007-12-31	2008-12-31	2009-12-31	2010-12-31	2011-12-30	2012-12-31	2013-12-31	N

12 rows × 35 columns

74. What is the difference in days between the first day and the oldest

In [153]:
```
apple.Date_Col.max()-apple.Date_Col.min()
```

Out[153]:
```
Timedelta('12261 days 00:00:00')
```

75. How many months in the data we have?

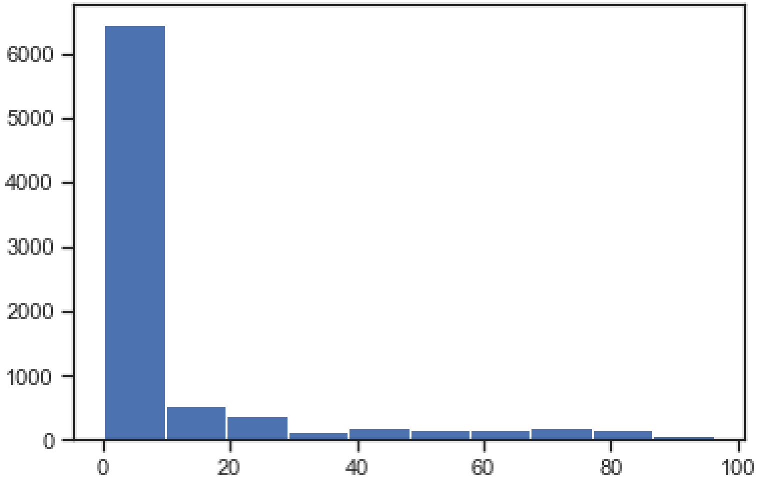
In [161]:
```
timediff=apple.Date_Col.max()-apple.Date_Col.min()
timediff.days/30
```

Out[161]:
```
408.7
```

Section-8: The purpose of the below exercises is to understand how to create basic graphs

76. Plot the 'Adj Close' value. Set the size of the figure to 13.5 x 9 inches

In [174]:
```
plt.hist(apple.Adj_Close)
plt.figure(figsize=(13.5,9))
plt.show()
```



```
<Figure size 972x648 with 0 Axes>
```

The below exercises (77-80) required to use Online_Retail.csv file

77. Import the dataset from this Online_Retail.csv and assign it to a variable called online_rt

In [190]:
```
online_rt=pd.read_csv('Online_Retail.csv')
online_rt
```

Out[190]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	12/9/11 12:50	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	12/9/11 12:50	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	12/9/11 12:50	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	12/9/11 12:50	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	12/9/11 12:50	4.95	12680.0	France

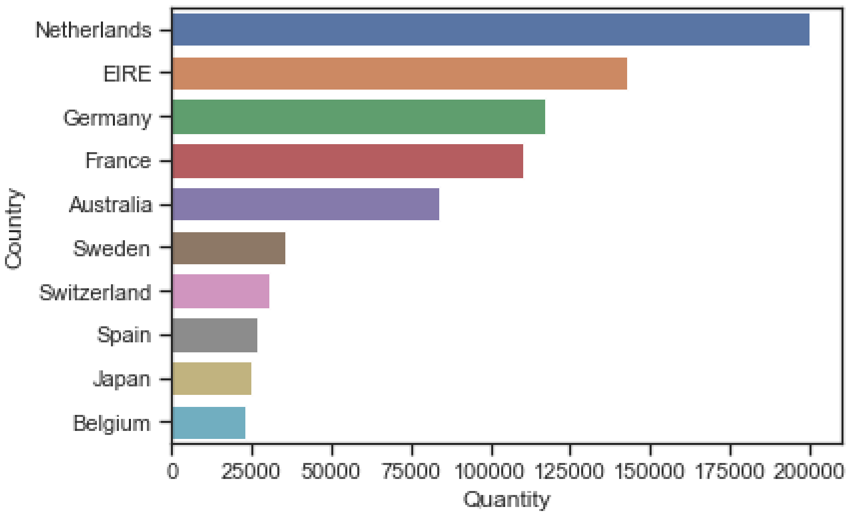
541909 rows × 8 columns

78. Create a barchart with the 10 countries that have the most 'Quantity' ordered except UK

In [203]:
```
plot_c=online_rt[online_rt.Country!='United Kingdom'].groupby('Country')['Quantity'].sum().\
reset_index().sort_values(by='Quantity', ascending=False).head(10)

sns.barplot(x=plot_c.Quantity, y=plot_c.Country)
#plt.show()
```

Out[203]: <AxesSubplot:xlabel='Quantity', ylabel='Country'>



79. Exclude negative Quatity entries

In [208]:
```
online_rt=online_rt[online_rt.Quantity>0]
```

80. Create a scatterplot with the Quantity per UnitPrice by CustomerID for the top 3 Countries

Hint: First we need to find top-3 countries based on revenue, then create scater plot between Quantity and Unitprice for each country separately

In [ ]:
```
online_rt['Revenue']=online_rt.UnitPrice*online_rt.Quantity
```

In [241]:
```
top_coun=online_rt.groupby('Country')['Revenue'].sum().reset_index().sort_values(by='Revenue', ascending=False).head(3)
```

In [250]:
```
top_coun=pd.DataFrame(top_coun)
top_coun
```

Out[250]:

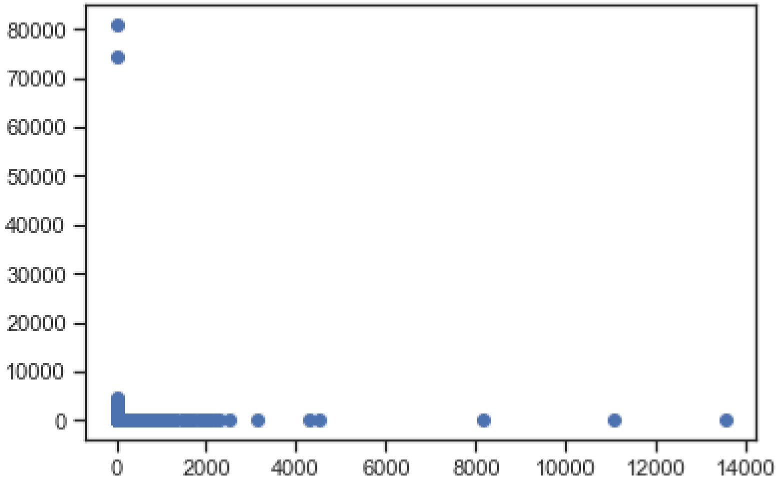
	Country	Revenue
36	United Kingdom	9.003098e+06
24	Netherlands	2.854463e+05
10	EIRE	2.834540e+05

In [253]:
```
coun1=online_rt[online_rt.Country=='United Kingdom']
coun2=online_rt[online_rt.Country=='Netherlands']
coun3=online_rt[online_rt.Country=='EIRE']
```

In [271]:
```
coun1=coun1[coun1.Revenue>0].sort_values(by='UnitPrice')
coun2=coun2[coun2.Revenue>0].sort_values(by='UnitPrice')
coun3=coun3[coun3.Revenue>0].sort_values(by='UnitPrice')
```

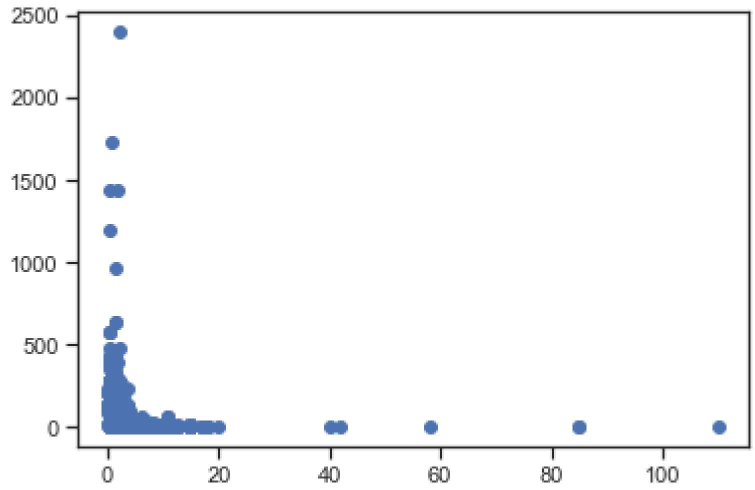
In [265]:
```
plt.scatter(coun1.UnitPrice, coun1.Quantity)
```

Out[265]: <matplotlib.collections.PathCollection at 0x21238f2afa0>



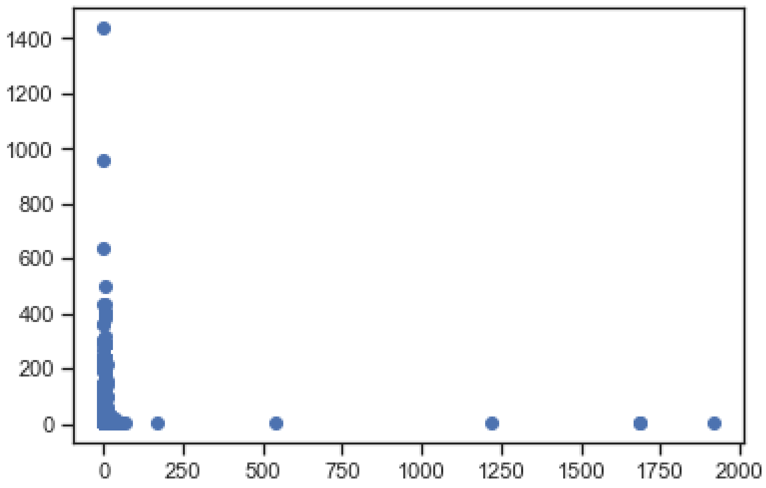
In [272]:
```
plt.scatter(coun2.UnitPrice, coun2.Quantity)
```

Out[272]:
```
<matplotlib.collections.PathCollection at 0x21234326280>
```



In [273]:
```
plt.scatter(coun3.UnitPrice, coun3.Quantity)
```

Out[273]: <matplotlib.collections.PathCollection at 0x21234197c10>



The below exercises (81-90) required to use FMCG_Company_Data_2019.csv file

81. Import the dataset FMCG_Company_Data_2019.csv and assign it to a variable called company_data

In [303]:
```
company_data=pd.read_csv('FMCG_Company_Data_2019.csv')
company_data
```

Out[303]:

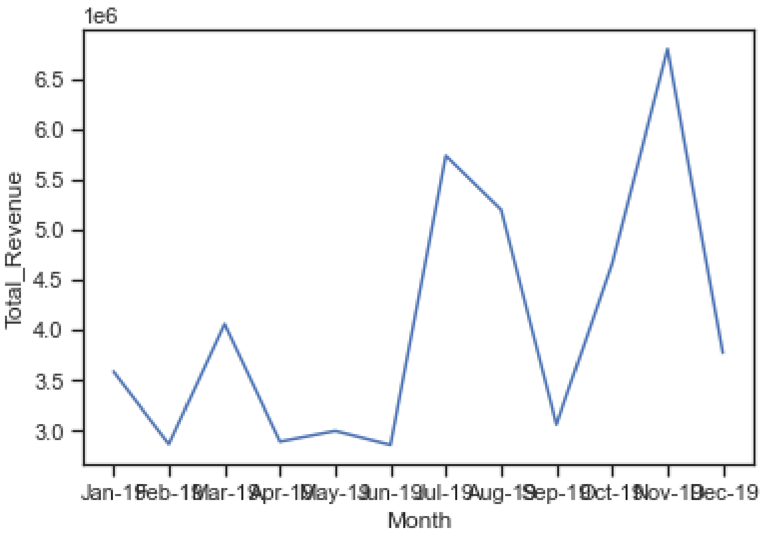
	Month	FaceCream	FaceWash	ToothPaste	Soap	Shampo	Moisturizer	Total_Units	Total_Revenue	Total_Profit
0	Jan-19	2500	1500	5200	9200	1200	1500	21100	3584890	211000
1	Feb-19	2630	1200	5100	6100	2100	1200	18330	2864979	183300
2	Mar-19	2140	1340	4550	9550	3550	1340	22470	4058082	224700
3	Apr-19	3400	1130	5870	8870	1870	1130	22270	2890646	222700
4	May-19	3600	1740	4560	7760	1560	1740	20960	2997280	209600
5	Jun-19	2760	1555	4890	7490	1890	1555	20140	2857866	201400
6	Jul-19	2980	1120	4780	8980	1780	1120	29550	5735655	295500
7	Aug-19	3700	1400	5860	9960	2860	1400	36140	5196932	361400
8	Sep-19	3540	1780	6100	8100	2100	1780	23400	3060720	234000
9	Oct-19	1990	1890	8300	10300	2300	1890	26670	4661916	266700
10	Nov-19	2340	2100	7300	13300	2400	2100	41280	6794688	412800
11	Dec-19	2900	1760	7400	14400	1800	1760	30020	3770512	300200

82. Create line chart for Total Revenue of all months with following properties

- X label name = Month
- Y label name = Total Revenue

In [304]:
```
sns.lineplot(x=company_data.Month, y=company_data.Total_Revenue)
```

Out[304]: <AxesSubplot:xlabel='Month', ylabel='Total_Revenue'>

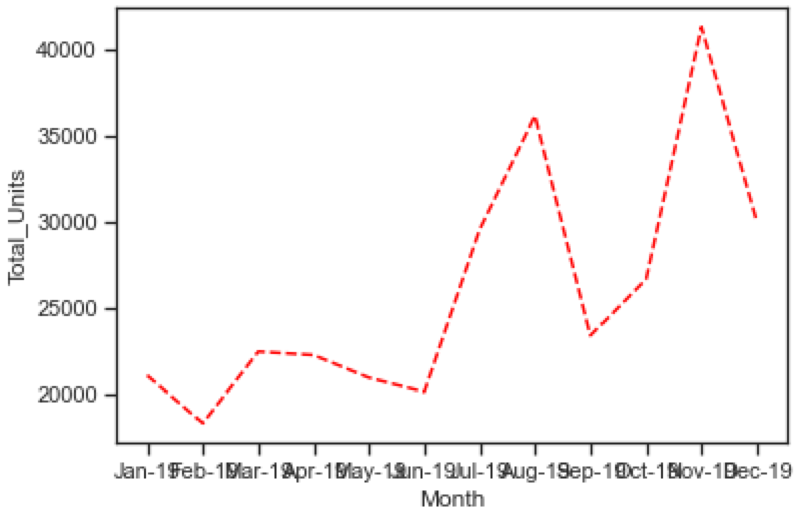


83. Create line chart for Total Units of all months with following properties

- X label name = Month
- Y label name = Total Units
- Line Style dotted and Line-color should be red
- Show legend at the lower right location.

In [312]:
```
sns.lineplot(x=company_data.Month, y=company_data.Total_Units,linestyle="dashed", color='red')
```

Out[312]:
```
<AxesSubplot:xlabel='Month', ylabel='Total_Units'>
```

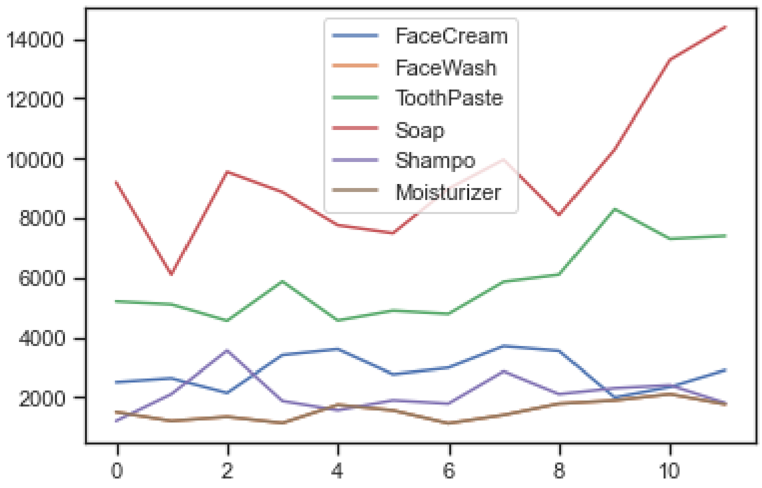


84. Read all product sales data (Facecream, FaceWash, Toothpaste, Soap, Shampo, Moisturizer) and show it using a multiline plot

- Display the number of units sold per month for each product using multiline plots. (i.e., Separate Plotline for each product).

In [315]:
```
company_data[['FaceCream','FaceWash', 'ToothPaste', 'Soap', 'Shampo', 'Moisturizer']].plot()
```

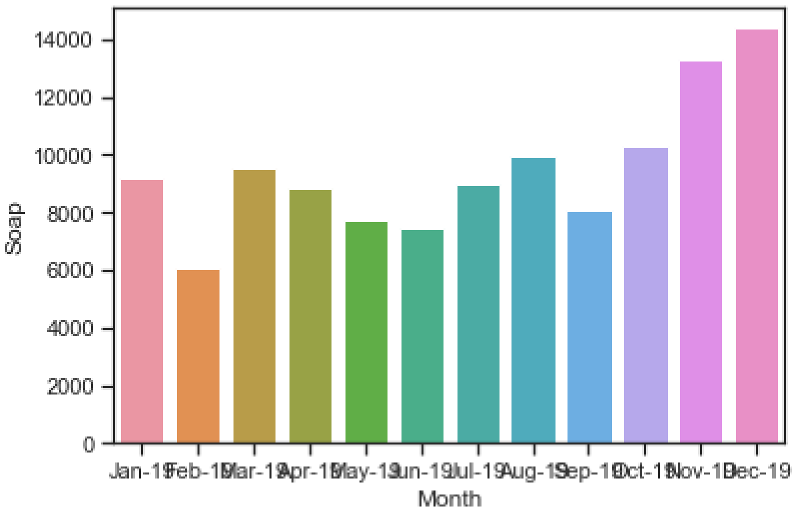
Out[315]: <AxesSubplot:>



85. Create Bar Chart for soap of all months and Save the chart in folder

In [317]:
```
sns.barplot(x=company_data.Month, y=company_data.Soap)
```

Out[317]: <AxesSubplot:xlabel='Month', ylabel='Soap'>

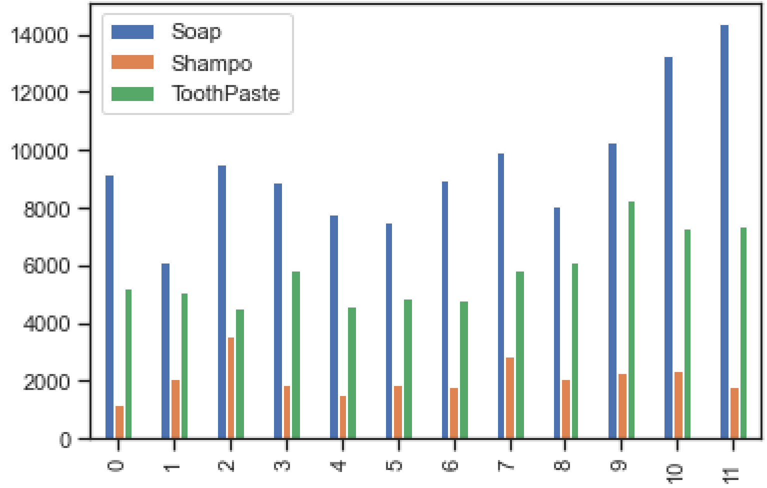


86. Create Stacked Bar Chart for Soap, Shampo, ToothPaste for each month

The bar chart should display the number of units sold per month for each product. Add a separate bar for each product in the same chart.

In [332]:
```
company_data.plot(y=['Soap','Shampo','ToothPaste'],
        kind='bar',
        stacked=False)
```

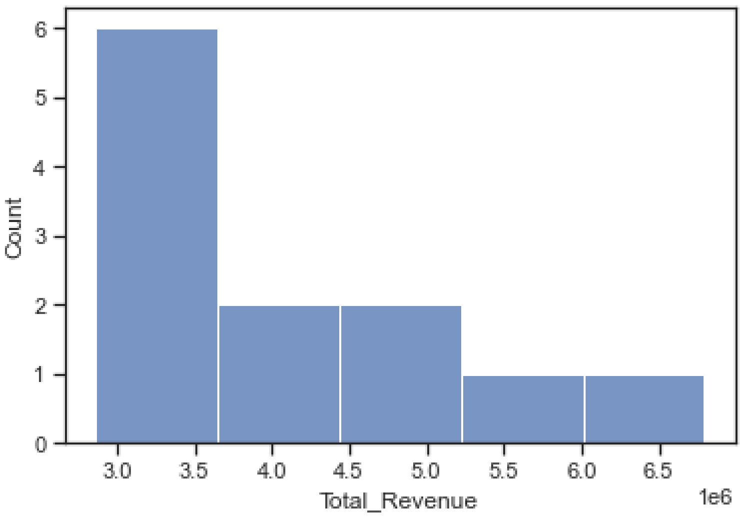
Out[332]: <AxesSubplot:>



87. Create Histogram for Total Revenue

In [20]:
```
sns.histplot(company_data.Total_Revenue)
```

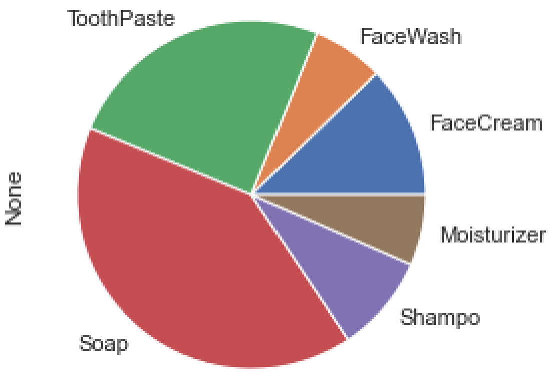
Out[20]: <AxesSubplot:xlabel='Total_Revenue', ylabel='Count'>



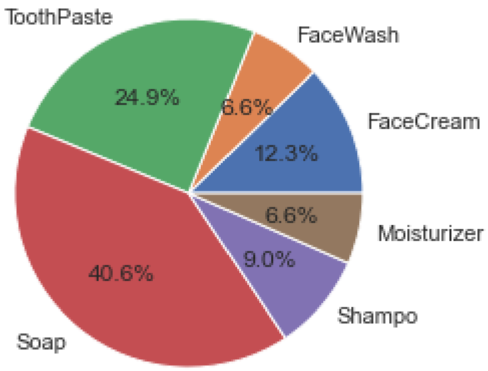
88. Calculate total sales data (quantity) for 2019 for each product and show it using a Pie chart. Understand percentage contribution from each product

In [346]:
```
company_data[['FaceCream','FaceWash', 'ToothPaste', 'Soap', 'Shampo', 'Moisturizer']].sum().plot(kind='pie')
```

Out[346]: <AxesSubplot:ylabel='None'>



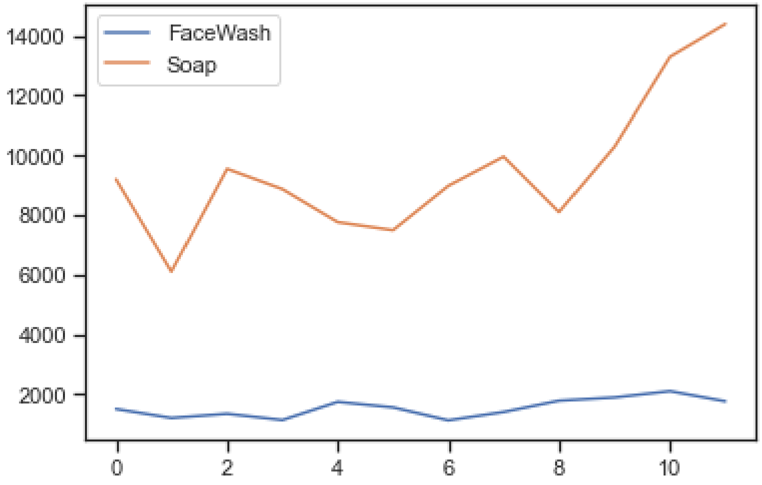
In [349]:
```
pieplot=company_data[['FaceCream','FaceWash', 'ToothPaste', 'Soap', 'Shampo', 'Moisturizer']].sum()
plt.pie(pieplot, labels = pieplot.index, autopct = '%.1f%%')
plt.show()
```



89. Create line plots for Soap & Facewash of all months in a single plot using Subplot

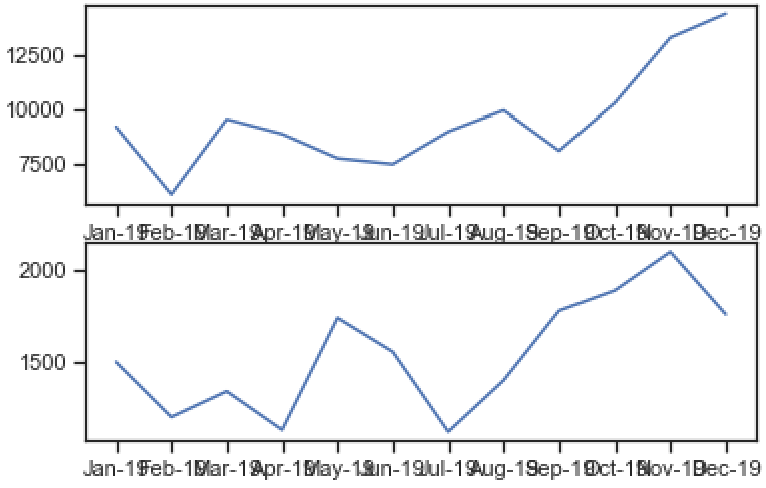
In [333]:
```
company_data[['FaceWash', 'Soap']].plot()
```

Out[333]: <AxesSubplot:>



In [343]:
```
x=company_data.Month
y=company_data.Soap
plt.subplot(2,1,1)
plt.plot(x,y)

x=company_data.Month
y=company_data.FaceWash
plt.subplot(2,1,2)
plt.plot(x,y)
plt.show()
```



90. Create Box Plot for Total Profit variable

In [23]:
```
sns.boxplot(y=company_data.Total_Profit)
```

Out[23]: <AxesSubplot:ylabel='Total_Profit'>