5f4zb1e6y

June 26, 2023

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

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

1.2 The below exercises required to create data

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

[137]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import seaborn as sns
import re
import matplotlib.pyplot as plt

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

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

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

[10]:
```
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']
            }
```

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

[11]:
```
pokemon = pd.DataFrame(raw_data)
pokemon
```

[11]:

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

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

[15]:
```
pokemon = pokemon[["name", "type", "hp", "evolution", "pokedex"]]
pokemon
```

[15]:

	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

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

[16]:
```
place = ["Hills","Volcano","Lakes","park"]
pokemon["Place"] = place
pokemon
```

```
C:\Users\Bhaskar thakur\AppData\Local\Temp\ipykernel_20604\3985941805.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  pokemon["Place"] = place
```

[16]:

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

1.2.6 6. Display the data type of each column

[18]:
```
pokemon.dtypes
```

[18]:
```
name         object
type         object
hp            int64
evolution    object
pokedex      object
Place        object
dtype: object
```

1.2.7 7. Display the info of dataframe

[19]:
```
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
```

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

1.4 The below exercises required to use wine.data

1.4.1 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.

[20]:
```
wine=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Basic Data Manipulation -␣
↪Visualization Exercise_2\Exercise Data Files\wine.txt")
# wine=pd.read_csv('wine.txt')
wine.head()
```

[20]:

	1	14.23	1.71	2.43	15.6	127	2.8	3.06	.28	2.29	5.64	1.04	3.92	\
0	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	

```
1   1  13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81  5.68  1.03  3.17
2   1  14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18  7.80  0.86  3.45
3   1  13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82  4.32  1.04  2.93
4   1  14.20  1.76  2.45  15.2  112  3.27  3.39  0.34  1.97  6.75  1.05  2.85

    1065
0   1050
1   1185
2   1480
3    735
4   1450
```

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

[21]:
```
wine.drop(wine.columns[[0, 3, 6,8,10,12,13]], axis = 1, inplace = True)
wine
```

[21]:
```
     14.23  1.71  15.6  127  3.06  2.29  1.04
0    13.20  1.78  11.2  100  2.76  1.28  1.05
1    13.16  2.36  18.6  101  3.24  2.81  1.03
2    14.37  1.95  16.8  113  3.49  2.18  0.86
3    13.24  2.59  21.0  118  2.69  1.82  1.04
4    14.20  1.76  15.2  112  3.39  1.97  1.05
..     …   …   …   …   …   …   …
172  13.71  5.65  20.5   95  0.61  1.06  0.64
173  13.40  3.91  23.0  102  0.75  1.41  0.70
174  13.27  4.28  20.0  120  0.69  1.35  0.59
175  13.17  2.59  20.0  120  0.68  1.46  0.60
176  14.13  4.10  24.5   96  0.76  1.35  0.61

[177 rows x 7 columns]
```

1.4.3 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

[182]:
```
wine.columns =␣
  ↪["alcohol","malic_acid","alcalinity_of_ash","magnesium","flavanoids","proanthocyanins","hue
wine.head()
```

[182]:
```
   alcohol  malic_acid  alcalinity_of_ash  magnesium  flavanoids  \
0    13.20        1.78               11.2        100        2.76
1    13.16        2.36               18.6        101        3.24
2    14.37        1.95               16.8        113        3.49
3    13.24        2.59               21.0        118        2.69
4    14.20        1.76               15.2        112        3.39

   proanthocyanins   hue
0             1.28  1.05
1             2.81  1.03
2             2.18  0.86
3             1.82  1.04
4             1.97  1.05
```

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

[183]:
```
wine.alcohol.iloc[:3] = np.NAN
```

```
C:\Users\Bhaskar thakur\AppData\Local\Temp\ipykernel_8400\2227344411.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  wine.alcohol.iloc[:3] = np.NAN
```

[184]:
```
wine.head()
```

[184]:
```
   alcohol  malic_acid  alcalinity_of_ash  magnesium  flavanoids  \
0      NaN        1.78               11.2        100        2.76
1      NaN        2.36               18.6        101        3.24
2      NaN        1.95               16.8        113        3.49
3    13.24        2.59               21.0        118        2.69
4    14.20        1.76               15.2        112        3.39

   proanthocyanins   hue
0             1.28  1.05
1             2.81  1.03
2             2.18  0.86
3             1.82  1.04
4             1.97  1.05
```

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

[185]:
```
wine.magnesium.iloc[2:4] = np.NAN
```

```
C:\Users\Bhaskar thakur\AppData\Local\Temp\ipykernel_8400\2186043864.py:1:
```

```
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  wine.magnesium.iloc[2:4] = np.NAN
```

[186]:
```
wine.head()
```

[186]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	\
0	NaN	1.78	11.2	100.0	2.76	
1	NaN	2.36	18.6	101.0	3.24	
2	NaN	1.95	16.8	NaN	3.49	
3	13.24	2.59	21.0	NaN	2.69	
4	14.20	1.76	15.2	112.0	3.39	

	proanthocyanins	hue
0	1.28	1.05
1	2.81	1.03
2	2.18	0.86
3	1.82	1.04
4	1.97	1.05

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

[187]:
```
wine.alcohol = wine.alcohol.fillna(10)
wine.magnesium = wine.magnesium.fillna(100)
wine.head()
```

[187]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	\
0	10.00	1.78	11.2	100.0	2.76	
1	10.00	2.36	18.6	101.0	3.24	
2	10.00	1.95	16.8	100.0	3.49	
3	13.24	2.59	21.0	100.0	2.69	
4	14.20	1.76	15.2	112.0	3.39	

	proanthocyanins	hue
0	1.28	1.05
1	2.81	1.03
2	2.18	0.86
3	1.82	1.04
4	1.97	1.05

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

[188]:
```
x = wine.isnull().sum().sum()
print("Number of mising values in wine dataset is :",x)
```

```
Number of mising values in wine dataset is : 0
```

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

[191]:
```
randNum = np.random.randint(0,11,10)
randNum
```

[191]:
```
array([ 0,  7,  4,  9, 10,  1,  3,  0,  5,  6])
```

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

[193]:
```
wine.alcohol.iloc[randNum] = np.nan
wine
```

[193]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	\
0	NaN	1.78	11.2	100.0	2.76	
1	NaN	2.36	18.6	101.0	3.24	
2	10.00	1.95	16.8	100.0	3.49	
3	NaN	2.59	21.0	100.0	2.69	
4	NaN	1.76	15.2	112.0	3.39	
..	…	…	…	…	…	
172	13.71	5.65	20.5	95.0	0.61	
173	13.40	3.91	23.0	102.0	0.75	
174	13.27	4.28	20.0	120.0	0.69	
175	13.17	2.59	20.0	120.0	0.68	
176	14.13	4.10	24.5	96.0	0.76	

	proanthocyanins	hue
0	1.28	1.05
1	2.81	1.03
2	2.18	0.86
3	1.82	1.04
4	1.97	1.05
..	…	…
172	1.06	0.64
173	1.41	0.70
174	1.35	0.59
175	1.46	0.60
176	1.35	0.61

```
[177 rows x 7 columns]
```

1.4.10 17. How many missing values do we have now?

[194]:
```
t = wine.alcohol.isnull().sum()
print("Number of mising values in wine dataset is :",t)
```

```
Number of mising values in wine dataset is : 9
```

1.4.11 18. Print only the non-null values in alcohol

[195]:
```
y = wine.alcohol.dropna()
y
```

[195]:
```
2      10.00
8      13.86
11     13.75
12     14.75
13     14.38
       …
172    13.71
173    13.40
174    13.27
175    13.17
176    14.13
Name: alcohol, Length: 168, dtype: float64
```

1.4.12 19. Delete the rows that contain missing values

[196]:
```
wine = wine.dropna()
wine
```

[196]:

	alcohol	malic_acid	alcalinity_of_ash	magnesium	flavanoids	\
2	10.00	1.95	16.8	100.0	3.49	
8	13.86	1.35	16.0	98.0	3.15	
11	13.75	1.73	16.0	89.0	2.76	
12	14.75	1.73	11.4	91.0	3.69	
13	14.38	1.87	12.0	102.0	3.64	
..	…	…	…	…	…	
172	13.71	5.65	20.5	95.0	0.61	
173	13.40	3.91	23.0	102.0	0.75	
174	13.27	4.28	20.0	120.0	0.69	
175	13.17	2.59	20.0	120.0	0.68	
176	14.13	4.10	24.5	96.0	0.76	

	proanthocyanins	hue
2	2.18	0.86
8	1.85	1.01
11	1.81	1.15

```
12              2.81  1.25
13              2.96  1.20
..               …    …
172             1.06  0.64
173             1.41  0.70
174             1.35  0.59
175             1.46  0.60
176             1.35  0.61

[168 rows x 7 columns]
```

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

[197]:
```
wine.reset_index(drop=True)
```

[197]:
```
     alcohol  malic_acid  alcalinity_of_ash  magnesium  flavanoids  \
0      10.00        1.95               16.8      100.0        3.49
1      13.86        1.35               16.0       98.0        3.15
2      13.75        1.73               16.0       89.0        2.76
3      14.75        1.73               11.4       91.0        3.69
4      14.38        1.87               12.0      102.0        3.64
..       …          …                  …         …          …
163    13.71        5.65               20.5       95.0        0.61
164    13.40        3.91               23.0      102.0        0.75
165    13.27        4.28               20.0      120.0        0.69
166    13.17        2.59               20.0      120.0        0.68
167    14.13        4.10               24.5       96.0        0.76

     proanthocyanins   hue
0               2.18  0.86
1               1.85  1.01
2               1.81  1.15
3               2.81  1.25
4               2.96  1.20
..               …    …
163             1.06  0.64
164             1.41  0.70
165             1.35  0.59
166             1.46  0.60
167             1.35  0.61

[168 rows x 7 columns]
```

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

1.6 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/c
and create dataframe called chipo

[121]:
```
chipo=pd.read_table("../Basic Data Manipulation - Visualization␣
↪Exercise_2\Exercise Data Files/chipotle.tsv")
chipo
```

[121]:
```
      order_id  quantity                              item_name  \
0            1         1           Chips and Fresh Tomato Salsa
1            1         1                                   Izze
2            1         1                       Nantucket Nectar
3            1         1  Chips and Tomatillo-Green Chili Salsa
4            2         2                           Chicken Bowl
…          …       …                                      …
4617      1833         1                       Steak Burrito
4618      1833         1                       Steak Burrito
4619      1834         1                  Chicken Salad Bowl
4620      1834         1                  Chicken Salad Bowl
4621      1834         1                  Chicken Salad Bowl

                                  choice_description item_price
0                                                NaN     $2.39
1                                       [Clementine]     $3.39
2                                            [Apple]     $3.39
3                                                NaN     $2.39
4     [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98
…                                                  …        …
4617  [Fresh Tomato Salsa, [Rice, Black Beans, Sour …    $11.75
4618  [Fresh Tomato Salsa, [Rice, Sour Cream, Cheese…    $11.75
4619  [Fresh Tomato Salsa, [Fajita Vegetables, Pinto…    $11.25
4620  [Fresh Tomato Salsa, [Fajita Vegetables, Lettu…     $8.75
4621  [Fresh Tomato Salsa, [Fajita Vegetables, Pinto…     $8.75

[4622 rows x 5 columns]
```

1.6.1 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.

[122]:
```
chipo['item_price']=chipo['item_price'].str.replace('$',' ').apply(pd.
↪to_numeric)
```

```
C:\Users\Bhaskar thakur\AppData\Local\Temp\ipykernel_20604\1599270324.py:1:
FutureWarning: The default value of regex will change from True to False in a
future version. In addition, single character regular expressions will *not* be
treated as literal strings when regex=True.
  chipo['item_price']=chipo['item_price'].str.replace('$','
').apply(pd.to_numeric)
```

[123]:
```
chipo['price_per_item']=chipo['item_price']/chipo['quantity']
```

[124]:
```
product_vs_price=chipo.groupby('item_name')[['price_per_item']].max().
↪reset_index()
product_vs_price.query('price_per_item>10').item_name.nunique()
```

[124]: 25

1.6.2 22. Print the Chipo Dataframe & info about data frame

[ ]:
```
chipotle.info()
```

1.6.3 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

[139]:
```
chipo1 = chipo.groupby(by ="item_name")[["item_price"]].min().reset_index()
chipo1
```

[139]:

	item_name	item_price
0	6 Pack Soft Drink	6.49
1	Barbacoa Bowl	8.69
2	Barbacoa Burrito	8.69
3	Barbacoa Crispy Tacos	8.99
4	Barbacoa Salad Bowl	9.39
5	Barbacoa Soft Tacos	8.99
6	Bottled Water	1.09
7	Bowl	7.40
8	Burrito	7.40
9	Canned Soda	1.09
10	Canned Soft Drink	1.25
11	Carnitas Bowl	8.99
12	Carnitas Burrito	8.69
13	Carnitas Crispy Tacos	8.99
14	Carnitas Salad	8.99
15	Carnitas Salad Bowl	9.39
16	Carnitas Soft Tacos	8.99
17	Chicken Bowl	8.19

```
18                       Chicken Burrito        8.19
19                   Chicken Crispy Tacos        8.49
20                         Chicken Salad        8.19
21                     Chicken Salad Bowl        8.75
22                     Chicken Soft Tacos        8.49
23                                  Chips        1.99
24            Chips and Fresh Tomato Salsa        2.29
25                    Chips and Guacamole        3.89
26       Chips and Mild Fresh Tomato Salsa        3.00
27      Chips and Roasted Chili Corn Salsa        2.95
28      Chips and Roasted Chili-Corn Salsa        2.39
29  Chips and Tomatillo Green Chili Salsa        2.95
30    Chips and Tomatillo Red Chili Salsa        2.95
31  Chips and Tomatillo-Green Chili Salsa        2.39
32    Chips and Tomatillo-Red Chili Salsa        2.39
33                            Crispy Tacos        7.40
34                                    Izze        3.39
35                      Nantucket Nectar        3.39
36                                  Salad        7.40
37                         Side of Chips        1.69
38                            Steak Bowl        8.69
39                         Steak Burrito        8.69
40                     Steak Crispy Tacos        8.69
41                           Steak Salad        8.69
42                       Steak Salad Bowl        9.39
43                       Steak Soft Tacos        8.99
44                            Veggie Bowl        8.49
45                         Veggie Burrito        8.49
46                     Veggie Crispy Tacos        8.49
47                           Veggie Salad        8.49
48                       Veggie Salad Bowl        8.75
49                       Veggie Soft Tacos        8.49
```

1.6.4 24. Sort by the name of the item

[140]:
```
chipo.sort_values(by = "item_name",ascending=True).reset_index(drop=True)
```

[140]:
```
      order_id  quantity          item_name  \
0         1360         2  6 Pack Soft Drink
1          148         1  6 Pack Soft Drink
2          749         1  6 Pack Soft Drink
3          754         1  6 Pack Soft Drink
4         1076         1  6 Pack Soft Drink
…          …         …                  …
4617       948         1  Veggie Soft Tacos
4618       322         1  Veggie Soft Tacos
4619      1132         1  Veggie Soft Tacos
```

```
4620       688         1  Veggie Soft Tacos
4621       567         1  Veggie Soft Tacos

                                     choice_description  item_price
0                                             [Diet Coke]       12.98
1                                             [Diet Coke]        6.49
2                                                  [Coke]        6.49
3                                             [Diet Coke]        6.49
4                                                  [Coke]        6.49
…                                                     …          …
4617  [Roasted Chili Corn Salsa, [Fajita Vegetables,…        8.75
4618  [Fresh Tomato Salsa, [Black Beans, Cheese, Sou…        8.75
4619  [Roasted Chili Corn Salsa (Medium), [Black Bea…        8.49
4620  [Fresh Tomato Salsa, [Fajita Vegetables, Rice,…       11.25
4621  [Fresh Tomato Salsa (Mild), [Pinto Beans, Rice…        8.49

[4622 rows x 5 columns]
```

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

[141]:
```
xx = np.where(chipo.item_price == chipo.item_price.max(),chipo.quantity,0)
for i in xx:
    if i!=0:
        print("Quantity of the most expensive item ordered is : ",i)
```

```
Quantity of the most expensive item ordered is :  15
```

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

[143]:
```
temp = np.count_nonzero(chipo.item_name == "Veggie Salad Bowl")
print(temp,"times a Veggie Salad Bowl was ordered")
```

```
18 times a Veggie Salad Bowl was ordered
```

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

[142]:
```
temp1 = np.count_nonzero((chipo.item_name == "Canned Soda") & (chipo.quantity >␣
↪1))
print(temp1,"times people orderd more than one Canned Soda")
```

```
20 times people orderd more than one Canned Soda
```

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

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

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

[106]:
```
users=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Basic Data Manipulation - Visualization Exercise_2\Exercise Data Files\occupation.csv",sep="|")
users.head(2)
```

[106]:
```
   user_id  age gender  occupation zip_code
0        1   24      M  technician    85711
1        2   53      F       other    94043
```

1.8.2 29. Discover what is the mean age per occupation

[107]:
```
users.groupby('occupation').age.mean()
```

[107]:
```
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
```

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

Use numpy.where() to encode gender column.

[108]:
```
Q=(pd.crosstab(users['occupation'],users['gender'])['M']/users.occupation.
  ↪value_counts().sort_index()).reset_index()
Q.rename(columns={0:'Male ratio'}).sort_values(by='Male ratio',ascending=False).
  ↪reset_index(drop=True)
```

[108]:

	occupation	Male ratio
0	doctor	1.000000
1	engineer	0.970149
2	technician	0.962963
3	retired	0.928571
4	programmer	0.909091
5	executive	0.906250
6	scientist	0.903226
7	entertainment	0.888889
8	lawyer	0.833333
9	salesman	0.750000
10	educator	0.726316
11	student	0.693878
12	other	0.657143
13	marketing	0.615385
14	writer	0.577778
15	none	0.555556
16	administrator	0.544304
17	artist	0.535714
18	librarian	0.431373
19	healthcare	0.312500
20	homemaker	0.142857

[ ]:
```

```

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

[ ]:
```
print(users.groupby('occupation').age.min())
print(users.groupby('occupation').age.max())
```

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

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

[ ]:
```
users.groupby(['occupation', 'gender']).age.mean()
```

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

[ ]:
```
# create a data frame and apply count to gender
gender_ocup = users.groupby(['occupation', 'gender']).agg({'gender': 'count'})

# create a DataFrame and apply count for each occupation
occup_count = users.groupby(['occupation']).count()

# divide the gender_ocup per the occup_count and multiply per 100
occup_gender = gender_ocup.div(occup_count, level = "occupation")
occup_gender.loc[:, 'gender']
```

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

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

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

[93]:
```
mat=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics assignments\Basic␣
↪Data Manipulation - Visualization Exercise_2\Exercise Data Files\student-mat.
↪csv")
por=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics assignments\Basic␣
↪Data Manipulation - Visualization Exercise_2\Exercise Data Files\student-por.
↪csv")
mat.head(2)
```

[93]:

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

	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	4	3	4	1	1	3	6	5	6	6
1	5	3	3	1	1	3	4	5	5	6

[2 rows x 33 columns]

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

[97]:
```
mat.loc[:,"school":"guardian"]
```

[97]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	\
0	GP	F	180	U	GT3	A	40	40	at_home	teacher	
1	GP	F	170	U	GT3	T	10	10	at_home	other	

```
2       GP   F  150       U     LE3       T     10     10   at_home     other
3       GP   F  150       U     GT3       T     40     20    health  services
4       GP   F  160       U     GT3       T     30     30     other     other
..     …   ..  …     …     …     …   …   …       …       …
390     MS   M  200       U     LE3       A     20     20  services  services
391     MS   M  170       U     LE3       T     30     10  services  services
392     MS   M  210       R     GT3       T     10     10     other     other
393     MS   M  180       R     LE3       T     30     20  services     other
394     MS   M  190       U     LE3       T     10     10     other   at_home

     reason guardian
0    course   mother
1    course   father
2     other   mother
3      home   mother
4      home   father
..      …       …
390  course    other
391  course   mother
392  course    other
393  course   mother
394  course   father

[395 rows x 12 columns]
```

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

[14]:
```
# define the lambda function
capitalize_str = lambda s: s.capitalize()

# test the function
input_str = 'at_home'
output_str = capitalize_str(input_str)
print(output_str)  # prints 'At_home'
```

```
At_home
```

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

[ ]:
```
mat.Mjob = mat.Mjob.apply(capitalize_str())
mat.Fjob = mat.Fjob.apply(capitalize_str())
mat.head()
```

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

[18]:
```
mat.tail()
```

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

     … famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
390  …      5        5      4     4     5      4       11   9   9   9
391  …      2        4      5     3     4      2        3  14  16  16
392  …      5        5      3     3     3      3        3  10   8   7
393  …      4        4      1     3     4      5        0  11  12  10
394  …      3        2      3     3     3      5        5   8   9   9

[5 rows x 33 columns]
```

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

[21]:
```
def Capital():
    return lambda x: x.capitalize()
mat.Mjob = mat.Mjob.apply(Capital())
mat.Fjob = mat.Fjob.apply(Capital())
mat.head()
```

[21]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  … \
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  …

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]
```

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

[96]:
```
def majority(x):
    if x == 1:
        return True
    else:
        return False
mat["legal_drinker"] = [majority(1) if x>=18 else majority(0) for x in␣
 ↪mat["age"]]
mat.head()
```

[96]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  … \
0     GP   F  180       U     GT3       A    40    40  at_home   teacher  …
1     GP   F  170       U     GT3       T    10    10  at_home     other  …
2     GP   F  150       U     LE3       T    10    10  at_home     other  …
3     GP   F  150       U     GT3       T    40    20   health  services  …
4     GP   F  160       U     GT3       T    30    30    other     other  …

  freetime goout  Dalc  Walc  health absences   G1   G2   G3 legal_drinker
0       30    40    10    10      30       60   50   60   60          True
1       30    30    10    10      30       40   50   50   60          True
2       30    20    20    30      30      100   70   80  100          True
3       20    20    10    10      50       20  150  140  150          True
4       30    20    10    20      50       40   60  100  100          True

[5 rows x 34 columns]
```

1.10.8 41. Multiply every number of the dataset by 10.

[94]:
```
Q=mat.select_dtypes(include='number')*10
mat[Q.columns]=Q
mat
```

[94]:
```
    school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
0       GP   F  180       U     GT3       A    40    40   at_home   teacher
1       GP   F  170       U     GT3       T    10    10   at_home     other
2       GP   F  150       U     LE3       T    10    10   at_home     other
3       GP   F  150       U     GT3       T    40    20    health  services
4       GP   F  160       U     GT3       T    30    30     other     other
..     …   ..  …     …     …       …   …   …        …        …
390     MS   M  200       U     LE3       A    20    20  services  services
391     MS   M  170       U     LE3       T    30    10  services  services
392     MS   M  210       R     GT3       T    10    10     other     other
393     MS   M  180       R     LE3       T    30    20  services     other
394     MS   M  190       U     LE3       T    10    10     other   at_home
```

```
     … famrel freetime  goout  Dalc  Walc health absences   G1   G2   G3
0    …     40       30     40    10    10     30       60   50   60   60
1    …     50       30     30    10    10     30       40   50   50   60
2    …     40       30     20    20    30     30      100   70   80  100
3    …     30       20     20    10    10     50       20  150  140  150
4    …     40       30     20    10    20     50       40   60  100  100
..   …    …       …     …    …    …     …       …   …  …  …
390  …     50       50     40    40    50     40      110   90   90   90
391  …     20       40     50    30    40     20       30  140  160  160
392  …     50       50     30    30    30     30       30  100   80   70
393  …     40       40     10    30    40     50        0  110  120  100
394  …     30       20     30    30    30     50       50   80   90   90

[395 rows x 33 columns]
```

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

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

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

[69]:
```
cars1=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics␣
 ↪assignments\Basic Data Manipulation - Visualization Exercise_2\Exercise Data␣
 ↪Files\cars1.csv")
cars2=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics␣
 ↪assignments\Basic Data Manipulation - Visualization Exercise_2\Exercise Data␣
 ↪Files\cars2.csv")
cars1
cars2
```

[69]:
```
      mpg  cylinders  displacement horsepower  weight  acceleration  model  \
0    33.0          4            91         53    1795          17.4     76
1    20.0          6           225        100    3651          17.7     76
2    18.0          6           250         78    3574          21.0     76
3    18.5          6           250        110    3645          16.2     76
4    17.5          6           258         95    3193          17.8     76
..    …         …           …        …     …           …     …
195  27.0          4           140         86    2790          15.6     82
196  44.0          4            97         52    2130          24.6     82
197  32.0          4           135         84    2295          11.6     82
198  28.0          4           120         79    2625          18.6     82
199  31.0          4           119         82    2720          19.4     82

     origin                 car
0         3         honda civic
1         1      dodge aspen se
```

```
2          1   ford granada ghia
3          1  pontiac ventura sj
4          1       amc pacer d/l
..       …                   …
195        1     ford mustang gl
196        2           vw pickup
197        1       dodge rampage
198        1         ford ranger
199        1           chevy s-10

[200 rows x 9 columns]
```

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

[ ]:
```
type(cars1)
```

[ ]:
```
cars1.head(3)
```

[ ]:
```
cars1.tail(3)
```

[ ]:
```
cars1.columns
```

[ ]:
```
cars1.info()
```

[ ]:
```
cars1.dtypes
```

[ ]:
```
cars1.index
```

[ ]:
```
cars1.shape
```

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

[ ]:
```
cars1.size
```

[ ]:
```
cars1.ndim
```

[82]:
```
cars1.axes
```

[82]:
```
[RangeIndex(start=0, stop=198, step=1),
 Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
        'acceleration', 'model', 'origin', 'car'],
       dtype='object')]
```

[83]:
```
cars1.describe()
```

[83]:
```
              mpg   cylinders  displacement       weight  acceleration  \
count  198.000000  198.000000    198.000000   198.000000    198.000000
mean    19.719697    5.898990    223.469697  3177.888889     15.005556
std      5.814254    1.785417    115.181017   934.783733      2.872382
min      9.000000    3.000000     68.000000  1613.000000      8.000000
25%     15.000000    4.000000    113.250000  2302.500000     13.000000
50%     19.000000    6.000000    228.000000  3030.000000     15.000000
75%     24.375000    8.000000    318.000000  4080.750000     16.800000
max     35.000000    8.000000    455.000000  5140.000000     23.500000

            model      origin
count  198.000000  198.000000
mean    72.818182    1.439394
std      1.865332    0.708085
min     70.000000    1.000000
25%     71.000000    1.000000
50%     73.000000    1.000000
75%     74.000000    2.000000
max     76.000000    3.000000
```

[ ]:
```
cars1.memory_usage()
```

[ ]:
```
cars1.car.value_counts()
```

[ ]:
```
cars1.memory_usage()
```

[ ]:
```
ProfileReport(cars1)
```

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

[78]:
```
cars1=cars1.dropna(axis=1)
cars1
```

[78]:
```
      mpg  cylinders  displacement horsepower  weight  acceleration  model  \
0    18.0          8           307        130    3504          12.0     70
1    15.0          8           350        165    3693          11.5     70
2    18.0          8           318        150    3436          11.0     70
3    16.0          8           304        150    3433          12.0     70
4    17.0          8           302        140    3449          10.5     70
..    …          …           …          …      …           …      …
193  24.0          6           200         81    3012          17.6     76
194  22.5          6           232         90    3085          17.6     76
195  29.0          4            85         52    2035          22.2     76
196  24.5          4            98         60    2164          22.1     76
197  29.0          4            90         70    1937          14.2     76

     origin                       car
```

```
0         1  chevrolet chevelle malibu
1         1          buick skylark 320
2         1         plymouth satellite
3         1              amc rebel sst
4         1                ford torino
..      …                        …
193       1              ford maverick
194       1                 amc hornet
195       1         chevrolet chevette
196       1            chevrolet woody
197       2                  vw rabbit

[198 rows x 9 columns]
```

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

[77]:
```
print('the number of observations in cars1 is',cars1.index.size)
print('the number of observations in cars2 is',cars2.index.size)
```

```
the number of observations in cars1 is 198
the number of observations in cars2 is 200
```

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

[72]:
```
cars = pd.concat([cars1,cars2],axis=0)
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 398 entries, 0 to 199
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    int64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model         398 non-null    int64
 7   origin        398 non-null    int64
 8   car           398 non-null    object
dtypes: float64(2), int64(5), object(2)
memory usage: 31.1+ KB
```

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

[74]:
```
random_num=np.random.randint(15000,73000,cars.index.size)
random_num
```

1.12.6 48. Add the column owners to cars

[75]:
```
cars['owners']=random_num
cars
```

[75]:
```
      mpg  cylinders  displacement horsepower  weight  acceleration  model  \
0    18.0          8           307        130    3504          12.0     70
1    15.0          8           350        165    3693          11.5     70
2    18.0          8           318        150    3436          11.0     70
3    16.0          8           304        150    3433          12.0     70
4    17.0          8           302        140    3449          10.5     70
..    …          …             …          …      …             …      …
195  27.0          4           140         86    2790          15.6     82
196  44.0          4            97         52    2130          24.6     82
197  32.0          4           135         84    2295          11.6     82
198  28.0          4           120         79    2625          18.6     82
199  31.0          4           119         82    2720          19.4     82

     origin                       car  owners
0         1  chevrolet chevelle malibu   29042
1         1          buick skylark 320   35709
2         1         plymouth satellite   29068
3         1              amc rebel sst   60562
4         1                ford torino   32838
..      …                          …      …
195       1            ford mustang gl   58158
196       2                  vw pickup   58425
197       1              dodge rampage   32429
198       1                ford ranger   34417
199       1                 chevy s-10   32438

[398 rows x 10 columns]
```

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

1.13.1 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

[125]:
```
import time
import datetime
print("Current date and time: " , datetime.datetime.now())
print("Current year: ", datetime.date.today().strftime("%Y"))
print("Month of year: ", datetime.date.today().strftime("%B"))
print("Week number of the year: ", datetime.date.today().strftime("%W"))
print("Weekday of the week: ", datetime.date.today().strftime("%w"))
print("Day of year: ", datetime.date.today().strftime("%j"))
print("Day of the month : ", datetime.date.today().strftime("%d"))
print("Day of week: ", datetime.date.today().strftime("%A"))
```

```
Current date and time:  2023-03-24 00:40:52.100057
Current year:  2023
Month of year:  March
Week number of the year:  12
Weekday of the week:  5
Day of year:  083
Day of the month :  24
Day of week:  Friday
```

1.13.2 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

[126]:
```
from datetime import datetime
date_object = datetime.strptime('Jul 1 2014 2:43PM', '%b %d %Y %I:%M%p')
print(date_object)
```

```
2014-07-01 14:43:00
```

1.13.3 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

[127]:
```
from datetime import date, timedelta
dt = date.today() - timedelta(5)
print('Current Date :',date.today())
print('5 days before Current Date :',dt)
```

```
Current Date : 2023-03-24
5 days before Current Date : 2023-03-19
```

1.13.4 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

[128]:
```
import datetime
print(
    datetime.datetime.fromtimestamp(
        int("1284105682")
    ).strftime('%Y-%m-%d %H:%M:%S')
)
```

```
2010-09-10 13:31:22
```

1.13.5 53. Convert the below Series to pandas datetime :

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

Make sure that the year is 19XX not 20XX

[ ]:
```
DoB = pd.Series(["07Sep59","01Jan55","15Dec47","11Jul42"])
pd.to_datetime(DoB)-pd.DateOffset(years=100)
```

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

[130]:
```
from datetime import date
f_date = date(2022, 7, 2)
l_date = date(2022, 7, 11)
delta = l_date - f_date
print(delta.days)
```

```
9
```

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

Date = “15Dec1989”

Result : “Friday, 15 Dec 98”

[ ]:
```
A=pd.to_datetime(input('date in formet like 15Dec1989 ')).strftime('%A, %d %b␣
↪%Y')
A
```

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

1.14.1 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.

1.14.2 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

[ ]:
```
data=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Basic Data Manipulation -␣
↪Visualization Exercise_2\Exercise Data Files\Exercise Data Files/wind.
↪data",parse_dates=[['Yr','Mo','Dy']])
data
```

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

[ ]:
```
data["Date"] = np.where(pd.DatetimeIndex(data["Date"]).year < 2000,data.
↪Date,data.Date - pd.offsets.DateOffset(years=100))
```

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

[ ]:
```
newData = data.set_index("Date")
newData.index.astype("datetime64[ns]")
```

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

They should be ignored in all calculations below.

[ ]:
```
print(newData.isnull().values.ravel().sum())
```

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

[ ]:
```
print((data.isna()==False).sum().sum())
(data.isna()==False).sum()
```

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

A single number for the entire dataset.

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

1.14.8 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.

[ ]:
```
loc_stats=data.describe().loc[['min','max','mean','std'],:]
loc_stats
```

1.14.9 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.

[ ]:
```
day_stats = newData.apply(stats,axis=1)
day_stats.head()
```

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

Treat January 1961 and January 1962 both as January.

[ ]:
```
january_data = newData[newData.index.month == 1]
print ("January windspeeds:")
print (january_data.mean())
```

1.14.11 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.)

[ ]:
```
data.groupby([data.index.year,data.index.month]).mean()
```

1.14.12 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.

[ ]:
```
data.resample('W',loffset=pd.DateOffset(days=1)).mean()
```

[ ]:
```
T.describe().loc[['min','max','mean','std'],:].head(52)
```

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

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

[41]:
```
apple=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics␣
↪assignments\Basic Data Manipulation - Visualization Exercise_2\Exercise Data␣
↪Files\appl_1980_2014.csv")
apple
```

[41]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	2014-07-08	96.27	96.80	93.92	95.35	65130000	95.35
1	2014-07-07	94.14	95.99	94.10	95.97	56305400	95.97
2	2014-07-03	93.67	94.10	93.20	94.03	22891800	94.03
3	2014-07-02	93.87	94.06	93.09	93.48	28420900	93.48
4	2014-07-01	93.52	94.07	93.13	93.52	38170200	93.52
…	…	…	…	…	…	…	…
8460	1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
8461	1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
8462	1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
8463	1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
8464	1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45

[8465 rows x 7 columns]

1.15.2 68. Check out the type of the columns

[42]:
```
apple.dtypes
```

[42]:
```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Volume         int64
Adj Close    float64
dtype: object
```

1.15.3 69. Transform the Date column as a datetime type

[43]:
```
apple.Date=pd.to_datetime(apple.Date)
apple.dtypes
```

[43]:
```
Date         datetime64[ns]
Open                float64
High                float64
Low                 float64
```

```
Close                float64
Volume                 int64
Adj Close            float64
dtype: object
```

1.15.4 70. Set the date as the index

[ ]:
```
apple.set_index('Date',inplace=True)
apple
```

1.15.5 71. Is there any duplicate dates?

[12]:
```
x = apple[apple.duplicated("Date")]
if len(x) != 0:
    print("Yes there are duplicates in date column")
else:
    print("No there are no duplicates in date column")
```

```
No there are no duplicates in date column
```

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

[13]:
```
apple = apple.sort_values(by="Date",ascending=True).reset_index(drop=True)
apple
```

[13]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45
1	1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
2	1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
3	1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
4	1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
…	…	…	…	…	…	…	…
8460	2014-07-01	93.52	94.07	93.13	93.52	38170200	93.52
8461	2014-07-02	93.87	94.06	93.09	93.48	28420900	93.48
8462	2014-07-03	93.67	94.10	93.20	94.03	22891800	94.03
8463	2014-07-07	94.14	95.99	94.10	95.97	56305400	95.97
8464	2014-07-08	96.27	96.80	93.92	95.35	65130000	95.35

[8465 rows x 7 columns]

1.15.7 73. Get the last business day of each month

[45]:
```
apple['day']=apple.index.day
apple.groupby([apple.index.year,apple.index.month_name()]).day.last()
```

[45]:
```
Date  Date
1980  December    12
1981  April        1
      August       3
      December     1
      February     2
                  ..
2014  January      2
      July         1
      June         2
      March        3
      May          1
Name: day, Length: 404, dtype: int64
```

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

[46]:
```
apple.index.max()-apple.index.min()
```

[46]:
```
Timedelta('12261 days 00:00:00')
```

1.15.9 75. How many months in the data we have?

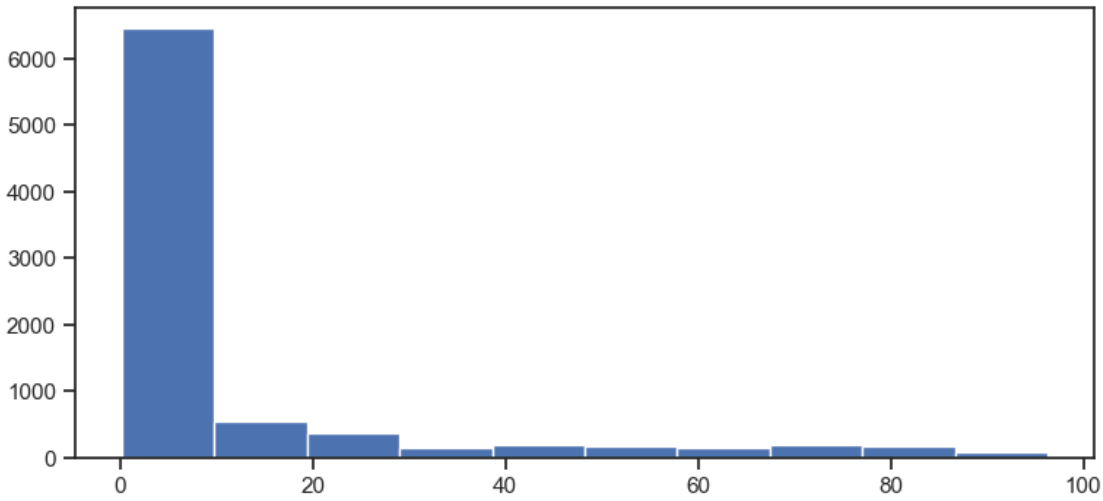
[47]:
```
len(apple.groupby([apple.index.year,apple.index.month]).count())
```

[47]:
```
404
```

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

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

[20]:
```
plt.figure(figsize=(9, 4))
plt.hist(apple["Adj Close"])
plt.show()
```



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

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

[51]:
```
online_rt =pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics␣
↪assignments\Basic Data Manipulation - Visualization Exercise_2\Exercise Data␣
↪Files\Online_Retail.csv",encoding='windows-1252')
online_rt
```

[51]:

	InvoiceNo	StockCode	Description	Quantity	\
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	
1	536365	71053	WHITE METAL LANTERN	6	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	
…	…	…	…	…	
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	

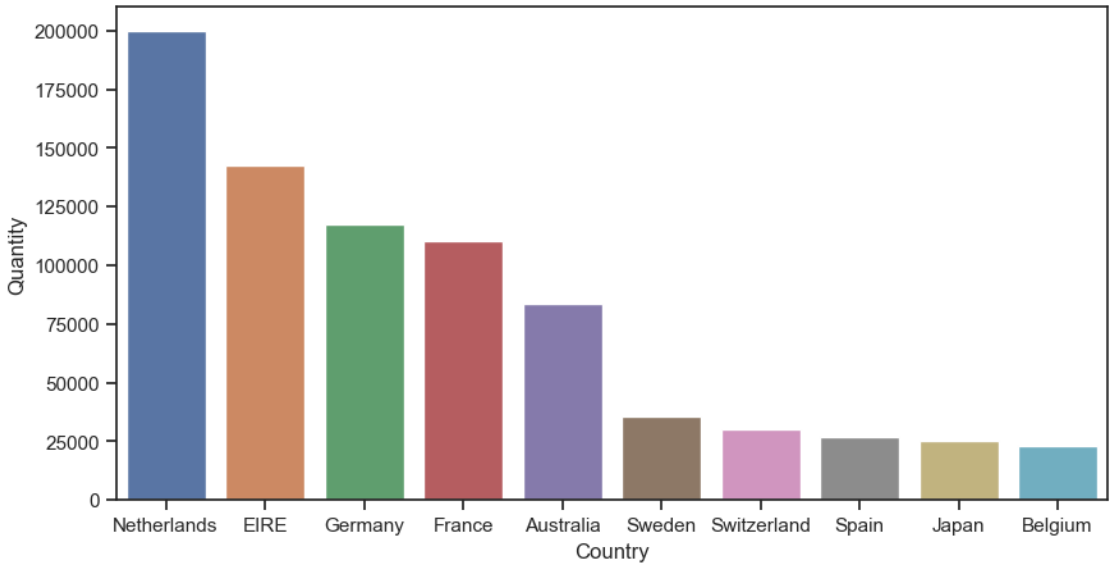
	InvoiceDate	UnitPrice	CustomerID	Country
0	12/1/10 8:26	2.55	17850.0	United Kingdom
1	12/1/10 8:26	3.39	17850.0	United Kingdom
2	12/1/10 8:26	2.75	17850.0	United Kingdom
3	12/1/10 8:26	3.39	17850.0	United Kingdom
4	12/1/10 8:26	3.39	17850.0	United Kingdom
…	…	…	…	…

```
541904   12/9/11 12:50       0.85     12680.0          France
541905   12/9/11 12:50       2.10     12680.0          France
541906   12/9/11 12:50       4.15     12680.0          France
541907   12/9/11 12:50       4.15     12680.0          France
541908   12/9/11 12:50       4.95     12680.0          France

[541909 rows x 8 columns]
```

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

[54]:
```
country=online_rt.groupby('Country')[['Quantity']].sum().
  ↪sort_values('Quantity',ascending=False).iloc[1:11]
plt.figure(figsize=(10,5))
sns.barplot(x=country.index,y='Quantity',data=country)
```

[54]:
```
<AxesSubplot:xlabel='Country', ylabel='Quantity'>
```



1.17.3 79. Exclude negative Quatity entries

[55]:
```
online_rt1 = online_rt[(online_rt.Quantity > 0)].reset_index(drop=True)
```

[56]:
```
online_rt1
```

[56]:
```
       InvoiceNo StockCode                          Description  Quantity  \
0         536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1         536365     71053                  WHITE METAL LANTERN         6
```

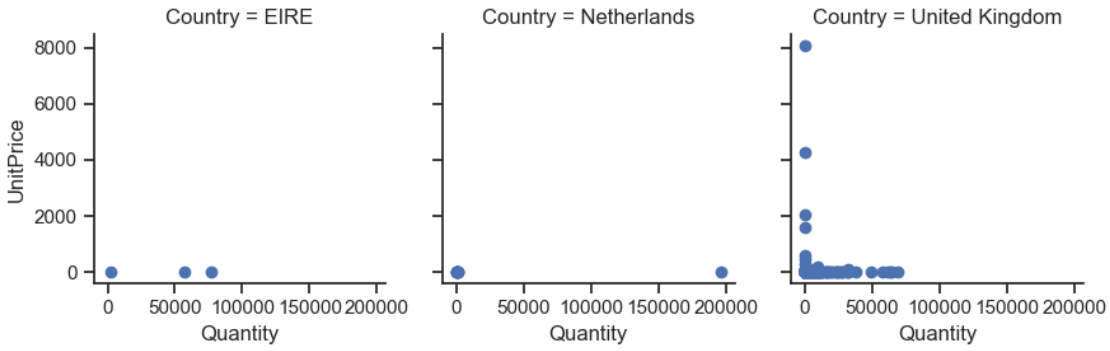
```
2         536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3         536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4         536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
…           …       …                                     …       …
531280    581587     22613          PACK OF 20 SPACEBOY NAPKINS        12
531281    581587     22899         CHILDREN'S APRON DOLLY GIRL          6
531282    581587     23254        CHILDRENS CUTLERY DOLLY GIRL          4
531283    581587     23255      CHILDRENS CUTLERY CIRCUS PARADE         4
531284    581587     22138        BAKING SET 9 PIECE RETROSPOT          3

          InvoiceDate  UnitPrice  CustomerID         Country
0        12/1/10 8:26       2.55     17850.0  United Kingdom
1        12/1/10 8:26       3.39     17850.0  United Kingdom
2        12/1/10 8:26       2.75     17850.0  United Kingdom
3        12/1/10 8:26       3.39     17850.0  United Kingdom
4        12/1/10 8:26       3.39     17850.0  United Kingdom
…                 …          …           …               …
531280  12/9/11 12:50       0.85     12680.0          France
531281  12/9/11 12:50       2.10     12680.0          France
531282  12/9/11 12:50       4.15     12680.0          France
531283  12/9/11 12:50       4.15     12680.0          France
531284  12/9/11 12:50       4.95     12680.0          France

[531285 rows x 8 columns]
```

1.17.4 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

[57]:
```
online_rt['Revenue']=online_rt['Quantity']*online_rt['UnitPrice']
Top3=online_rt.groupby('Country')[['Revenue']].sum().
 ↪sort_values(by='Revenue',ascending=False).head(3).index.to_list()
b=online_rt.query(f'Country== {Top3} ').groupby(['Country','CustomerID']).
 ↪agg({'Quantity':'sum','UnitPrice':'mean'}).reset_index()
c=sns.FacetGrid(b,col='Country')
c.map(plt.scatter,"Quantity","UnitPrice")
```

[57]:
```
<seaborn.axisgrid.FacetGrid at 0x1b92eb776d0>
```



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

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

[58]:
```
company_data=pd.read_csv(r"C:\Users\Bhaskar thakur\Documents\Analytics␣
↪assignments\Basic Data Manipulation - Visualization Exercise_2\Exercise Data␣
↪Files\FMCG_Company_Data_2019.csv")
company_data
```

[58]:

	Month	FaceCream	FaceWash	ToothPaste	Soap	Shampo	Moisturizer	\
0	Jan-19	2500	1500	5200	9200	1200	1500	
1	Feb-19	2630	1200	5100	6100	2100	1200	
2	Mar-19	2140	1340	4550	9550	3550	1340	
3	Apr-19	3400	1130	5870	8870	1870	1130	
4	May-19	3600	1740	4560	7760	1560	1740	
5	Jun-19	2760	1555	4890	7490	1890	1555	
6	Jul-19	2980	1120	4780	8980	1780	1120	
7	Aug-19	3700	1400	5860	9960	2860	1400	
8	Sep-19	3540	1780	6100	8100	2100	1780	
9	Oct-19	1990	1890	8300	10300	2300	1890	
10	Nov-19	2340	2100	7300	13300	2400	2100	
11	Dec-19	2900	1760	7400	14400	1800	1760	

	Total_Units	Total_Revenue	Total_Profit
0	21100	3584890	211000
1	18330	2864979	183300
2	22470	4058082	224700
3	22270	2890646	222700
4	20960	2997280	209600
5	20140	2857866	201400
6	29550	5735655	295500
7	36140	5196932	361400

8	23400	3060720	234000
9	26670	4661916	266700
10	41280	6794688	412800
11	30020	3770512	300200

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

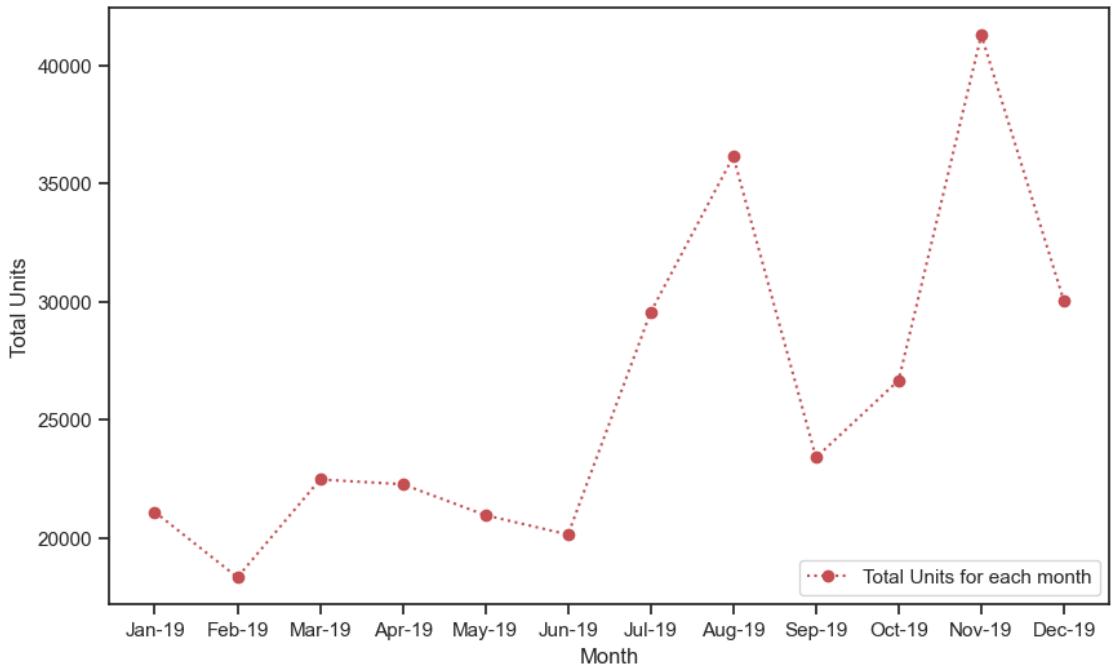
- X label name = Month
- Y label name = Total Revenue

[ ]:
```
plt.figure(figsize=(10,6))
plt.plot(company_data.Month,company_data.Total_Revenue,marker='o')
plt.ticklabel_format(style='plain',axis='y')
plt.xlabel('Month')
plt.ylabel('Total Revenue')
plt.show()
```

1.18.3 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.

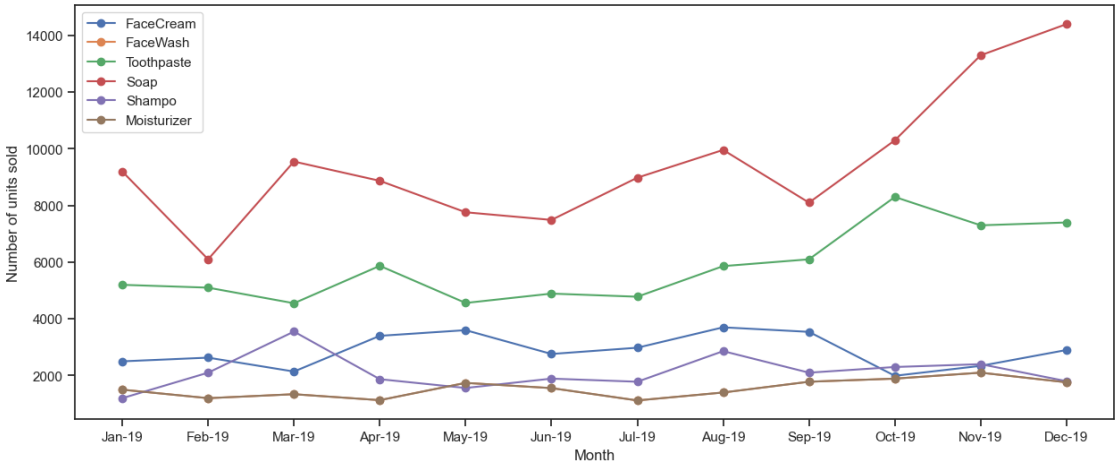
[65]:
```
plt.figure(figsize=(10,6))
plt.plot(company_data.Month,company_data.Total_Units,label='Total Units for␣
 ↪each month',linestyle='dotted',color='r',marker='o')
plt.xlabel('Month')
plt.ylabel('Total Units')
plt.legend(loc='lower right')
plt.show()
```



1.18.4 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).

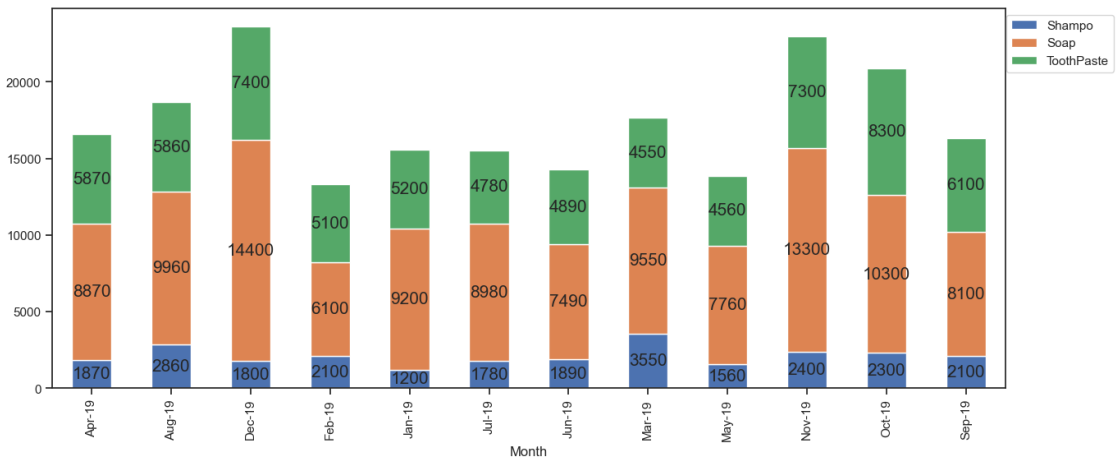
[64]:
```
plt.figure(figsize=(15,6))
plt.plot(company_data.Month,company_data.FaceCream,label='FaceCream',marker='o')
plt.plot(company_data.Month,company_data.FaceWash,label='FaceWash',marker='o')
plt.plot(company_data.Month,company_data.
 ↪ToothPaste,label='Toothpaste',marker='o')
plt.plot(company_data.Month,company_data.Soap,label='Soap',marker='o')
plt.plot(company_data.Month,company_data.Shampo,label='Shampo',marker='o')
plt.plot(company_data.Month,company_data.
 ↪Moisturizer,label='Moisturizer',marker='o')
plt.xlabel('Month')
plt.ylabel('Number of units sold')
plt.legend(loc='upper left')
plt.show()
```



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

[62]:
```
a=pd.
  ↪pivot_table(data=company_data,index='Month',values=['Soap','Shampo','ToothPaste']).
  ↪plot(kind='bar',stacked=True,figsize=(15,6))
for i in a.containers:
    a.bar_label(i,size=15,label_type='center')
plt.legend(bbox_to_anchor=(1.12,1))
```

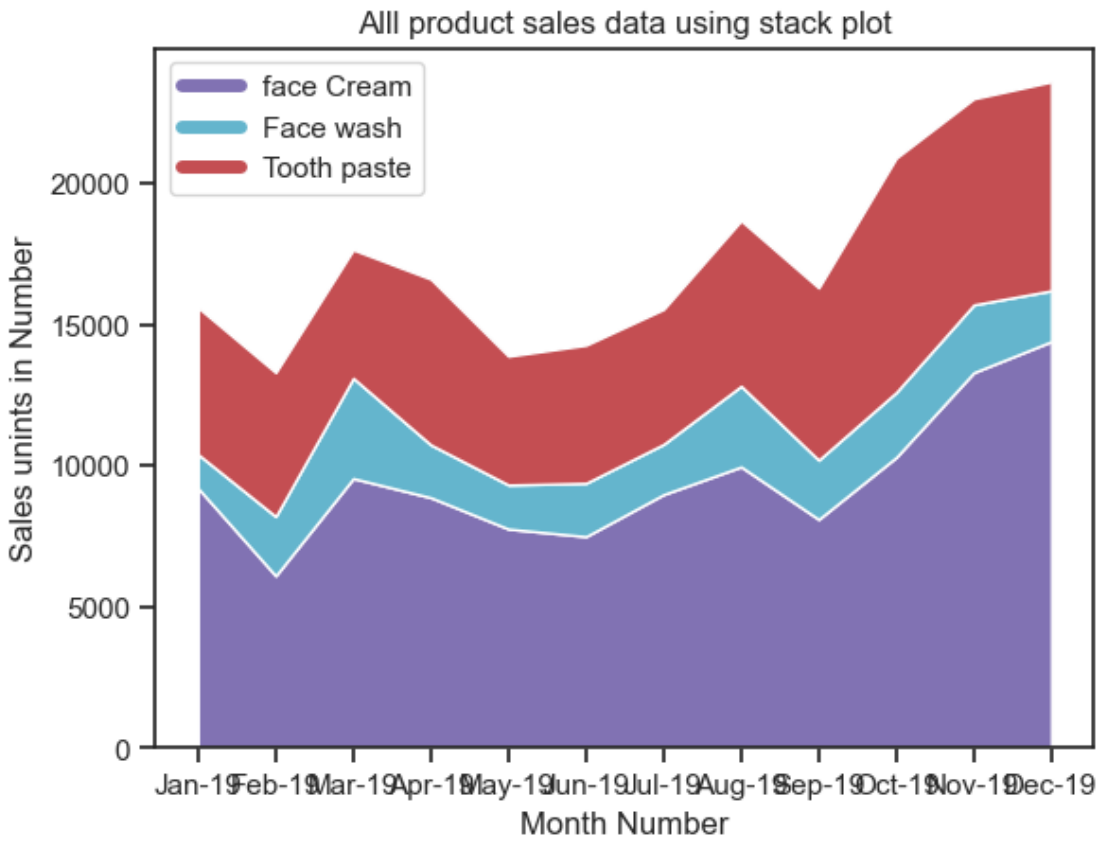
[62]: <matplotlib.legend.Legend at 0x1b92ee0afa0>



1.18.6 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.

[94]:
```
## need to check
monthList  = company_data ['Month'].tolist()

faceCremSalesData   = company_data ['Soap'].tolist()
faceWashSalesData   = company_data ['Shampo'].tolist()
toothPasteSalesData = company_data ['ToothPaste'].tolist()

plt.plot([],[],color='m', label='face Cream', linewidth=5)
plt.plot([],[],color='c', label='Face wash', linewidth=5)
plt.plot([],[],color='r', label='Tooth paste', linewidth=5)


plt.stackplot(monthList, faceCremSalesData, faceWashSalesData,␣
 ↪toothPasteSalesData,

              colors=['m','c','r'])

plt.xlabel('Month Number')
plt.ylabel('Sales unints in Number')
plt.title('Alll product sales data using stack plot')
plt.legend(loc='upper left')
plt.show()
```

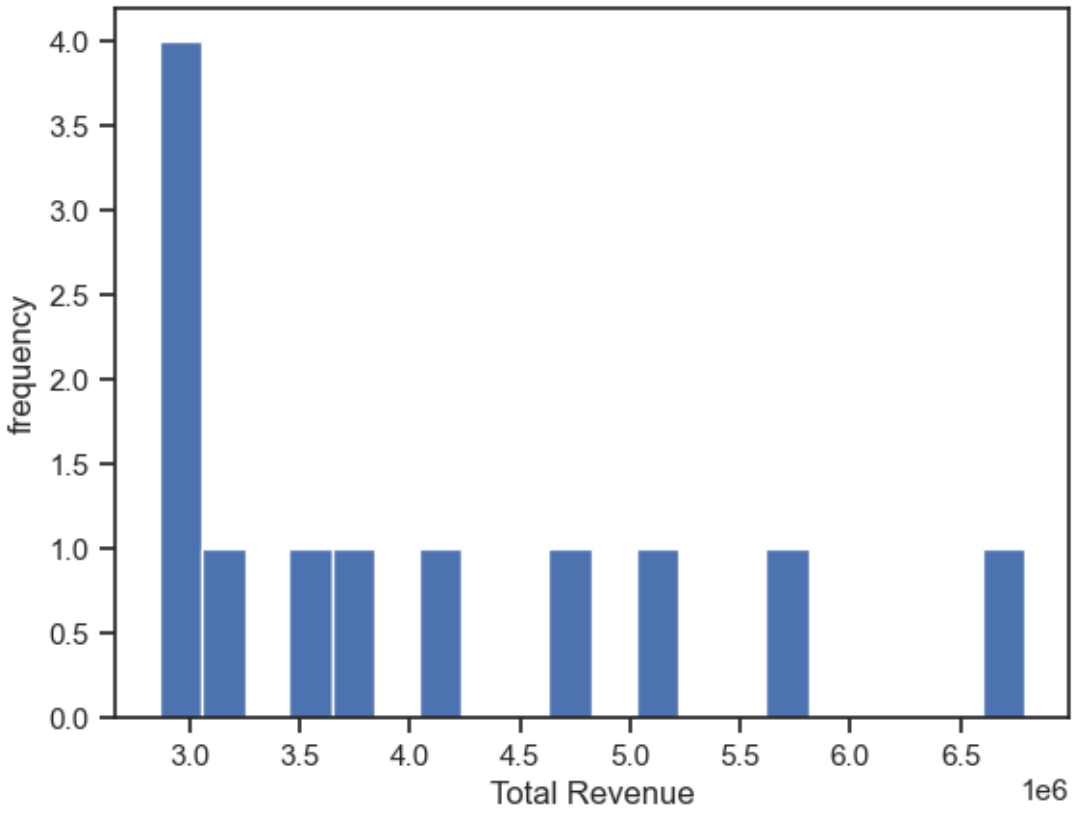


[ ]:
```
company_data
```

1.18.7 87. Create Histogram for Total Revenue

[61]:
```
plt.hist(company_data.Total_Revenue,bins=20)
plt.xlabel('Total Revenue')
plt.ylabel('frequency')
plt.show
```

[61]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[79]:
```
company_data.head(2)
```

[79]:

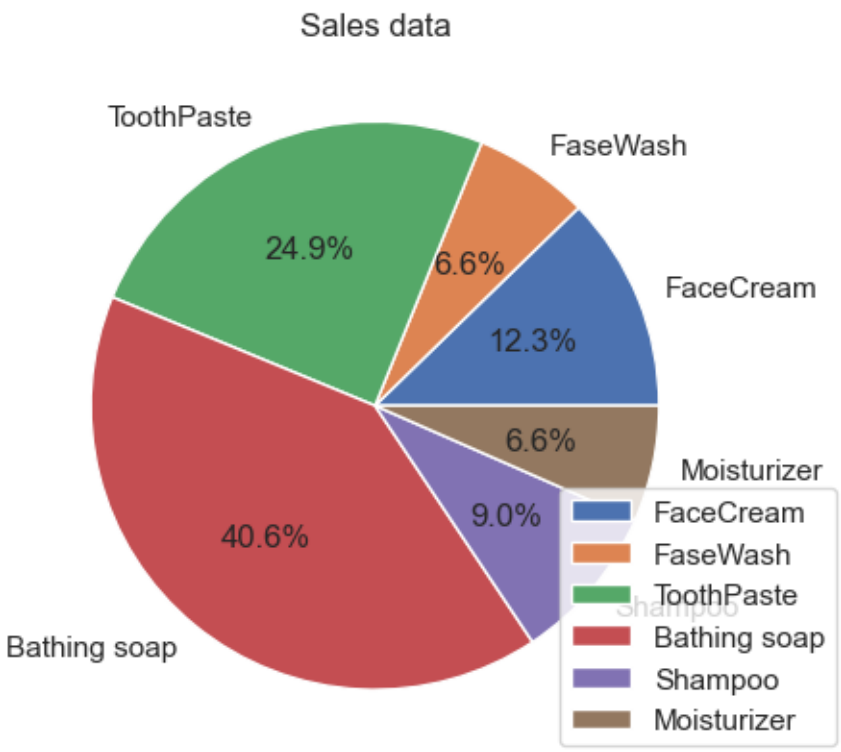
	Month	FaceCream	FaceWash	ToothPaste	Soap	Shampo	Moisturizer	\
0	Jan-19	2500	1500	5200	9200	1200	1500	
1	Feb-19	2630	1200	5100	6100	2100	1200	

	Total_Units	Total_Revenue	Total_Profit

0	21100	3584890	211000
1	18330	2864979	183300

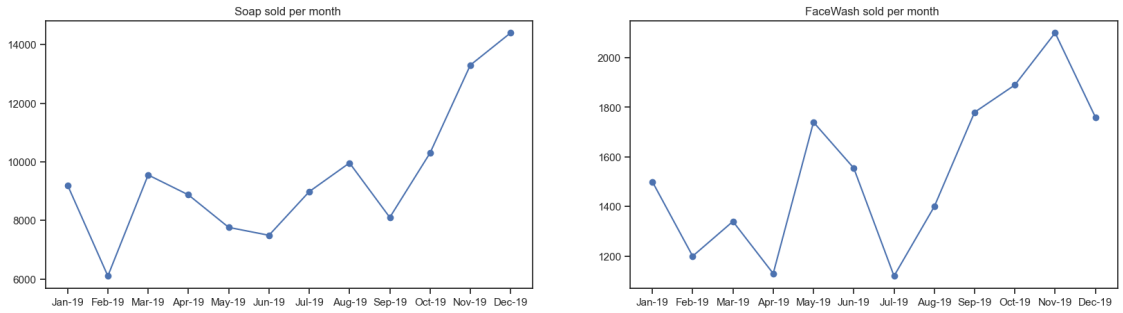
1.18.8 88. Calculate total sales data (quantity) for 2019 for each product and show it using a Pie chart. Understand percentage contribution from each product

[81]:
```
monthList  = company_data ['Month'].tolist()

labels = ['FaceCream', 'FaseWash', 'ToothPaste', 'Bathing soap', 'Shampoo',␣
 ↪'Moisturizer']
salesData   = [company_data ['FaceCream'].sum(), company_data ['FaceWash'].
 ↪sum(), company_data ['ToothPaste'].sum(),
        company_data ['Soap'].sum(), company_data ['Shampo'].sum(),␣
 ↪company_data ['Moisturizer'].sum()]
plt.axis("equal")
plt.pie(salesData, labels=labels, autopct='%1.1f%%')
plt.legend(loc='lower right')
plt.title('Sales data')
plt.show()
```



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

[60]:
```
f, axes = plt.subplots(1,2,figsize=(20,5))
axes[0].plot(company_data.Month,company_data.Soap,marker='o')
axes[0].set_title('Soap sold per month ')
axes[1].plot(company_data.Month,company_data.FaceWash,marker='o')
axes[1].set_title('FaceWash sold per month ')
plt.show()
```



1.18.10 90. Create Box Plot for Total Profit variable

[71]:
```
company_data.head(2)
```

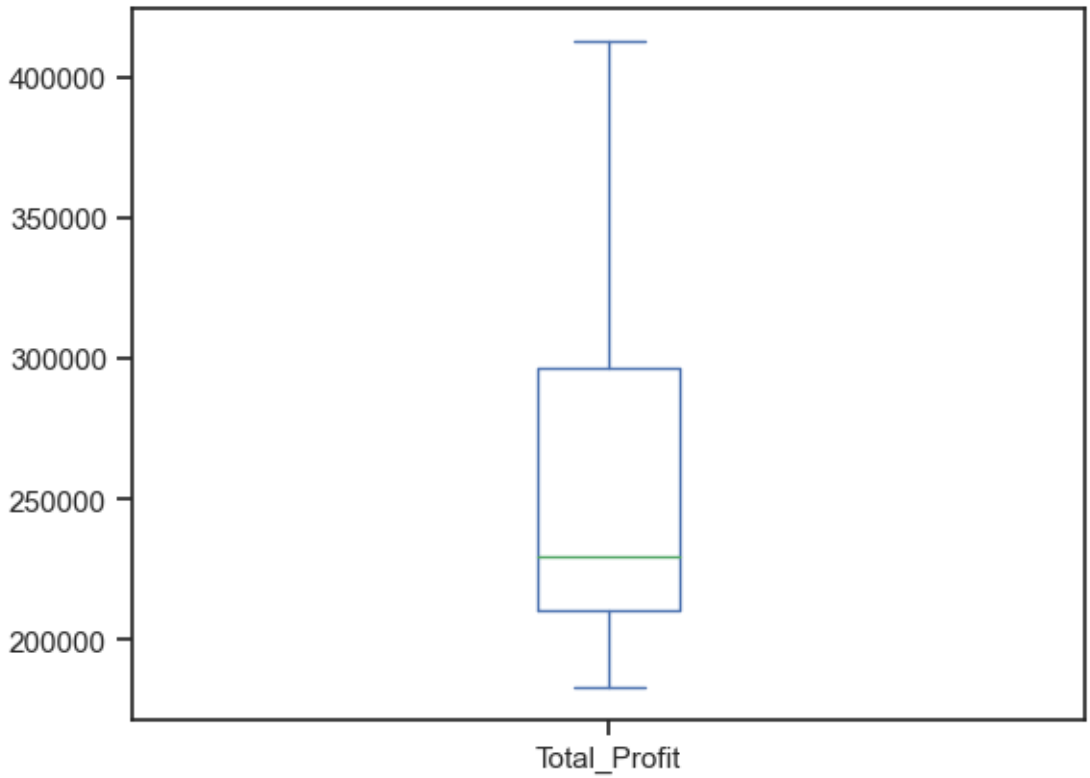
[71]:

	Month	FaceCream	FaceWash	ToothPaste	Soap	Shampo	Moisturizer	\
0	Jan-19	2500	1500	5200	9200	1200	1500	
1	Feb-19	2630	1200	5100	6100	2100	1200	

	Total_Units	Total_Revenue	Total_Profit
0	21100	3584890	211000
1	18330	2864979	183300

[59]:
```
company_data.Total_Profit.plot(kind='box')
```

[59]: <AxesSubplot:>


400000
350000
300000
250000
200000
Total_Profit

[ ]: