question-1

November 15, 2023

1 ASSIGNMENT 1:

MY GITHUB LINK: click me

1.1 Question 1:

NAME: RISHAV KUMAR

ROLL NO. 2301560042

2 EX 1 : Python Pandas Exercise

[ ]:
```
import pandas as pd
path =r'C:\Users\risha\Documents\KRMU\AIML_assigment\datasets\Automobile_data.
↪csv'
df= pd.read_csv(path)
```

Exercise 1: From the given dataset print the first and last five rows

[ ]:
```
df.head()
```

```
   index      company   body-style  wheel-base  length engine-type  \
0      0  alfa-romero  convertible        88.6   168.8        dohc
1      1  alfa-romero  convertible        88.6   168.8        dohc
2      2  alfa-romero    hatchback        94.5   171.2        ohcv
3      3         audi        sedan        99.8   176.6         ohc
4      4         audi        sedan        99.4   176.6         ohc

  num-of-cylinders  horsepower  average-mileage    price
0             four         111               21  13495.0
1             four         111               21  16500.0
2              six         154               19  16500.0
3             four         102               24  13950.0
4             five         115               18  17450.0
```

[ ]:
```
df.tail()
```

```
    index      company body-style  wheel-base  length engine-type  \
56     81  volkswagen      sedan        97.3   171.7         ohc
57     82  volkswagen      sedan        97.3   171.7         ohc
58     86  volkswagen      sedan        97.3   171.7         ohc
59     87       volvo      sedan       104.3   188.8         ohc
60     88       volvo      wagon       104.3   188.8         ohc

   num-of-cylinders  horsepower  average-mileage    price
56             four          85               27   7975.0
57             four          52               37   7995.0
58             four         100               26   9995.0
59             four         114               23  12940.0
60             four         114               23  13415.0
```

[ ]:
```
path=r'C:\Users\risha\Documents\KRMU\AIML_assigment\datasets\Automobile_data.
↪csv'
ds= pd.read_csv(path, na_values=
        {'price': ['?','n.a'],
        'stroke': ['?','n.a'],
        'horsepower': ['?','n.a'],
        'peak-rpm': ['?','n.a'],
        'average-milage': ['?','n.a']})

ds.to_csv(r'C:
↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\Automobile_data.csv')
```

Exercise 3: Find the most expensive car company name

[ ]:
```
dx= df[['company', 'price']][df.price==df['price'].max()]
dx
```

```
          company    price
35  mercedes-benz  45400.0
```

[ ]:
```
toyota_df= df.groupby('company')
data= toyota_df.get_group('toyota')
data
```

```
    index company body-style  wheel-base  length engine-type num-of-cylinders  \
48     66  toyota  hatchback        95.7   158.7         ohc             four
49     67  toyota  hatchback        95.7   158.7         ohc             four
50     68  toyota  hatchback        95.7   158.7         ohc             four
51     69  toyota      wagon        95.7   169.7         ohc             four
52     70  toyota      wagon        95.7   169.7         ohc             four
53     71  toyota      wagon        95.7   169.7         ohc             four
54     79  toyota      wagon       104.5   187.8        dohc              six

    horsepower  average-mileage    price
```

```
48          62               35   5348.0
49          62               31   6338.0
50          62               31   6488.0
51          62               31   6918.0
52          62               27   7898.0
53          62               27   8778.0
54         156               19  15750.0
```

Exercise 5: Count total cars per company

[ ]:
```
data= df['company'].value_counts()
data
```

```
company
toyota           7
bmw              6
mazda            5
nissan           5
audi             4
mercedes-benz    4
mitsubishi       4
volkswagen       4
alfa-romero      3
chevrolet        3
honda            3
isuzu            3
jaguar           3
porsche          3
dodge            2
volvo            2
Name: count, dtype: int64
```

Exercise 6: Find each company s Higesht price car

[ ]:
```
categ= df.groupby('company')
data= categ['price'].max()
data
```

```
company
alfa-romero      16500.0
audi             18920.0
bmw              41315.0
chevrolet         6575.0
dodge             6377.0
honda            12945.0
isuzu             6785.0
jaguar           36000.0
mazda            18344.0
mercedes-benz    45400.0
```

```
mitsubishi         8189.0
nissan            13499.0
porsche           37028.0
toyota            15750.0
volkswagen         9995.0
volvo             13415.0
Name: price, dtype: float64
```

Exercise 7: Find the average mileage of each car making company

```
[ ]: cat_comp= df.groupby('company')
     data= cat_comp['average-mileage'].mean()
     data
```

```
company
alfa-romero      20.333333
audi             20.000000
bmw              19.000000
chevrolet        41.000000
dodge            31.000000
honda            26.333333
isuzu            33.333333
jaguar           14.333333
mazda            28.000000
mercedes-benz    18.000000
mitsubishi       29.500000
nissan           31.400000
porsche          17.000000
toyota           28.714286
volkswagen       31.750000
volvo            23.000000
Name: average-mileage, dtype: float64
```

Exercise 8: Sort all cars by Price column

```
[ ]: sort= df.sort_values(by= ['price'], ascending= False).reset_index()
     sort
```

```
    level_0  index        company   body-style  wheel-base  length  \
0        35     47  mercedes-benz      hardtop       112.0   199.2
1        11     14            bmw        sedan       103.5   193.8
2        34     46  mercedes-benz        sedan       120.9   208.1
3        46     62        porsche  convertible        89.5   168.9
4        12     15            bmw        sedan       110.0   197.0
..        …    …              …          …         …       …
56       27     36          mazda    hatchback        93.1   159.1
57       13     16      chevrolet    hatchback        88.4   141.1
58       22     31          isuzu        sedan        94.5   155.9
59       23     32          isuzu        sedan        94.5   155.9
```

```
60      47     63        porsche    hatchback        98.4   175.7

   engine-type num-of-cylinders  horsepower  average-mileage    price
0         ohcv            eight         184               14  45400.0
1          ohc              six         182               16  41315.0
2         ohcv            eight         184               14  40960.0
3         ohcf              six         207               17  37028.0
4          ohc              six         182               15  36880.0
..         …               …          …                …       …
56         ohc             four          68               30   5195.0
57           l            three          48               47   5151.0
58         ohc             four          70               38      NaN
59         ohc             four          70               38      NaN
60       dohcv            eight         288               17      NaN

[61 rows x 11 columns]
```

Exercise 9: Concatenate two data frames using the following conditions

[ ]:
```
germoon= {'comapny':['ford', 'mecidies','BMW', 'Audi'],'price':[300, 696,899,␣
  ↪454]}
japaan= {'comapny':['toyoota', 'onii','nii-san', 'Ohayoo'],'price':[335,␣
  ↪654,459, 934]}

df1= pd.DataFrame.from_dict(germoon)
df2= pd.DataFrame.from_dict(japaan)

main_df= pd.concat([df1, df2], keys=['Germany', 'Japan'])

main_df
```

```
            comapny  price
Germany 0      ford    300
        1  mecidies    696
        2       BMW    899
        3      Audi    454
Japan   0   toyoota    335
        1      onii    654
        2   nii-san    459
        3    Ohayoo    934
```

Exercise 10: Merge two data frames using the following condition

[ ]:
```
car_price= {
    "Company": ['Toyota','Honda', 'BMW','Audi'],
    'Price'  : [1234,7653, 9874,4982]
}
car_horsepow= {
    "Company": ['Toyota','Honda', 'BMW','Audi'],
```

```
    "horsepower":[2,4,5,8]
}

df1= pd.DataFrame.from_dict(car_price)
df2= pd.DataFrame.from_dict(car_horsepow)

data= pd.merge(df1, df2, on='Company')

data
```

```
  Company  Price  horsepower
0  Toyota   1234           2
1   Honda   7653           4
2     BMW   9874           5
3    Audi   4982           8
```

[ ]:

```

```

3 EX 2 : Getting and Knowing your Data

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

3.0.1 Step 1. Import the necessary libraries

[ ]:

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

3.0.2 Step 2. Import the dataset from this address.

3.0.3 Step 3. Assign it to a variable called chipo.

[ ]:

```
chipo=pd.read_csv(r'C:
↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\chipotle.csv')
```

3.0.4 Step 4. See the first 10 entries

[ ]:

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

```
   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
5         3         1                           Chicken Bowl
6         3         1                          Side of Chips
```

```
7         4         1                          Steak Burrito
8         4         1                       Steak Soft Tacos
9         5         1                          Steak Burrito

                                  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
5  [Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou…    $10.98
6                                                NaN     $1.69
7  [Tomatillo Red Chili Salsa, [Fajita Vegetables…    $11.75
8  [Tomatillo Green Chili Salsa, [Pinto Beans, Ch…     $9.25
9  [Fresh Tomato Salsa, [Rice, Black Beans, Pinto…     $9.25
```

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

[ ]:
```
# Solution 1
chipo.shape[0]
```

```
4622
```

[ ]:
```
# Solution 2
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   object
dtypes: int64(2), object(3)
memory usage: 180.7+ KB
```

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

[ ]:
```
chipo.shape[1]
```

```
5
```

[ ]:
```
chipo.columns
```

```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
```

```
       'item_price'],
      dtype='object')
```

3.0.7 Step 8. How is the dataset indexed?

[ ]:
```
chipo.index
```

```
RangeIndex(start=0, stop=4622, step=1)
```

3.0.8 Step 9. Which was the most-ordered item?

[ ]:
```
c= chipo.groupby('item_name')
c= c.sum()
c= c.sort_values(['quantity'], ascending=False)
c.head(1)
```

```
              order_id  quantity  \
item_name
Chicken Bowl    713926       761

                                               choice_description  \
item_name
Chicken Bowl  [Tomatillo-Red Chili Salsa (Hot), [Black Beans…

                                                       item_price
item_name
Chicken Bowl  $16.98 $10.98 $11.25 $8.75 $8.49 $11.25 $8.75 …
```

3.0.9 Step 10. For the most-ordered item, how many items were ordered?

[ ]:
```
c= chipo.groupby('item_name')
c= c.sum()
c= c.sort_values(['quantity'], ascending=False)
c.head(1)
```

```
              order_id  quantity  \
item_name
Chicken Bowl    713926       761

                                               choice_description  \
item_name
Chicken Bowl  [Tomatillo-Red Chili Salsa (Hot), [Black Beans…

                                                       item_price
item_name
Chicken Bowl  $16.98 $10.98 $11.25 $8.75 $8.49 $11.25 $8.75 …
```

3.0.10 Step 11. What was the most ordered item in the choice_description column?

[ ]:
```
ch= chipo.groupby('choice_description').sum()
ch= ch.sort_values(['quantity'], ascending= False)
ch.head(1)
```

```
                    order_id  quantity  \
choice_description
[Diet Coke]           123455       159

                                                            item_name  \
choice_description
[Diet Coke]         Canned SodaCanned SodaCanned Soda6 Pack Soft D…

                                                           item_price
choice_description
[Diet Coke]         $2.18 $1.09 $1.09 $6.49 $2.18 $1.25 $1.09 $6.4…
```

3.0.11 Step 12. How many items were orderd in total?

[ ]:
```
tod= chipo.quantity.sum()
tod
```

```
4972
```

3.0.12 Step 13. Turn the item price into a float

Step 13.a. Check the item price type

[ ]:
```
chipo.item_price.dtype
```

```
dtype('O')
```

Step 13.b. Create a lambda function and change the type of item price

[ ]:
```
dol= lambda x:float(x[1:-1])
chipo.item_price=chipo.item_price.apply(dol)
```

Step 13.c. Check the item price type

[ ]:
```
chipo.item_price.dtype
```

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

3.0.13 Step 14. How much was the revenue for the period in the dataset?

[ ]:
```
rev= (chipo['quantity']*chipo['item_price']).sum()
print('Revenue was : '+ str(np.round(rev, 2)))
```

```
Revenue was : 39237.02
```

3.0.14 Step 15. How many orders were made in the period?

```
[ ]: order = chipo.order_id.value_counts().count()
     order
```

```
1834
```

3.0.15 Step 16. What is the average revenue amount per order?

```
[ ]: # Solution 1
     chipo['revenue']= chipo['quantity']*chipo['item_price']
     ord= chipo.groupby(by=['order_id']).sum()
     ord['revenue'].mean()
```

```
21.39423118865867
```

3.0.16 Step 17. How many different items are sold?

```
[ ]: countin= chipo.item_name.value_counts().count()
     countin
```

```
50
```

```
[ ]:
```

4 EX 3 : Filtering and Sorting Data

This time we are going to pull data directly from the internet.

4.0.1 Step 1. Import the necessary libraries

```
[ ]: import pandas as pd
```

4.0.2 Step 2. Import the dataset from this address.

4.0.3 Step 3. Assign it to a variable called euro12.

```
[ ]: euro12= pd.read_csv(r"C:
     ↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\Euro_2012_stats_TEAM.
     ↪csv")
```

4.0.4 Step 4. Select only the Goal column.

```
[ ]: go= euro12['Goals']
     go
```

```
0      4
1      4
2      4
3      5
4      3
5     10
6      5
7      6
8      2
9      2
10     6
11     1
12     5
13    12
14     5
15     2
Name: Goals, dtype: int64
```

4.0.5 Step 5. How many team participated in the Euro2012?

[ ]:
```
euro12.shape[0]
```

```
16
```

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

[ ]:
```
euro12.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 35 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Team                        16 non-null     object
 1   Goals                       16 non-null     int64
 2   Shots on target             16 non-null     int64
 3   Shots off target            16 non-null     int64
 4   Shooting Accuracy           16 non-null     object
 5   % Goals-to-shots            16 non-null     object
 6   Total shots (inc. Blocked)  16 non-null     int64
 7   Hit Woodwork                16 non-null     int64
 8   Penalty goals               16 non-null     int64
 9   Penalties not scored        16 non-null     int64
 10  Headed goals                16 non-null     int64
 11  Passes                      16 non-null     int64
 12  Passes completed            16 non-null     int64
 13  Passing Accuracy            16 non-null     object
 14  Touches                     16 non-null     int64
```

```
 15   Crosses                     16 non-null     int64
 16   Dribbles                    16 non-null     int64
 17   Corners Taken               16 non-null     int64
 18   Tackles                     16 non-null     int64
 19   Clearances                  16 non-null     int64
 20   Interceptions               16 non-null     int64
 21   Clearances off line         15 non-null     float64
 22   Clean Sheets                16 non-null     int64
 23   Blocks                      16 non-null     int64
 24   Goals conceded              16 non-null     int64
 25   Saves made                  16 non-null     int64
 26   Saves-to-shots ratio        16 non-null     object
 27   Fouls Won                   16 non-null     int64
 28   Fouls Conceded              16 non-null     int64
 29   Offsides                    16 non-null     int64
 30   Yellow Cards                16 non-null     int64
 31   Red Cards                   16 non-null     int64
 32   Subs on                     16 non-null     int64
 33   Subs off                    16 non-null     int64
 34   Players Used                16 non-null     int64
dtypes: float64(1), int64(29), object(5)
memory usage: 4.5+ KB
```

4.0.7 Step 7. View only the columns Team, Yellow Cards and Red Cards and assign them to a dataframe called discipline

[ ]:
```
dis= euro12[['Team', 'Yellow Cards', 'Red Cards']]
dis
```

```
                   Team  Yellow Cards  Red Cards
0               Croatia             9          0
1        Czech Republic             7          0
2               Denmark             4          0
3               England             5          0
4                France             6          0
5               Germany             4          0
6                Greece             9          1
7                 Italy            16          0
8           Netherlands             5          0
9                Poland             7          1
10             Portugal            12          0
11  Republic of Ireland             6          1
12               Russia             6          0
13                Spain            11          0
14               Sweden             7          0
15              Ukraine             5          0
```

4.0.8 Step 8. Sort the teams by Red Cards, then to Yellow Cards

[ ]:
```
dis.sort_values(['Red Cards', 'Yellow Cards'], ascending=False)
dis
```

```
                  Team  Yellow Cards  Red Cards
0              Croatia             9          0
1       Czech Republic             7          0
2              Denmark             4          0
3              England             5          0
4               France             6          0
5              Germany             4          0
6               Greece             9          1
7                Italy            16          0
8          Netherlands             5          0
9               Poland             7          1
10            Portugal            12          0
11 Republic of Ireland             6          1
12              Russia             6          0
13               Spain            11          0
14              Sweden             7          0
15             Ukraine             5          0
```

4.0.9 Step 9. Calculate the mean Yellow Cards given per Team

[ ]:
```
round(dis['Yellow Cards'].mean())
```

```
7
```

4.0.10 Step 10. Filter teams that scored more than 6 goals

[ ]:
```
euro12[euro12.Goals > 6]
```

```
       Team  Goals  Shots on target  Shots off target Shooting Accuracy  \
5   Germany     10               32                32             47.8%
13    Spain     12               42                33             55.9%

  % Goals-to-shots  Total shots (inc. Blocked)  Hit Woodwork  Penalty goals  \
5            15.6%                          80             2              1
13           16.0%                         100             0              1

   Penalties not scored  …  Saves made  Saves-to-shots ratio  Fouls Won  \
5                     0  …          10                 62.6%         63
13                    0  …          15                 93.8%        102

  Fouls Conceded  Offsides  Yellow Cards  Red Cards  Subs on  Subs off  \
5             49        12             4          0       15        15
13            83        19            11          0       17        17
```

```
    Players Used
5             17
13            18

[2 rows x 35 columns]
```

4.0.11 Step 11. Select the teams that start with G

```
[ ]: euro12[euro12.Team.str.startswith('G')]
```

```
      Team  Goals  Shots on target  Shots off target Shooting Accuracy  \
5  Germany     10               32                32             47.8%
6   Greece      5                8                18             30.7%

  % Goals-to-shots  Total shots (inc. Blocked)  Hit Woodwork  Penalty goals  \
5            15.6%                          80             2              1
6            19.2%                          32             1              1

   Penalties not scored  …  Saves made  Saves-to-shots ratio  Fouls Won  \
5                     0  …          10                 62.6%         63
6                     1  …          13                 65.1%         67

  Fouls Conceded  Offsides  Yellow Cards  Red Cards  Subs on  Subs off  \
5             49        12             4          0       15        15
6             48        12             9          1       12        12

   Players Used
5            17
6            20

[2 rows x 35 columns]
```

4.0.12 Step 12. Select the first 7 columns

```
[ ]: euro12.iloc[: , 0:7]
```

```
                   Team  Goals  Shots on target  Shots off target  \
0               Croatia      4               13                12
1        Czech Republic      4               13                18
2               Denmark      4               10                10
3               England      5               11                18
4                France      3               22                24
5               Germany     10               32                32
6                Greece      5                8                18
7                 Italy      6               34                45
8           Netherlands      2               12                36
9                Poland      2               15                23
```

```
10               Portugal      6                22                42
11    Republic of Ireland      1                 7                12
12                 Russia      5                 9                31
13                  Spain     12                42                33
14                 Sweden      5                17                19
15                Ukraine      2                 7                26

   Shooting Accuracy % Goals-to-shots  Total shots (inc. Blocked)
0                51.9%          16.0%                          32
1                41.9%          12.9%                          39
2                50.0%          20.0%                          27
3                50.0%          17.2%                          40
4                37.9%           6.5%                          65
5                47.8%          15.6%                          80
6                30.7%          19.2%                          32
7                43.0%           7.5%                         110
8                25.0%           4.1%                          60
9                39.4%           5.2%                          48
10               34.3%           9.3%                          82
11               36.8%           5.2%                          28
12               22.5%          12.5%                          59
13               55.9%          16.0%                         100
14               47.2%          13.8%                          39
15               21.2%           6.0%                          38
```

4.0.13 Step 13. Select all columns except the last 3.

[ ]:
```
euro12.iloc[: , :-3]
```

```
                   Team  Goals  Shots on target  Shots off target  \
0               Croatia      4               13                12
1        Czech Republic      4               13                18
2               Denmark      4               10                10
3               England      5               11                18
4                France      3               22                24
5               Germany     10               32                32
6                Greece      5                8                18
7                 Italy      6               34                45
8           Netherlands      2               12                36
9                Poland      2               15                23
10             Portugal      6               22                42
11  Republic of Ireland      1                7                12
12               Russia      5                9                31
13                Spain     12               42                33
14               Sweden      5               17                19
15              Ukraine      2                7                26

   Shooting Accuracy % Goals-to-shots  Total shots (inc. Blocked)  \
```

0	51.9%	16.0%	32
1	41.9%	12.9%	39
2	50.0%	20.0%	27
3	50.0%	17.2%	40
4	37.9%	6.5%	65
5	47.8%	15.6%	80
6	30.7%	19.2%	32
7	43.0%	7.5%	110
8	25.0%	4.1%	60
9	39.4%	5.2%	48
10	34.3%	9.3%	82
11	36.8%	5.2%	28
12	22.5%	12.5%	59
13	55.9%	16.0%	100
14	47.2%	13.8%	39
15	21.2%	6.0%	38

	Hit Woodwork	Penalty goals	Penalties not scored	…	Clean Sheets	\
0	0	0	0	…	0	
1	0	0	0	…	1	
2	1	0	0	…	1	
3	0	0	0	…	2	
4	1	0	0	…	1	
5	2	1	0	…	1	
6	1	1	1	…	1	
7	2	0	0	…	2	
8	2	0	0	…	0	
9	0	0	0	…	0	
10	6	0	0	…	2	
11	0	0	0	…	0	
12	2	0	0	…	0	
13	0	1	0	…	5	
14	3	0	0	…	1	
15	0	0	0	…	0	

	Blocks	Goals conceded	Saves made	Saves-to-shots ratio	Fouls Won	\
0	10	3	13	81.3%	41	
1	10	6	9	60.1%	53	
2	10	5	10	66.7%	25	
3	29	3	22	88.1%	43	
4	7	5	6	54.6%	36	
5	11	6	10	62.6%	63	
6	23	7	13	65.1%	67	
7	18	7	20	74.1%	101	
8	9	5	12	70.6%	35	
9	8	3	6	66.7%	48	
10	11	4	10	71.5%	73	
11	23	9	17	65.4%	43	

```
12        8                3         10                    77.0%          34
13        8                1         15                    93.8%         102
14       12                5          8                    61.6%          35
15        4                4         13                    76.5%          48

    Fouls Conceded  Offsides  Yellow Cards  Red Cards
0               62         2             9          0
1               73         8             7          0
2               38         8             4          0
3               45         6             5          0
4               51         5             6          0
5               49        12             4          0
6               48        12             9          1
7               89        16            16          0
8               30         3             5          0
9               56         3             7          1
10              90        10            12          0
11              51        11             6          1
12              43         4             6          0
13              83        19            11          0
14              51         7             7          0
15              31         4             5          0

[16 rows x 32 columns]
```

4.0.14 Step 14. Present only the Shooting Accuracy from England, Italy and Russia

[ ]:
```
euro12.loc[euro12.Team.isin(['England', 'Italy', 'Russia']), ['Team','Shooting Accuracy']]
```

```
       Team Shooting Accuracy
3   England             50.0%
7     Italy             43.0%
12   Russia             22.5%
```

[ ]:
```

```

5 EX 4 : GroupBy

5.0.1 Introduction:

GroupBy can be summarized as Split-Apply-Combine.

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

Check out this Diagram
Step 1. Import the necessary libraries

```
[ ]: import pandas as pd
```

5.0.2 Step 2. Import the dataset from this address.

5.0.3 Step 3. Assign it to a variable called drinks.

```
[ ]: drinks= pd.read_csv(r'C:
     ↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\drinks.csv')
     dr=drinks
     dr=dr.drop(['country'], axis=1)
```

5.0.4 Step 4. Which continent drinks more beer on average?

```
[ ]: drinks.groupby('continent').beer_servings.mean()
```

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

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

```
[ ]: drinks.groupby('continent').wine_servings.describe()
```

```
           count        mean        std  min   25%    50%     75%    max
continent
AF          53.0   16.264151  38.846419  0.0   1.0    2.0   13.00  233.0
AS          44.0    9.068182  21.667034  0.0   0.0    1.0    8.00  123.0
EU          45.0  142.222222  97.421738  0.0  59.0  128.0  195.00  370.0
OC          16.0   35.625000  64.555790  0.0   1.0    8.5   23.25  212.0
SA          12.0   62.416667  88.620189  1.0   3.0   12.0   98.50  221.0
```

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

```
[ ]: dr.groupby('continent').mean()
```

```
           beer_servings  spirit_servings  wine_servings  \
continent
AF             61.471698        16.339623      16.264151
AS             37.045455        60.840909       9.068182
EU            193.777778       132.555556     142.222222
OC             89.687500        58.437500      35.625000
SA            175.083333       114.750000      62.416667
```

```
           total_litres_of_pure_alcohol
continent
AF                             3.007547
AS                             2.170455
EU                             8.617778
OC                             3.381250
SA                             6.308333
```

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

[ ]:
```
dr.groupby('continent').median()
```

```
           beer_servings  spirit_servings  wine_servings  \
continent
AF                  32.0              3.0            2.0
AS                  17.5             16.0            1.0
EU                 219.0            122.0          128.0
OC                  52.5             37.0            8.5
SA                 162.5            108.5           12.0

           total_litres_of_pure_alcohol
continent
AF                                 2.30
AS                                 1.20
EU                                10.00
OC                                 1.75
SA                                 6.85
```

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

This time output a DataFrame

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

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

[ ]:
```

```

6 EX 5 : Student Alcohol Consumption

6.0.1 Introduction:

This time you will download a dataset from the UCI.

6.0.2 Step 1. Import the necessary libraries

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

6.0.3 Step 3. Assign it to a variable called df.

6.0.4 Step 2. Import the dataset from this address.

[ ]:
```
df = pd.read_csv(r'C:
↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\student-mat.csv')
df.head()
```

```
  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]
```

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

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

```
  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

   reason guardian
0  course   mother
1  course   father
2   other   mother
3    home   mother
4    home   father
```

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

```
[ ]: cap = lambda x: x.capitalize()
```

6.0.7 Step 6. Capitalize both Mjob and Fjob

```
[ ]: st['Mjob'].apply(cap)
     st['Fjob'].apply(cap)
```

```
0       Teacher
1         Other
2         Other
3      Services
4         Other
         …
390    Services
391    Services
392       Other
393       Other
394     At_home
Name: Fjob, Length: 395, dtype: object
```

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

```
[ ]: st.tail()
```

```
    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

     reason guardian
390  course    other
391  course   mother
392  course    other
393  course   mother
394  course   father
```

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

```
[ ]: st['Mjob']= st['Mjob'].apply(cap)
     st['Fjob']= st['Fjob'].apply(cap)
     st.tail()
```

```
    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

     reason guardian
390  course    other
391  course   mother
392  course    other
393  course   mother
394  course   father
```

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

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

[ ]:
```python
st['legal_drinker'] = st['age'].apply(majority)
st.head()
```

```
  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

   reason guardian  legal_drinker
0  course   mother           True
1  course   father          False
2   other   mother          False
3    home   mother          False
4    home   father          False
```

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

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

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

```
    return x
```

[ ]:
```
st.map(times10).head(10)
```

```
  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
5     GP   M  160       U     LE3       T    40    30  Services     Other
6     GP   M  160       U     LE3       T    20    20     Other     Other
7     GP   F  170       U     GT3       A    40    40     Other   Teacher
8     GP   M  150       U     LE3       A    30    20  Services     Other
9     GP   M  150       U     GT3       T    30    40     Other     Other

       reason guardian  legal_drinker
0      course   mother           True
1      course   father          False
2       other   mother          False
3        home   mother          False
4        home   father          False
5  reputation   mother          False
6        home   mother          False
7        home   mother          False
8        home   mother          False
9        home   mother          False
```

[ ]:
```

```

7 EX 6 : MPG Cars

7.0.1 Introduction:

The following exercise utilizes data from UC Irvine Machine Learning Repository

7.0.2 Step 1. Import the necessary libraries

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

7.0.3 Step 2. Import the first dataset cars1 and cars2.

Step 3. Assign each to a variable called cars1 and cars2

[ ]:
```

```

```
cars1= pd.read_csv(r'C:
↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\cars1.csv')
cars2= pd.read_csv(r'C:
↪\Users\risha\Documents\KRMU\AIML_assigment\datasets\cars2.csv')
```

7.0.4 Step 4. Oops, it seems our first dataset has some unnamed blank columns, fix cars1

[ ]:
```
cars1 = cars1.loc[:, "mpg":"car"]
cars1.head()
```

```
    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

   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
```

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

[ ]:
```
print(cars1.shape)
print(cars2.shape)
```

```
(198, 9)
(200, 9)
```

7.0.6 Step 6. Join cars1 and cars2 into a single DataFrame called cars

[ ]:
```
cars= pd.concat([cars1, cars2])
cars
```

```
      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
0         1  chevrolet chevelle malibu
1         1          buick skylark 320
2         1         plymouth satellite
3         1              amc rebel sst
4         1                ford torino
..      …                        …
195       1            ford mustang gl
196       2                  vw pickup
197       1              dodge rampage
198       1                ford ranger
199       1                 chevy s-10

[398 rows x 9 columns]
```

7.0.7 Step 7. Oops, there is a column missing, called owners. Create a random number Series from 15,000 to 73,000.

[ ]:
```
nr_owners = np.random.randint(15000, high=73001, size=398, dtype='l')
nr_owners
```

```
array([65936, 60313, 64074, 19727, 46458, 32859, 15854, 25202, 61591,
       71275, 69046, 42269, 43071, 71470, 33756, 64702, 67920, 28883,
       42901, 37801, 32844, 31589, 25355, 37036, 62024, 40424, 42214,
       51227, 67386, 35957, 68596, 62853, 59400, 58149, 15766, 20884,
       38316, 31332, 49183, 52629, 58666, 27067, 42665, 21323, 25310,
       16915, 27009, 34352, 27182, 38928, 35607, 42473, 64360, 35536,
       63309, 39607, 17950, 21137, 24150, 18726, 38823, 42662, 65610,
       37072, 45130, 15826, 44513, 30955, 30146, 60154, 28394, 69465,
       58242, 22564, 43992, 19830, 54167, 21301, 64035, 38726, 61014,
       58021, 26688, 28023, 28208, 21168, 16239, 45498, 55714, 40814,
       57083, 48080, 23943, 40653, 32532, 68711, 60197, 59249, 24013,
       49195, 60156, 69382, 50582, 25031, 35913, 57778, 33459, 37708,
       21392, 50280, 17308, 35473, 49947, 67387, 43350, 67936, 24651,
       32968, 48698, 24003, 64259, 24320, 25793, 44880, 45540, 44127,
       55030, 63775, 36094, 35085, 32179, 31563, 44832, 42522, 68647,
       58826, 16599, 39432, 29608, 50629, 61549, 52827, 40926, 34532,
       71371, 64723, 47175, 30128, 54753, 40464, 47399, 42144, 62229,
       27922, 29076, 34164, 51387, 18319, 40510, 58262, 53211, 44960,
       20022, 49345, 49929, 53941, 66550, 66695, 19150, 71361, 45789,
       27849, 51603, 35294, 61627, 30242, 34935, 24233, 60856, 34499,
       38347, 27096, 58580, 42339, 20847, 72874, 36260, 29927, 25658,
       32956, 26488, 18581, 49463, 33759, 39963, 58050, 41653, 21919,
       51689, 35537, 34726, 55749, 64014, 27145, 65419, 57077, 65605,
```

```
       15594, 41119, 33782, 55997, 69149, 50644, 43761, 30912, 57679,
       20446, 34101, 22717, 63875, 70576, 35875, 39259, 25748, 15522,
       52394, 23511, 42116, 25723, 30822, 26037, 21048, 68679, 31401,
       38908, 66014, 17188, 41575, 52715, 53340, 37054, 43863, 63587,
       60334, 48631, 50993, 51280, 31021, 25787, 61185, 40604, 53679,
       31365, 69889, 48046, 55310, 32009, 31170, 17010, 59147, 18216,
       51884, 61649, 36653, 71332, 16496, 26149, 15042, 15894, 33214,
       46400, 27251, 63621, 60874, 30420, 52260, 70297, 50180, 40689,
       27168, 48477, 19023, 32963, 69731, 52837, 56693, 39096, 41045,
       42696, 62822, 38116, 18595, 46404, 16834, 52438, 67402, 63948,
       56535, 71791, 44752, 46813, 54498, 53262, 37212, 57063, 56622,
       53087, 66948, 25728, 18137, 72221, 25551, 29372, 69172, 67487,
       50871, 68969, 18025, 61123, 40303, 70533, 33191, 18511, 55802,
       18326, 59010, 53900, 20022, 62234, 67857, 49132, 59838, 28700,
       19316, 34012, 29589, 57902, 55858, 64641, 43533, 33249, 49912,
       50643, 52258, 26276, 52595, 59682, 35905, 58119, 37653, 35764,
       60345, 36076, 61023, 31121, 36629, 30196, 60624, 65721, 42995,
       33627, 54700, 59084, 15029, 70851, 35956, 60479, 62738, 46377,
       46942, 23015, 59618, 33611, 60746, 56107, 35091, 57988, 64651,
       25399, 22613, 70639, 33401, 59906, 16954, 30503, 16979, 55529,
       70457, 49810, 48253, 18524, 53372, 62029, 45074, 17187, 21994,
       48616, 46251])
```

7.0.8 Step 8. Add the column owners to cars

```
[ ]: cars['owners'] = nr_owners
     cars.tail()
```

```
      mpg  cylinders  displacement horsepower  weight  acceleration  model  \
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
195       1  ford mustang gl   45074
196       2        vw pickup   17187
197       1    dodge rampage   21994
198       1      ford ranger   48616
199       1       chevy s-10   46251
```

8 EX 7 : Online Retails Purchase

8.0.1 Introduction:

8.0.2 Step 1. Import the necessary libraries

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set(style="ticks")
```

8.0.3 Step 2. Import the dataset from this address.

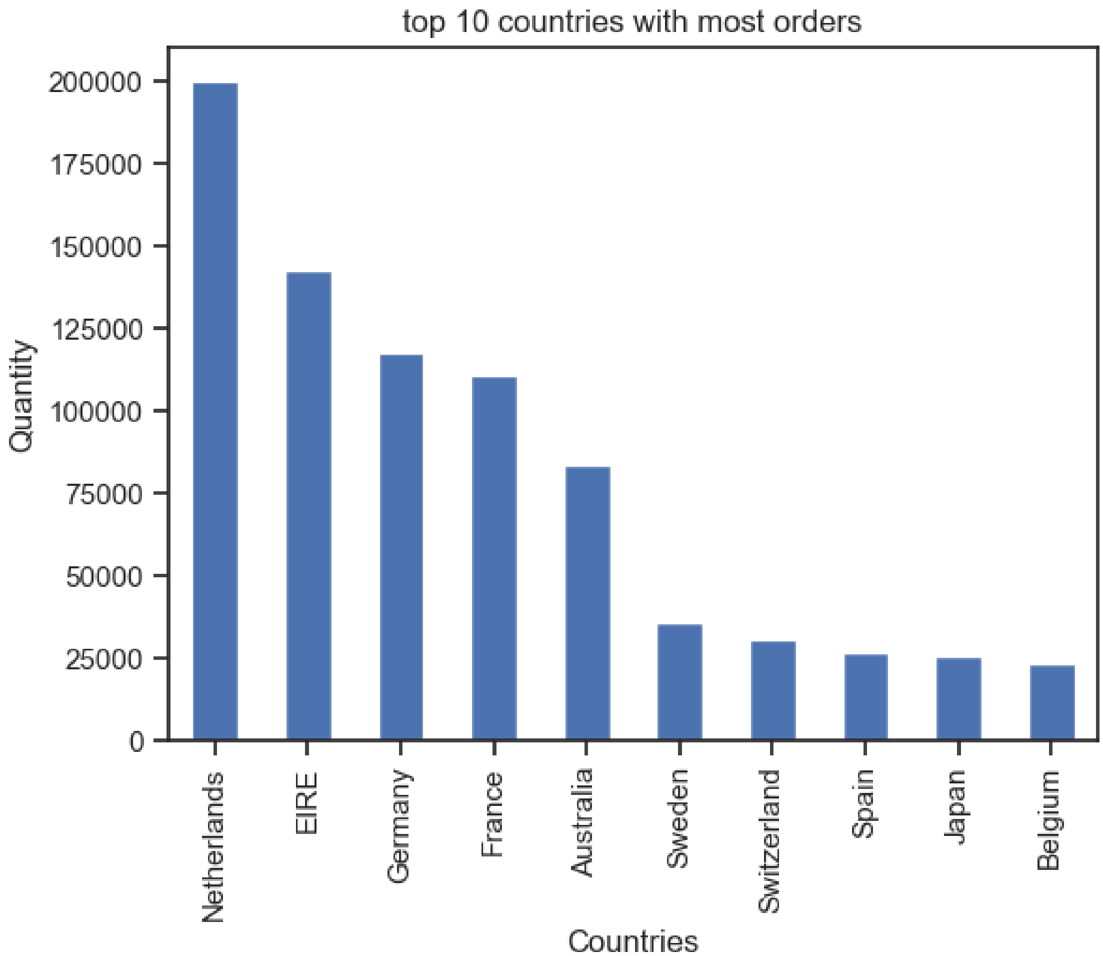
8.0.4 Step 3. Assign it to a variable called online_rt

Note: if you receive a utf-8 decode error, set `encoding = 'latin1'` in `pd.read_csv()`.

[ ]:
```python
path=r'C:\Users\risha\Documents\KRMU\AIML_assigment\datasets\online_Retail.csv'
online_rt=pd.read_csv(path, encoding = 'latin1')
```

8.0.5 Step 4. Create a histogram with the 10 countries that have the most 'Quantity' ordered except UK

[ ]:
```python
countries = online_rt.groupby(["Country"]).sum()
countries= countries.sort_values(by= 'Quantity', ascending= False)[1:11]
countries['Quantity'].plot(kind='bar')
plt.xlabel('Countries')
plt.ylabel('Quantity')
plt.title("top 10 countries with most orders")
plt.show()
```



8.0.6 Step 5. Exclude negative Quantity entries

[ ]:
```
online_rt= online_rt[online_rt.Quantity>0]
online_rt.head()
```

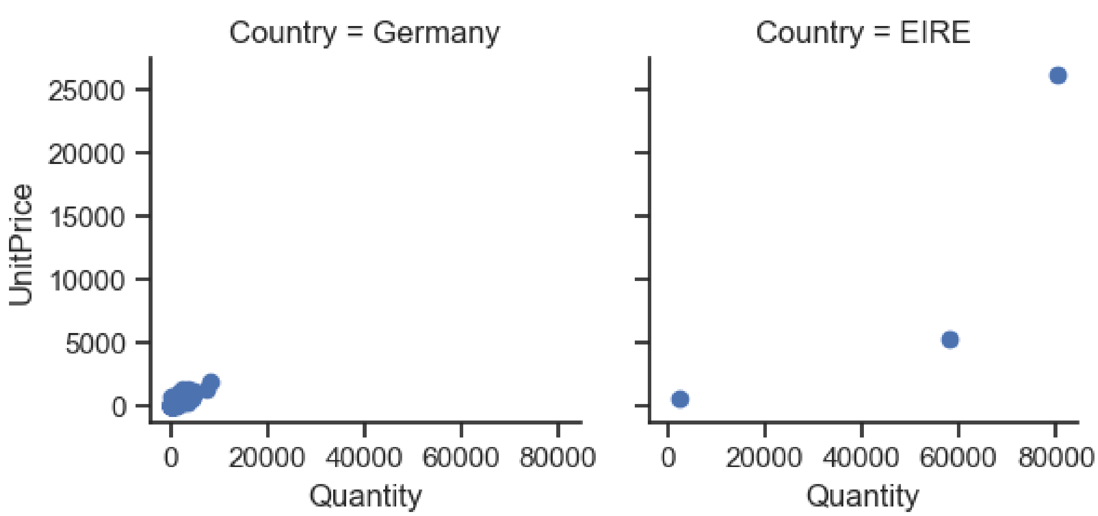
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

    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
```

8.0.7 Step 6. Create a scatterplot with the Quantity per UnitPrice by CustomerID for the top 3 Countries (except UK)

[ ]:
```
customers= online_rt.groupby(['CustomerID', 'Country']).sum()
customers= customers[customers.UnitPrice>0]
customers['Country']= customers.index.get_level_values(1)
top_countries= ['Netherland','EIRE', 'Germany']
customers= customers[customers['Country'].isin(top_countries)]

gs= sns.FacetGrid(customers, col='Country')
gs.map(plt.scatter, 'Quantity', 'UnitPrice', alpha=1)
gs.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x1af130ecce0>
```



8.0.8 Step 7. Investigate why the previous results look so uninformative.

This section might seem a bit tedious to go through. But I've thought of it as some kind of a simulation of problems one might encounter when dealing with data and other people. Besides there is a prize at the end (i.e. Section 8).

(But feel free to jump right ahead into Section 8 if you want; it doesn't require that you finish this section.)

Step 7.1 Look at the first line of code in Step 6. And try to figure out if it leads to any kind of problem.

Step 7.1.1 Display the first few rows of that DataFrame.

[ ]:
```python
customers= online_rt.groupby(['CustomerID', 'Country']).sum().head()
customers
```

```
                                                                  InvoiceNo  \
CustomerID Country
12346.0    United Kingdom                                            541431
12347.0    Iceland         5376265376265376265376265376265376265376265376…
12348.0    Finland         5393185393185393185393185393185393185393185393…
12349.0    Italy           5776095776095776095776095776095776095776095776…
12350.0    Norway          5430375430375430375430375430375430375430375430…

                                                                  StockCode  \
CustomerID Country
12346.0    United Kingdom                                             23166
12347.0    Iceland         8511622375714772249222771227722277322774227752…
12348.0    Finland         8499222951849918499121213212132261621981219822…
12349.0    Italy           2311223460215642141121563221312219548194849782…
12350.0    Norway          219082241279066K79191C2234884086C2255122557218…

                                                                Description  \
CustomerID Country
12346.0    United Kingdom                        MEDIUM CERAMIC TOP STORAGE JAR
12347.0    Iceland         BLACK CANDELABRA T-LIGHT HOLDERAIRLINE BAG VIN…
12348.0    Finland         72 SWEETHEART FAIRY CAKE CASES60 CAKE CASES DO…
12349.0    Italy           PARISIENNE CURIO CABINETSWEETHEART WALL TIDY P…
12350.0    Norway          CHOCOLATE THIS WAY METAL SIGNMETAL SIGN NEIGHB…

                           Quantity  \
CustomerID Country
12346.0    United Kingdom     74215
12347.0    Iceland             2458
12348.0    Finland             2341
12349.0    Italy                631
12350.0    Norway               197

                                                                InvoiceDate  \
CustomerID Country
12346.0    United Kingdom                                      1/18/11 10:01
12347.0    Iceland         12/7/10 14:5712/7/10 14:5712/7/10 14:5712/7/10…
12348.0    Finland         12/16/10 19:0912/16/10 19:0912/16/10 19:0912/1…
12349.0    Italy           11/21/11 9:5111/21/11 9:5111/21/11 9:5111/21/1…
12350.0    Norway          2/2/11 16:012/2/11 16:012/2/11 16:012/2/11 16:…

                           UnitPrice
CustomerID Country
12346.0    United Kingdom       1.04
12347.0    Iceland            481.21
```

```
12348.0    Finland             178.71
12349.0    Italy               605.10
12350.0    Norway               65.30
```

Step 7.1.2 Think about what that piece of code does and display the dtype of `UnitPrice`

[ ]:
```python
customers.UnitPrice.dtype
```

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

Step 7.1.3 Pull data from `online_rt`for `CustomerIDs` 12346.0 and 12347.0.

[ ]:
```python
display(online_rt[online_rt.CustomerID == 12347.0].
        sort_values(by='UnitPrice', ascending = False).head())
display(online_rt[online_rt.CustomerID == 12346.0].
        sort_values(by='UnitPrice', ascending = False).head())
```

```
       InvoiceNo StockCode             Description  Quantity  \
428966    573511     22423  REGENCY CAKESTAND 3 TIER         6
286637    562032     22423  REGENCY CAKESTAND 3 TIER         3
72267     542237     22423  REGENCY CAKESTAND 3 TIER         3
148300    549222     22423  REGENCY CAKESTAND 3 TIER         3
428967    573511     23173     REGENCY TEAPOT ROSES          2

           InvoiceDate  UnitPrice  CustomerID  Country
428966  10/31/11 12:25      12.75     12347.0  Iceland
286637    8/2/11 8:48       12.75     12347.0  Iceland
72267    1/26/11 14:30      12.75     12347.0  Iceland
148300    4/7/11 10:43      12.75     12347.0  Iceland
428967  10/31/11 12:25       9.95     12347.0  Iceland
```

```
      InvoiceNo StockCode                     Description  Quantity  \
61619    541431     23166  MEDIUM CERAMIC TOP STORAGE JAR     74215

          InvoiceDate  UnitPrice  CustomerID         Country
61619  1/18/11 10:01       1.04     12346.0  United Kingdom
```

Step 7.2 Reinterpreting the initial problem. To reiterate the question that we were dealing with:
"Create a scatterplot with the Quantity per UnitPrice by CustomerID for the top 3 Countries"

The question is open to a set of different interpretations. We need to disambiguate.

We could do a single plot by looking at all the data from the top 3 countries. Or we could do one plot per country. To keep things consistent with the rest of the exercise, let's stick to the latter oprion. So that's settled.

But "top 3 countries" with respect to what? Two answers suggest themselves: Total sales volume (i.e. total quantity sold) or total sales (i.e. revenue). This exercise goes for sales volume, so let's stick to that.

Step 7.2.1 Find out the top 3 countries in terms of sales volume.

```
[ ]: sales_volume = online_rt.groupby('Country').Quantity.sum().
      ↪sort_values(ascending=False)

     top3 = sales_volume.index[1:4]
     top3
```

```
Index(['Netherlands', 'EIRE', 'Germany'], dtype='object', name='Country')
```

Step 7.2.2 Now that we have the top 3 countries, we can focus on the rest of the problem: "Quantity per UnitPrice by CustomerID".
We need to unpack that.

"by CustomerID" part is easy. That means we're going to be plotting one dot per CustomerID's on our plot. In other words, we're going to be grouping by CustomerID.

"Quantity per UnitPrice" is trickier. Here's what we know:
One axis will represent a Quantity assigned to a given customer. This is easy; we can just plot the total Quantity for each customer.
The other axis will represent a UnitPrice assigned to a given customer. Remember a single customer can have any number of orders with different prices, so summing up prices isn't quite helpful. Besides it's not quite clear what we mean when we say "unit price per customer"; it sounds like price of the customer! A reasonable alternative is that we assign each customer the average amount each has paid per item. So let's settle that question in that manner.

Step 7.3 Modify, select and plot data

Step 7.3.1 Add a column to online_rt called `Revenue` calculate the revenue (Quantity * UnitPrice) from each sale. We will use this later to figure out an average price per customer.

```
[ ]: online_rt['Revenue'] = online_rt.Quantity * online_rt.UnitPrice
     online_rt.head()
```

```
  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

    InvoiceDate  UnitPrice  CustomerID         Country  Revenue
0  12/1/10 8:26       2.55     17850.0  United Kingdom    15.30
1  12/1/10 8:26       3.39     17850.0  United Kingdom    20.34
2  12/1/10 8:26       2.75     17850.0  United Kingdom    22.00
3  12/1/10 8:26       3.39     17850.0  United Kingdom    20.34
4  12/1/10 8:26       3.39     17850.0  United Kingdom    20.34
```

Step 7.3.2 Group by `CustomerID` and `Country` and find out the average price (`AvgPrice`) each customer spends per unit.

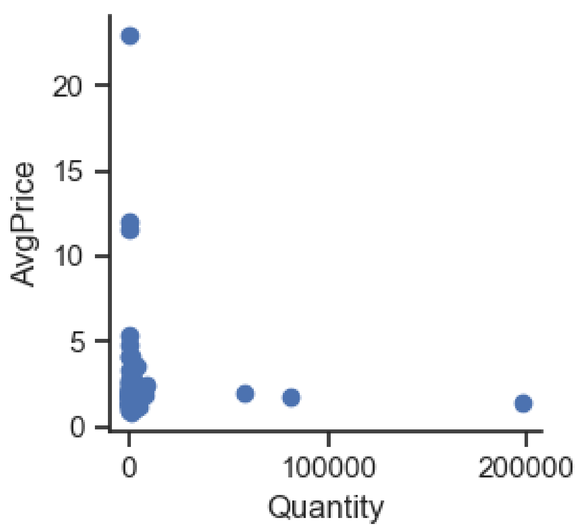
[ ]:
```
grouped = online_rt[online_rt.Country.isin(top3)].
  ↪groupby(['CustomerID','Country'])

plottable = grouped[['Quantity','Revenue']].agg('sum')
plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

plottable['Country','Revenue',] = plottable.index.get_level_values(1)
plottable.head()
```

```
                    Quantity   Revenue   AvgPrice (Country, Revenue)
CustomerID Country
12426.0    Germany       258    582.73   2.258643            Germany
12427.0    Germany       533    825.80   1.549343            Germany
12468.0    Germany       366    729.54   1.993279            Germany
12471.0    Germany      8212  19824.05   2.414034            Germany
12472.0    Germany      4148   6572.11   1.584405            Germany
```

Step 7.3.3 Plot

[ ]:
```
g = sns.FacetGrid(plottable)
g.map(plt.scatter, "Quantity", "AvgPrice", alpha=1)

g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x1af1317bec0>
```



Step 7.4 What to do now? We aren't much better-off than what we started with. The data are still extremely scattered around and don't seem quite informative.

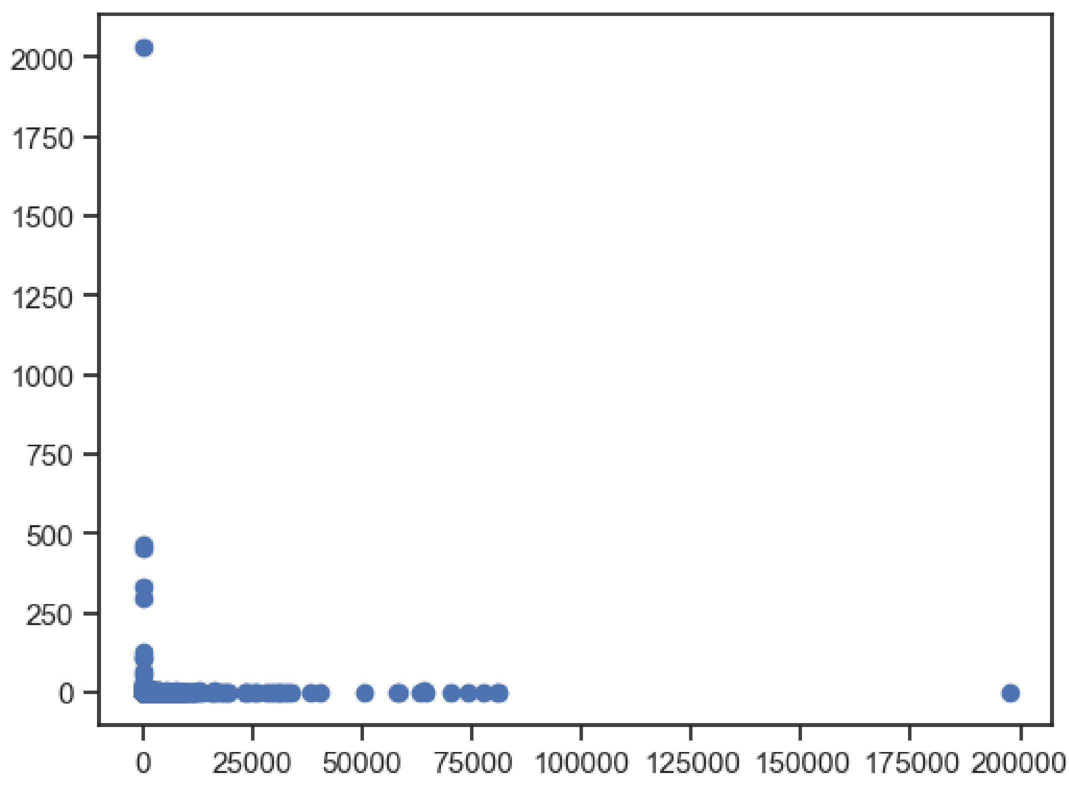
But we shouldn't despair! There are two things to realize: 1) The data seem to be skewed towaards the axes (e.g. we don't have any values where Quantity = 50000 and AvgPrice = 5). So that might suggest a trend. 2) We have more data! We've only been looking at the data from 3 different countries and they are plotted on different graphs.

So: we should plot the data regardless of `Country` and hopefully see a less scattered graph.

Step 7.4.1 Plot the data for each `CustomerID` on a single graph

[ ]:
```
grouped = online_rt.groupby(['CustomerID'])
plottable = grouped[['Quantity','Revenue']].agg('sum')
plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

plt.scatter(plottable.Quantity, plottable.AvgPrice)
plt.plot()
```

```
[]
```

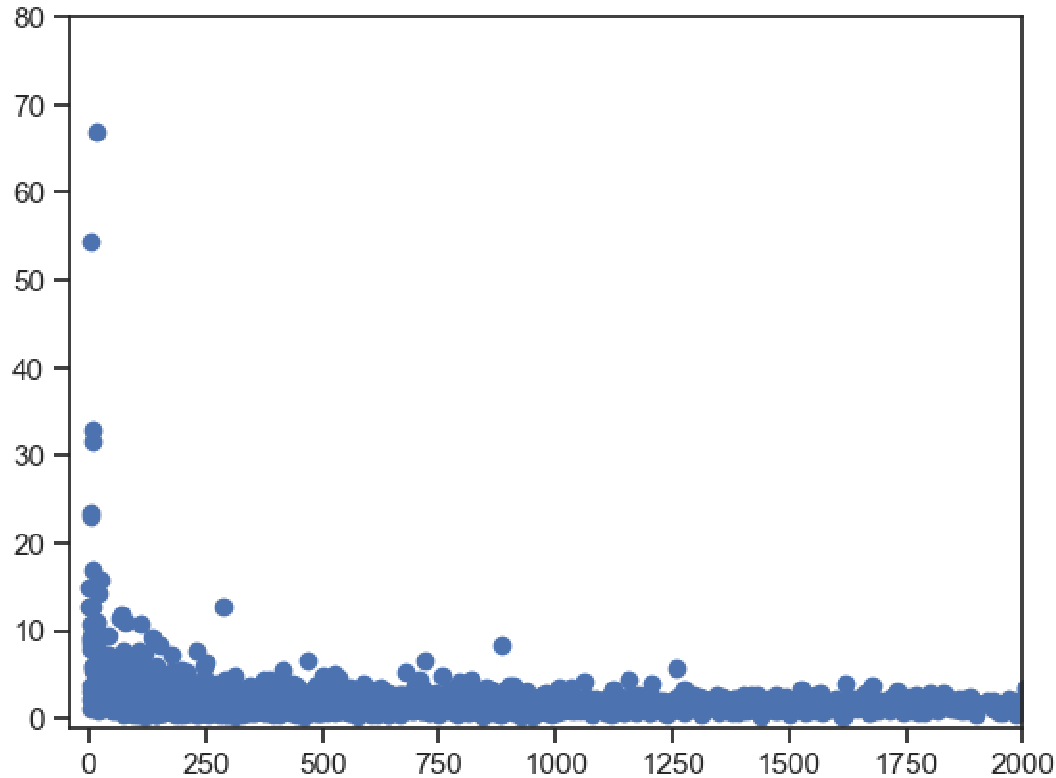


Step 7.4.2 Zoom in so we can see that curve more clearly

[ ]:
```
grouped = online_rt.groupby(['CustomerID','Country'])
plottable = grouped.agg({'Quantity': 'sum',
                         'Revenue': 'sum'})
plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity
```

```
plt.scatter(plottable.Quantity, plottable.AvgPrice)

plt.xlim(-40,2000)
plt.ylim(-1,80)

plt.plot()
```

```
[]
```



8.0.9 8. Plot a line chart showing revenue (y) per UnitPrice (x).

Did Step 7 give us any insights about the data? Sure! As average price increases, the quantity ordered decreses. But that's hardly surprising. It would be surprising if that wasn't the case!

Nevertheless the rate of drop in quantity is so drastic, it makes me wonder how our revenue changes with respect to item price. It would not be that surprising if it didn't change that much. But it would be interesting to know whether most of our revenue comes from expensive or inexpensive items, and how that relation looks like.

That is what we are going to do now.

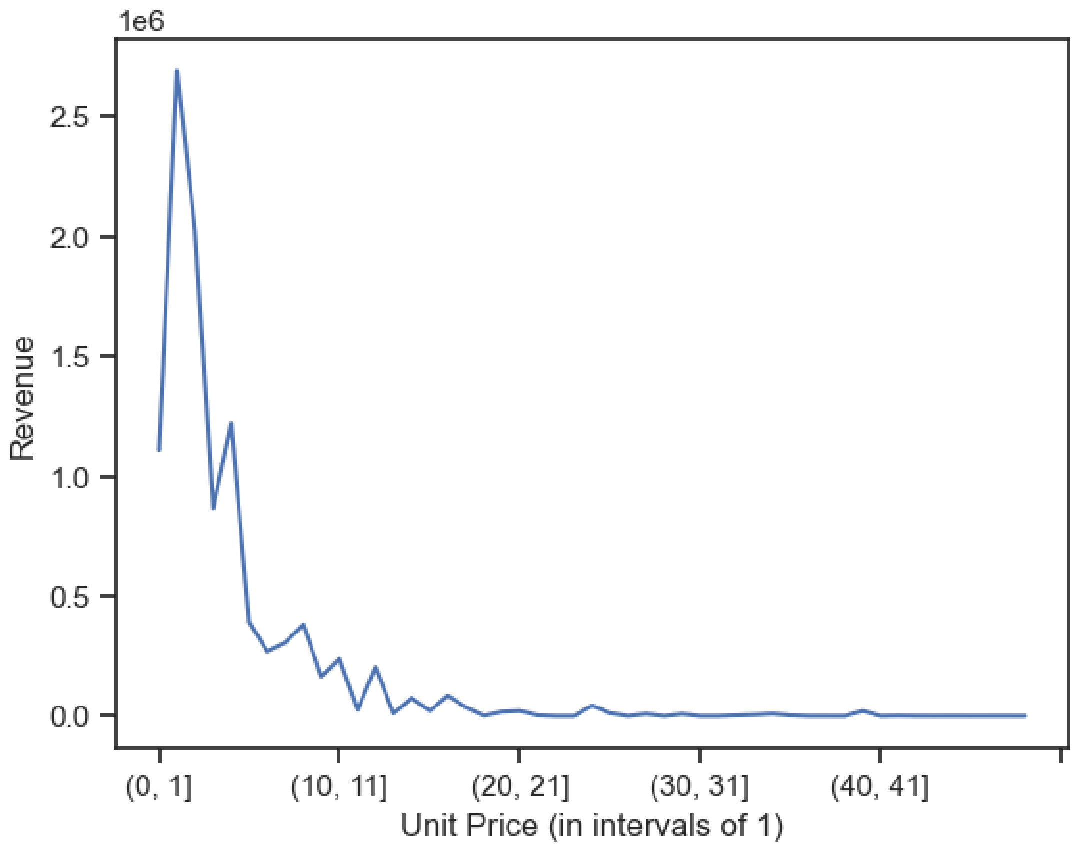
8.1 Group `UnitPrice` by intervals of 1 for prices [0,50), and sum `Quantity` and `Revenue`.

[ ]:
```python
price_start = 0
price_end = 50
price_interval = 1

buckets = np.arange(price_start,price_end,price_interval)

revenue_per_price = online_rt.groupby(pd.cut(online_rt.UnitPrice,␣
↪buckets),observed=False).Revenue.sum()
revenue_per_price.head()
```

```
UnitPrice
(0, 1]     1107774.544
(1, 2]     2691765.110
(2, 3]     2024143.090
(3, 4]      865101.780
(4, 5]     1219377.050
Name: Revenue, dtype: float64
```

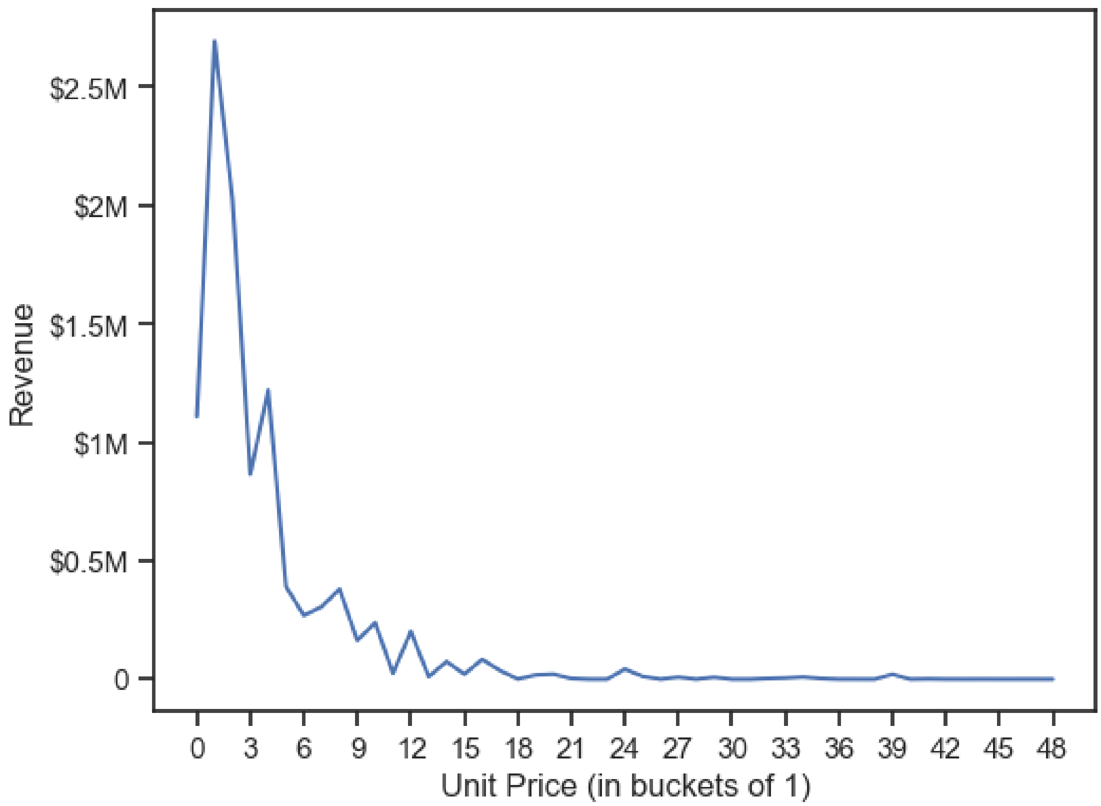
8.3 Plot.

[ ]:
```python
revenue_per_price.plot()
plt.xlabel('Unit Price (in intervals of '+str(price_interval)+')')
plt.ylabel('Revenue')
plt.show()
```



8.4 Make it look nicer. x-axis needs values.
y-axis isn't that easy to read; show in terms of millions.

[ ]:
```
revenue_per_price.plot()

plt.xlabel('Unit Price (in buckets of '+str(price_interval)+')')
plt.ylabel('Revenue')

plt.xticks(np.arange(price_start,price_end,3),
           np.arange(price_start,price_end,3))
plt.yticks([0, 500000, 1000000, 1500000, 2000000, 2500000],
           ['0', '$0.5M', '$1M', '$1.5M', '$2M', '$2.5M'])
plt.show()
```



$2.5M
$2M
$1.5M
$1M
$0.5M
0
Revenue
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48
Unit Price (in buckets of 1)


```
[ ]:
```