Assignment 2 : Data Analysis with pandas and Visualization Libraries

Name: Navneet Yadav

Roll no. 2301560044

Program: MCA-I

Subject: AI-ML

Question 1

- GitHub Link: https://github.com/navYadav20/AI-ML- (https://github.com/navYadav20/AI-ML-)

---

1. General Pandas Exercise

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

In [32]:
```
import pandas as pd
df = pd.read_csv("Automobile_data.csv")
# first five rows
df.head(5)
```

Out[32]:

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	
0	0	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	134
1	1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	165
2	2	alfa-romero	hatchback	94.5	171.2	ohcv	six	154	19	165
3	3	audi	sedan	99.8	176.6	ohc	four	102	24	139
4	4	audi	sedan	99.4	176.6	ohc	five	115	18	174

In [33]:
```
# last five rows
df.tail(5)
```

Out[33]:

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	p
56	81	volkswagen	sedan	97.3	171.7	ohc	four	85	27	797
57	82	volkswagen	sedan	97.3	171.7	ohc	four	52	37	799
58	86	volkswagen	sedan	97.3	171.7	ohc	four	100	26	999
59	87	volvo	sedan	104.3	188.8	ohc	four	114	23	1294
60	88	volvo	wagon	104.3	188.8	ohc	four	114	23	134

Exercise 2: Clean the dataset and update the CSV file

In [5]:
```
import pandas as pd

df.replace({
    'price': ["?", "n.a"],
    'stroke': ["?", "n.a"],
    'horsepower': ["?", "n.a"],
    'peak-rpm': ["?", "n.a"],
    'average-mileage': ["?", "n.a"]
}, inplace=True)

print(df)

df.to_csv("Automobile_dataset_cleaned.csv", index=False)
```

```
    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
..    ...          ...          ...         ...     ...         ...
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
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
..              ...         ...              ...      ...
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

[61 rows x 10 columns]
```

Exercise 3: Find the most expensive car company name

In [6]:
```
df = df [['company','price']][df.price==df['price'].max()]
df
```

Out[6]:

	company	price
35	mercedes-benz	45400.0

Exercise 4: Print All Toyota Cars details

In [7]:
```
df = pd.read_csv("Automobile_data.csv")
toyota_cars = df[df['company'] == 'toyota']
toyota_cars
```

Out[7]:

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	
48	66	toyota	hatchback	95.7	158.7	ohc	four	62	35	5
49	67	toyota	hatchback	95.7	158.7	ohc	four	62	31	6
50	68	toyota	hatchback	95.7	158.7	ohc	four	62	31	6
51	69	toyota	wagon	95.7	169.7	ohc	four	62	31	6
52	70	toyota	wagon	95.7	169.7	ohc	four	62	27	7
53	71	toyota	wagon	95.7	169.7	ohc	four	62	27	8
54	79	toyota	wagon	104.5	187.8	dohc	six	156	19	15

Exercise 5: Count total cars per company

In [8]:
```
df['company'].value_counts()
```

Out[8]:
```
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

In [9]:
```
car_Manufacturers = df.groupby('company')
priceDf = car_Manufacturers[['company', 'price']].max()
priceDf
```

Out[9]:

company	company	price
alfa-romero	alfa-romero	16500.0
audi	audi	18920.0
bmw	bmw	41315.0
chevrolet	chevrolet	6575.0
dodge	dodge	6377.0
honda	honda	12945.0
isuzu	isuzu	6785.0
jaguar	jaguar	36000.0
mazda	mazda	18344.0
mercedes-benz	mercedes-benz	45400.0
mitsubishi	mitsubishi	8189.0
nissan	nissan	13499.0
porsche	porsche	37028.0
toyota	toyota	15750.0
volkswagen	volkswagen	9995.0
volvo	volvo	13415.0

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

In [10]:
```
unique_values = df['average-mileage'].unique()

# Print unique values
print("Unique values in 'average-mileage' column:")
print(unique_values)
```

```
Unique values in 'average-mileage' column:
[21 19 24 18 23 16 15 47 38 31 30 25 13 17 22 14 37 45 35 27 26]
```

In [11]:
```
car_Manufacturers = df.groupby('company')
mileageDf = car_Manufacturers['average-mileage'].mean()
mileageDf
```

Out[11]:
```
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

In [12]:
```
df = df.sort_values(by=['price', 'horsepower'], ascending=False)
df.head(5)
```

Out[12]:

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	
35	47	mercedes-benz	hardtop	112.0	199.2	ohcv	eight	184	14	4
11	14	bmw	sedan	103.5	193.8	ohc	six	182	16	4
34	46	mercedes-benz	sedan	120.9	208.1	ohcv	eight	184	14	4
46	62	porsche	convertible	89.5	168.9	ohcf	six	207	17	3
12	15	bmw	sedan	110.0	197.0	ohc	six	182	15	3

Exercise 9: Concatenate two data frames using the following conditions

In [13]:
```
GermanCars = {'Company': ['Ford', 'Mercedes', 'BMW', 'Audi'], 'Price': [238
carsDf1 = pd.DataFrame.from_dict(GermanCars)

japaneseCars = {'Company': ['Toyota', 'Honda', 'Nissan', 'Mitsubishi '], 'P
carsDf2 = pd.DataFrame.from_dict(japaneseCars)

carsDf = pd.concat([carsDf1, carsDf2], keys=["Germany", "Japan"])
carsDf
```

Out[13]:

		Company	Price
Germany	0	Ford	23845
	1	Mercedes	171995
	2	BMW	135925
	3	Audi	71400
Japan	0	Toyota	29995
	1	Honda	23600
	2	Nissan	61500
	3	Mitsubishi	58900

Exercise 10: Merge two data frames using the following condition

In [14]:
```
Car_Price = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'Price': [23845
carPriceDf = pd.DataFrame.from_dict(Car_Price)

car_Horsepower = {'Company': ['Toyota', 'Honda', 'BMV', 'Audi'], 'horsepowe
carsHorsepowerDf = pd.DataFrame.from_dict(car_Horsepower)

carsDf = pd.merge(carPriceDf, carsHorsepowerDf, on="Company")
carsDf
```

Out[14]:

	Company	Price	horsepower
0	Toyota	23845	141
1	Honda	17995	80
2	BMV	135925	182
3	Audi	71400	160

In [ ]:
```

```

2. Simple Exercise

Import the necessary libraries

In [34]:
```
import pandas as pd
import numpy as np
```

Import the dataset and assign it to the variable chipo

In [35]:
```
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipo

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

In [36]:
```
chipo.head(10)
```

Out[36]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

What is the number of observations in the dataset?

In [18]:
```
chipo.shape[0]
```

Out[18]:
```
4622
```

In [19]:
```
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
```

What is the number of columns in the dataset?

In [20]:
```
chipo.shape[1]
```

Out[20]:
```
5
```

Print the name of columns in the dataset?

In [21]:
```
chipo.columns
```

Out[21]:
```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'],
      dtype='object')
```

How is the dataset indexed?

In [22]:
```
chipo.index
```

Out[22]:
```
RangeIndex(start=0, stop=4622, step=1)
```

Which was the most-ordered item?

In [23]:
```
c = chipo.groupby('item_name')
c = c.sum()
c = c.sort_values(['quantity'], ascending=False)
c.head(1)
```

Out[23]:

item_name	order_id	quantity	choice_description	item_price
Chicken Bowl	713926	761	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	16.9810.98 11.258.75 8.49 11.25 $8.75 ...

What was the most ordered item in the choice_description column?

In [24]:
```
c = chipo.groupby('choice_description').sum()
c = c.sort_values(['quantity'], ascending=False)
c.head(1)
```

Out[24]:

choice_description	order_id	quantity	item_name	item_price
[Diet Coke]	123455	159	Canned SodaCanned SodaCanned Soda6 Pack Soft D...	2.181.09 1.096.49 2.18 1.25 1.096.4...

How many items were orderd in total?

In [25]:
```
total_order = chipo.quantity.sum()
total_order
```

Out[25]:
```
4972
```

Turn the item price into a float

a. Check the item price type

In [26]:
```
chipo.item_price.dtype
```

Out[26]:
```
dtype('O')
```

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

In [27]:
```
dollarizer = lambda x: float(x[1:-1])
chipo.item_price = chipo.item_price.apply(dollarizer)
```

c. Check the item price type

In [28]:
```
chipo.item_price.dtype
```

Out[28]:
```
dtype('float64')
```

How much was the revenue for the period in the dataset?

In [29]:
```
revenue = (chipo['quantity']* chipo['item_price']).sum()

print('Revenue was: $' + str(np.round(revenue,2)))
```

```
Revenue was: $39237.02
```

How many orders were made in the period?

In [30]:
```
orders = chipo.order_id.value_counts().count()
orders
```

Out[30]: 1834

What is the average revenue amount per order?

In [ ]:
```
chipo['revenue'] = chipo['quantity'] * chipo['item_price']
order_grouped = chipo.groupby(by=['order_id']).sum()
order_grouped.mean()['revenue']
```

21.394231188658654

How many different items are sold?

In [37]:
```
chipo.item_name.value_counts().count()
```

Out[37]: 50

In [ ]:
```

```

3. Filtering and Sorting Exercise

https://nbviewer.jupyter.org/github/guipsamora
(https://nbviewer.jupyter.org/github/guipsamor

Import the necessary libraries

In [38]:
```
import pandas as pd
```

Import dataset and assign to variable euro12

In [39]:
```
euro12 = pd.read_csv('https://raw.githubusercontent.com/guipsamora/pandas_e
euro12
```

Out[39]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Pena sc
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	
3	England	5	11	18	50.0%	17.2%	40	0	0	
4	France	3	22	24	37.9%	6.5%	65	1	0	
5	Germany	10	32	32	47.8%	15.6%	80	2	1	
6	Greece	5	8	18	30.7%	19.2%	32	1	1	
7	Italy	6	34	45	43.0%	7.5%	110	2	0	
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	
9	Poland	2	15	23	39.4%	5.2%	48	0	0	
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	
12	Russia	5	9	31	22.5%	12.5%	59	2	0	
13	Spain	12	42	33	55.9%	16.0%	100	0	1	
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	

16 rows × 35 columns

Select only the Goal column.

In [40]:
```
euro12.Goals
```

Out[40]:
```
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
```

How many team participated in the Euro2012?

In [41]:
```
euro12.shape[0]
```

Out[41]:
```
16
```

What is the number of columns in the dataset?

In [42]:
```
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
```

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

In [43]:
```
discipline = euro12[['Team', 'Yellow Cards', 'Red Cards']]
discipline
```

Out[43]:

	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

Sort the teams by Red Cards, then to Yellow Cards

In [44]:
```
discipline.sort_values(['Red Cards', 'Yellow Cards'], ascending = False)
```

Out[44]:

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

Calculate the mean Yellow Cards given per Team

In [45]:
```
round(discipline['Yellow Cards'].mean())
```

Out[45]: 7

Filter teams that scored more than 6 goals

In [46]:
```
euro12[euro12.Goals>6]
```

Out[46]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penaltie n score
5	Germany	10	32	32	47.8%	15.6%	80	2	1	
13	Spain	12	42	33	55.9%	16.0%	100	0	1	

2 rows × 35 columns

Select the teams that start with G

In [47]:
```
euro12[euro12.Team.str.startswith('G')]
```

Out[47]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1

2 rows × 35 columns

Select the first 7 columns

In [48]:
```
euro12.iloc[:, 0:7]
```

Out[48]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)
0	Croatia	4	13	12	51.9%	16.0%	32
1	Czech Republic	4	13	18	41.9%	12.9%	39
2	Denmark	4	10	10	50.0%	20.0%	27
3	England	5	11	18	50.0%	17.2%	40
4	France	3	22	24	37.9%	6.5%	65
5	Germany	10	32	32	47.8%	15.6%	80
6	Greece	5	8	18	30.7%	19.2%	32
7	Italy	6	34	45	43.0%	7.5%	110
8	Netherlands	2	12	36	25.0%	4.1%	60
9	Poland	2	15	23	39.4%	5.2%	48
10	Portugal	6	22	42	34.3%	9.3%	82
11	Republic of Ireland	1	7	12	36.8%	5.2%	28
12	Russia	5	9	31	22.5%	12.5%	59
13	Spain	12	42	33	55.9%	16.0%	100
14	Sweden	5	17	19	47.2%	13.8%	39
15	Ukraine	2	7	26	21.2%	6.0%	38

Select all columns except the last 3.

In [49]:
```
euro12.iloc[: , :-3]
```

Out[49]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Pena sc
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	
3	England	5	11	18	50.0%	17.2%	40	0	0	
4	France	3	22	24	37.9%	6.5%	65	1	0	
5	Germany	10	32	32	47.8%	15.6%	80	2	1	
6	Greece	5	8	18	30.7%	19.2%	32	1	1	
7	Italy	6	34	45	43.0%	7.5%	110	2	0	
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	
9	Poland	2	15	23	39.4%	5.2%	48	0	0	
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	
12	Russia	5	9	31	22.5%	12.5%	59	2	0	
13	Spain	12	42	33	55.9%	16.0%	100	0	1	
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	

16 rows × 32 columns

Present only the Shooting Accuracy from England, Italy and Russia

In [50]:
```
euro12.loc[euro12.Team.isin(['England', 'Italy', 'Russia']), ['Team','Shoot
```

Out[50]:

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

4. GroupBy Exercise

https://nbviewer.jupyter.org/github/guipsamora

In [51]:
```
import pandas as pd
```

Import dataset and assign to variable drinks

In [52]:
```
drinks = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/ma
drinks.head()
```

Out[52]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	con
0	Afghanistan	0	0	0	0.0	
1	Albania	89	132	54	4.9	
2	Algeria	25	0	14	0.7	
3	Andorra	245	138	312	12.4	
4	Angola	217	57	45	5.9	

Which continent drinks more beer on average?

In [53]:
```
drinks.groupby('continent').beer_servings.mean()
```

Out[53]:
```
continent
AF     61.471698
AS     37.045455
EU    193.777778
OC     89.687500
SA    175.083333
Name: beer_servings, dtype: float64
```

For each continent print the statistics for wine consumption.

In [54]:
```
drinks.groupby('continent').wine_servings.describe()
```

Out[54]:

continent	count	mean	std	min	25%	50%	75%	max
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

Print the mean alcohol consumption per continent for every column

In [55]:
```
# Assuming 'continent' column contains numeric values
numeric_continent_data = drinks[pd.to_numeric(drinks['continent'], errors='

# Now you can calculate the mean
mean_numeric_continent = numeric_continent_data.groupby('continent').mean()

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

Out[55]:
```
country                         NaN
beer_servings                   NaN
spirit_servings                 NaN
wine_servings                   NaN
total_litres_of_pure_alcohol    NaN
dtype: object
```

In [56]:
```
drinks.continent.dtype
```

Out[56]:
```
dtype('O')
```

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

In [57]:
```
drinks.groupby('continent').spirit_servings.agg(['mean', 'min', 'max'])
```

Out[57]:

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

5. Apply method

https://nbviewer.jupyter.org/github/guipsamora (https://nbviewer.jupyter.org/github/guipsamor

Student Alcohol Consumption

In [58]:
```
import pandas as pd
import numpy as np
```

Import dataset and assign to variable called df

In [59]:
```
csv_url = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/ma
df = pd.read_csv(csv_url)
df.head()
```

Out[59]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4
3	GP	F	15	U	GT3	T	4	2	health	services	...	3
4	GP	F	16	U	GT3	T	3	3	other	other	...	4

5 rows × 33 columns

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

In [60]:
```
stud_alcoh = df.loc[: , "school":"guardian"]
stud_alcoh.head()
```

Out[60]:

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

Create a lambda function that will capitalize strings.

In [61]:
```
capitalizer = lambda x: x.capitalize()
```

Capitalize both Mjob and Fjob

In [62]:
```
stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'].apply(capitalizer)
```

Out[62]:
```
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
```

Print the last elements of the data set.

In [63]:
```
stud_alcoh.tail()
```

Out[63]:

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

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

In [64]:
```
stud_alcoh['Mjob'] = stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'] = stud_alcoh['Fjob'].apply(capitalizer)
stud_alcoh.tail()
```

Out[64]:

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

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)

In [65]:
```
def majority(x):
    if x > 17:
        return True
    else:
        return False
```

In [66]:
```
stud_alcoh['legal_drinker'] = stud_alcoh['age'].apply(majority)
stud_alcoh.head()
```

Out[66]:

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

Multiply every number of the dataset by 10.

In [67]:
```
def times10(x):
    if type(x) is int:
        return 10 * x
    return x
```

In [68]:
```
stud_alcoh.applymap(times10).head(10)
```

```
C:\Users\msuse\AppData\Local\Temp\ipykernel_7308\1982092898.py:1: FutureWa
rning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  stud_alcoh.applymap(times10).head(10)
```

Out[68]:

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

6. Merge

https://nbviewer.jupyter.org/github/guipsamora
(https://nbviewer.jupyter.org/github/guipsamor

MPG Cars

Import the necessary libraries

In [69]:
```
import pandas as pd
import numpy as np
```

Import dataset and assign to variables

In [70]:
```
cars1 = pd.read_csv("https://raw.githubusercontent.com/guipsamora/pandas_ex
cars2 = pd.read_csv("https://raw.githubusercontent.com/guipsamora/pandas_ex

print(cars1.head())
print(cars2.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  Unnamed: 9  Unnamed: 10  Unnamed: 11
\
0       1  chevrolet chevelle malibu         NaN          NaN          NaN
1       1          buick skylark 320         NaN          NaN          NaN
2       1         plymouth satellite         NaN          NaN          NaN
3       1              amc rebel sst         NaN          NaN          NaN
4       1                ford torino         NaN          NaN          NaN

   Unnamed: 12  Unnamed: 13
0          NaN          NaN
1          NaN          NaN
2          NaN          NaN
3          NaN          NaN
4          NaN          NaN
    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

   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
```

Oops, it seems our first dataset has some unnamed blank columns, fix cars1

In [71]:
```
cars1 = cars1.loc[:, "mpg":"car"]
cars1.head()
```

Out[71]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	car
0	18.0	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302	140	3449	10.5	70	1	ford torino

What is the number of observations in each dataset ?

In [72]:
```
print(cars1.shape)
print(cars2.shape)
```

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

Join cars1 and cars2 into a single DataFrame called cars

In [73]:
```
cars = pd.concat([cars1, cars2], ignore_index=True)
cars
```

Out[73]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	ca
0	18.0	8	307	130	3504	12.0	70	1	chevrole chevelle malibu
1	15.0	8	350	165	3693	11.5	70	1	buicl skylarl 32(
2	18.0	8	318	150	3436	11.0	70	1	plymoutl satellite
3	16.0	8	304	150	3433	12.0	70	1	amc rebel ss
4	17.0	8	302	140	3449	10.5	70	1	forc torinc
...	...	...	...	...	...	...	...	...	..
393	27.0	4	140	86	2790	15.6	82	1	forc mustang g
394	44.0	4	97	52	2130	24.6	82	2	vw picku
395	32.0	4	135	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120	79	2625	18.6	82	1	forc range
397	31.0	4	119	82	2720	19.4	82	1	chevy s 1(

398 rows × 9 columns

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

In [74]:
```
owners = np.random.randint(1500, high=73001, size=398, dtype="l")
owners
```

Out[74]:
```
array([31385, 25396, 17270, 37191, 63670, 51483, 49623, 12572, 57831,
       53062, 28244, 31274, 65948, 16802, 29703, 72758, 54484, 22063,
       55618, 22182, 44847, 40547, 63526,  7813, 72778, 52500, 66791,
       29384, 43869, 54522, 31420, 32712, 48743, 69132, 25431, 48199,
       58772, 50709, 60310, 12041, 60351, 29905, 53421, 49510,  6203,
       39691, 18948, 15339, 19908, 10614, 33293, 29691, 54072, 27893,
       44799, 30241, 32148, 60098,  4805, 44031, 41016, 53497, 49511,
       54864, 40502, 33911, 67091, 21467, 56949, 42842, 53030, 15682,
       43243, 35458, 62045, 22432,  4363, 69944, 51604, 17048,  8633,
       21530, 19026, 33445, 56175, 21564, 54326, 15233, 71329,  9568,
       39474,  6855, 19555, 72457, 60680, 29765, 59301, 10316, 53541,
       61508, 39781,  8815, 68060, 18226,  2809, 66758,  3001, 51949,
       32013, 41718, 20396, 18626, 32956, 11913,  2270,  3052, 42243,
       14065, 11386, 34606, 24508,  2653, 54447, 23687, 36245, 12812,
       57298, 69404, 49057, 55156, 61090,  1761, 25705, 47665, 37621,
       11794, 17822, 68637,  7967, 57474, 21926, 65727, 56681, 53570,
       41195, 45576,  7708, 59031, 48781, 51857, 56545, 21477, 11616,
       43136, 19095,  6435, 59290,  5960, 45078, 17680, 65862, 24869,
       53198, 25892, 43512, 50026,  8882, 63911, 24989, 16549, 20427,
       38031,  1772, 72179, 45311, 37056, 22109, 56485, 19249, 12153,
       48855, 32380, 35993, 26527, 52104, 70171, 26728, 61740, 39968,
       60288, 11347, 45531, 48337, 13284,  5848, 45223, 40959, 57353,
       11058, 33165, 54799, 27105, 17038, 10315, 69001,  9635, 34673,
       55808, 41778, 62763,  6653, 15412,  1580, 29208,  6970,  3679,
       68631, 46650, 14741, 67512, 63032, 44240, 56087,  6445, 30450,
       65867, 35972, 34551, 67505, 50149,  7251, 61389, 70860, 20610,
       67982, 10134,  1654, 69279, 70239, 31861, 15701, 15142,  3095,
       44309, 50738, 51367, 65414, 64793, 17211, 51484, 65404, 44129,
       48637, 48393, 65749, 60917, 38454, 67977, 14594, 60191, 50227,
       24268, 47105, 17385, 27222, 30506, 57291, 10709, 52700,  3412,
       15889, 42556, 62498, 32738, 66051, 35433, 40948, 31925,  9033,
       43148, 61471, 64795, 24471, 20871, 63948, 38369, 55037, 56241,
       48302, 19086,  5381, 22790, 38920, 38893, 40547, 71645, 41931,
       14722, 13782, 72218, 26625, 23797,  6229, 10635, 71745, 67711,
       43066, 43235,  5003, 16118, 46683, 31560,  7953, 46330, 32952,
        8464, 65533, 72897,  8436,  1845, 16642,  9607, 35498,  8676,
       39389, 61300, 59274, 35121, 21047, 49598, 46909, 12128, 18192,
       13739, 61564, 39488, 24417, 11620, 15651, 34080,  4180, 51102,
       31383, 13727, 68028, 69987, 42333,  2988, 42922, 58226, 55439,
       10245, 55993, 52942, 45968, 28571, 55181, 34562,  2049, 34374,
       38426, 65590, 70841, 57781, 13564,  3792,  9002,  7594, 55871,
       59767, 57783, 33559, 31045, 31166, 28735, 21750, 49252, 42967,
       70600, 56079, 21016, 17903, 50970, 28577, 30314,  5295, 31135,
       18088, 47711, 60825, 18952, 51922, 56817, 59628, 17835, 15159,
        2167,  7204])
```

Add the column owners to cars

In [75]:
```
cars['owners'] = owners
cars.tail()
```

Out[75]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	car
393	27.0	4	140	86	2790	15.6	82	1	ford mustang g
394	44.0	4	97	52	2130	24.6	82	2	vw pickup
395	32.0	4	135	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120	79	2625	18.6	82	1	ford range
397	31.0	4	119	82	2720	19.4	82	1	chevy s 10

Type *Markdown* and LaTeX: α^2

7. Data Vizulatization

https://nbviewer.jupyter.org/github/guipsamora (https://nbviewer.jupyter.org/github/guipsamor

Online Reatails Purchase

Import the necessary libraries

In [76]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

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

Import the dataset and assign to a variable called online_rt

In [77]:
```
path = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/maste

online_rt = pd.read_csv(path, encoding = 'latin1')

online_rt.head()
```

Out[77]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850.0	United Kingdom

Create a histogram with the 10 countries that have the most 'Quantity' ordered except UK

In [ ]:
```
countries = online_rt.groupby('Country').sum()

# sort the value and get the first 10 after UK
countries = countries.sort_values(by = 'Quantity',ascending = False)[1:11]

# create the plot
countries['Quantity'].plot(kind='bar')

# Set the title and labels
plt.xlabel('Countries')
plt.ylabel('Quantity')
plt.title('10 Countries with most orders')

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

Exclude negative Quantity entries

In [78]:
```
online_rt = online_rt[online_rt.Quantity > 0]
online_rt.head()
```

Out[78]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850.0	United Kingdom

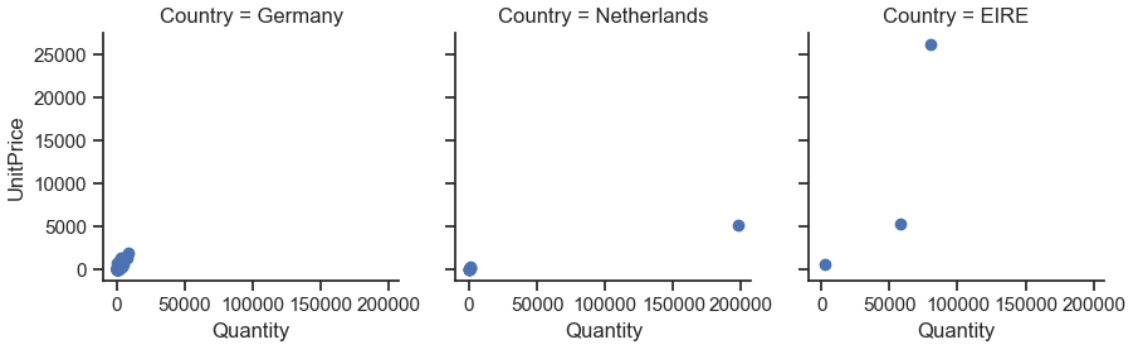
In [ ]:
```
```

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

In [79]:
```
# groupby CustomerID
customers = online_rt.groupby(['CustomerID','Country']).sum()

# there is an outlier with negative price
customers = customers[customers.UnitPrice > 0]

# get the value of the index and put in the column Country
customers['Country'] = customers.index.get_level_values(1)

# top three countries
top_countries =  ['Netherlands', 'EIRE', 'Germany']

# filter the dataframe to just select ones in the top_countries
customers = customers[customers['Country'].isin(top_countries)]

##################
# Graph Section #
##################

# creates the FaceGrid
g = sns.FacetGrid(customers, col="Country")

# map over a make a scatterplot
g.map(plt.scatter, "Quantity", "UnitPrice", alpha=1)

# adds legend
g.add_legend()
```

```
C:\Users\msuse\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: Futur
eWarning: is_categorical_dtype is deprecated and will be removed in a futu
re version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\msuse\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWa
rning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[79]: <seaborn.axisgrid.FacetGrid at 0x1d30990bcd0>



Investigate why the previous results look so uninformative.

- 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.

In [80]:
```
#This takes our initial dataframe groups it primarily by 'CustomerID' and s
#It sums all the (non-indexical) columns that have numerical values under e
customers = online_rt.groupby(['CustomerID','Country']).sum().head()

#Here's what it looks like:
customers
```

Out[80]:

CustomerID	Country	InvoiceNo	
12346.0	**United Kingdom**	541431	
12347.0	**Iceland**	537626537626537626537626537626537626537626537626537626...	85116223757147
12348.0	**Finland**	539318539318539318539318539318539318539318539318539318...	84992229518499
12349.0	**Italy**	577609577609577609577609577609577609577609577609577609776...	23112234602156
12350.0	**Norway**	543037543037543037543037543037543037543037543037543037430...	219082241279066

In [81]:
```
online_rt['Revenue'] = online_rt.Quantity * online_rt.UnitPrice
online_rt.head()
```

Out[81]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850.0	United Kingdom

In [82]:
```
price_start = 0
price_end = 50
price_interval = 1

#Creating the buckets to collect the data accordingly
buckets = np.arange(price_start,price_end,price_interval)

#Select the data and sum
revenue_per_price = online_rt.groupby(pd.cut(online_rt.UnitPrice, buckets))
revenue_per_price.head()
```

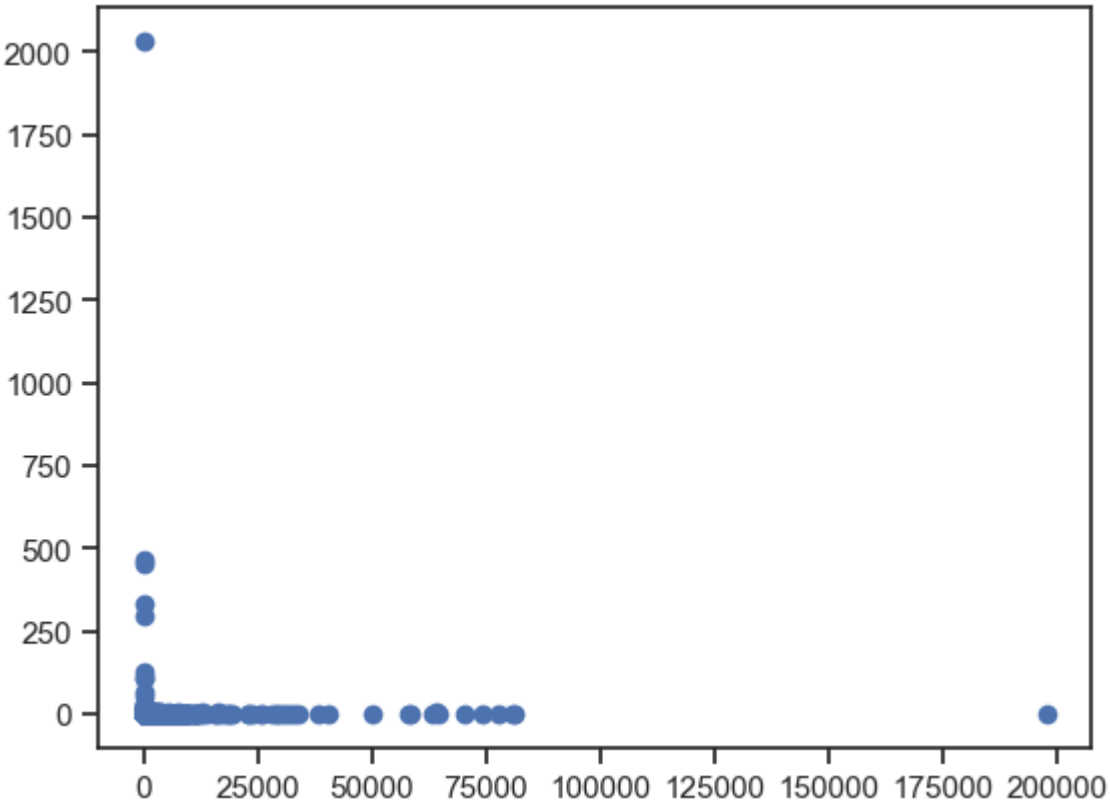
```
C:\Users\msuse\AppData\Local\Temp\ipykernel_7308\156934289.py:9: FutureWar
ning: The default of observed=False is deprecated and will be changed to T
rue in a future version of pandas. Pass observed=False to retain current b
ehavior or observed=True to adopt the future default and silence this warn
ing.
  revenue_per_price = online_rt.groupby(pd.cut(online_rt.UnitPrice, bucket
s)).Revenue.sum()
```

Out[82]:
```
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
```

Plot the data for each CustomerID on a single graph

In [83]:
```
grouped = online_rt.groupby(['CustomerID'])
plottable = grouped[['Quantity','Revenue']].agg('sum')
plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

# map over a make a scatterplot
plt.scatter(plottable.Quantity, plottable.AvgPrice)
plt.plot()
```

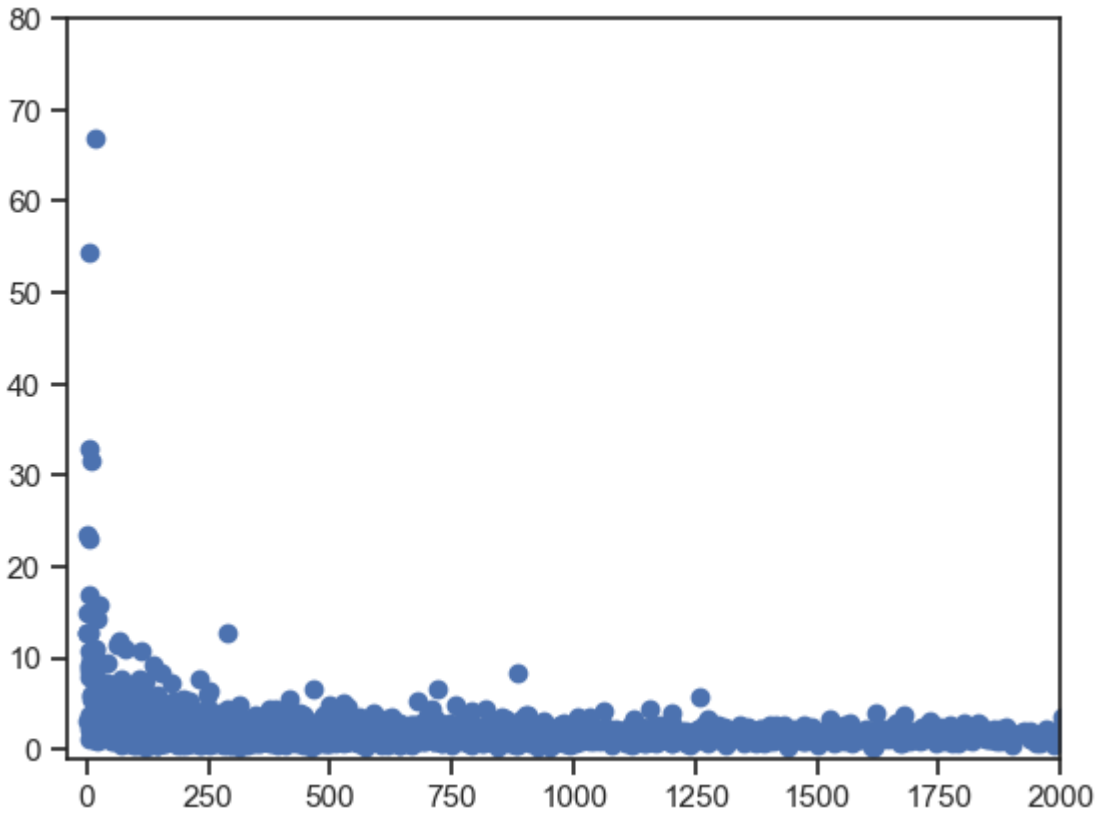
Out[83]:
```
[]
```



- Zoom in so we can see that curve more clearly

In [84]:
```
grouped = online_rt.groupby(['CustomerID','Country'])
plottable = grouped.agg({'Quantity': 'sum',
                         'Revenue': 'sum'})
plottable['AvgPrice'] = plottable.Revenue / plottable.Quantity

# map over a make a scatterplot
plt.scatter(plottable.Quantity, plottable.AvgPrice)

#Zooming in. (I'm starting the axes from a negative value so that
#the dots can be plotted in the graph completely.)
plt.xlim(-40,2000)
plt.ylim(-1,80)

plt.plot()
```

Out[84]: []



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

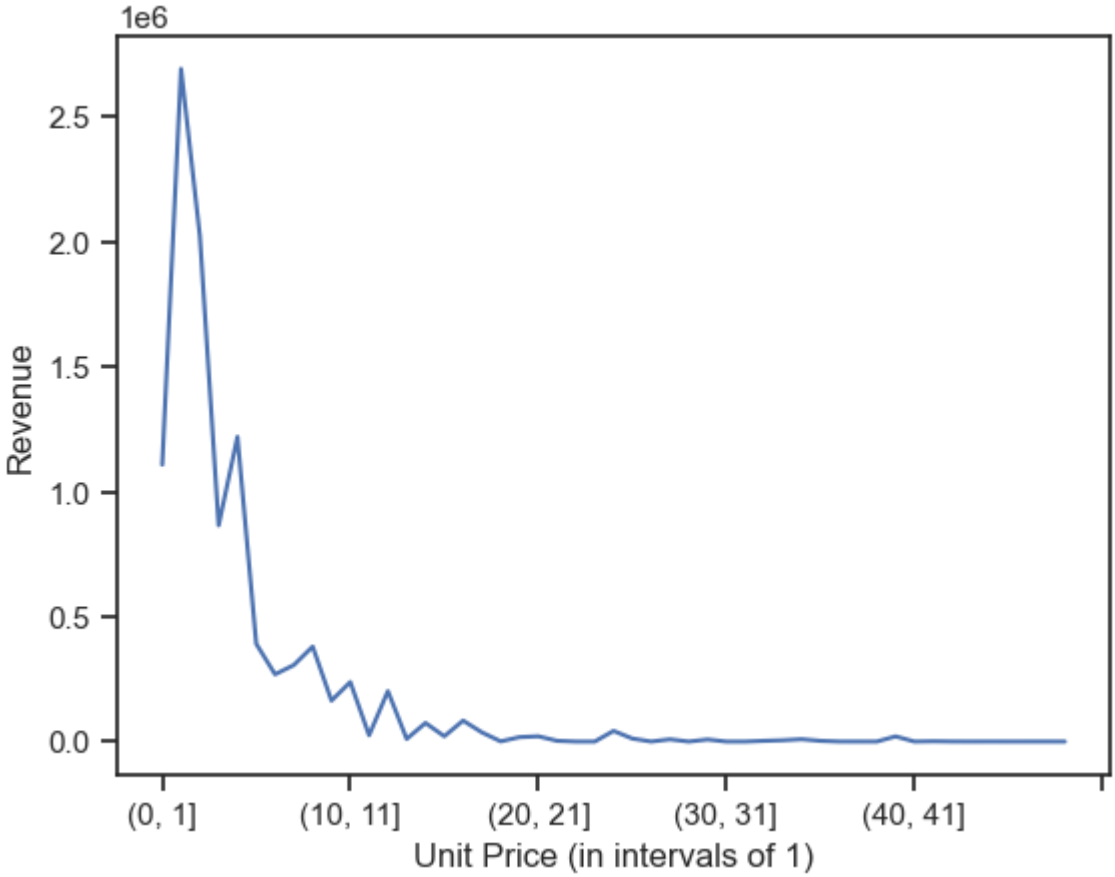
- Group UnitPrice by intervals of 1 for prices [0,50), and sum Quantity and Revenue.

In [85]:
```
#These are the values for the graph.
#They are used both in selecting data from
#the DataFrame and plotting the data so I've assigned
#them to variables to increase consistency and make things easier
#when playing with the variables.
price_start = 0
price_end = 50
price_interval = 1

#Creating the buckets to collect the data accordingly
buckets = np.arange(price_start,price_end,price_interval)

#Select the data and sum
revenue_per_price = online_rt.groupby(pd.cut(online_rt.UnitPrice, buckets))
revenue_per_price.head()
```

```
C:\Users\msuse\AppData\Local\Temp\ipykernel_7308\1044348675.py:14: FutureW
arning: The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this war
ning.
  revenue_per_price = online_rt.groupby(pd.cut(online_rt.UnitPrice, bucket
s)).Revenue.sum()
```

Out[85]:
```
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
```

In [86]:
```
revenue_per_price.plot()
plt.xlabel('Unit Price (in intervals of '+str(price_interval)+')')
plt.ylabel('Revenue')
plt.show()
```



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

In [87]:
```
revenue_per_price.plot()

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

#Even though the data is bucketed in intervals of 1,
#I'll plot ticks a little bit further apart from each other to avoid clutte
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()
```