Day 3: Programming with Python

Advanced Pandas

In this unit we will explore more functionalities of `Pandas` that we can apply to analyse the data of the DataFrames. We will see how we can select data that meet specific conditions and how we can perform statistical analysis on them.

In the first part we will see how to find and deal with missing values and then we will see how to calculate statistics from the data. We will also explore the functionality of `groupby` that allows us to split the data into separate groups to perform computations for better analysis.

For this unit we need `Pandas` .

0. Import pandas and read a file

We will start with importing pandas and reading the file `netherlands-population-2021-06-09_missing.csv`

In [1]:

```
import pandas as pd

df = pd.read_csv("../data/netherlands-population-2021-06-09_missing.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   date        40 non-null     int64
 1   population  36 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 768.0 bytes
```

In [2]:

```
df.head()
```

Out[2]:

	date	population
0	1980	14148415.0
1	1981	14223763.0
2	1982	NaN
3	1983	14365385.0
4	1984	NaN

1. Missing values

Key points:

- **df.isnull()** - Returns a boolean same-sized object indicating if the values are NA

- **df.isnull().sum()** - Returns the number of missing values in the DataFrame per column
- **df.dropna()** - Drops rows which contain missing values
- **df.fillna()** - Fills missing values with the specified method
- **df.fillna(method = 'ffill')** - Fills missing values with forward filling
- **df.fillna(method = 'bfill')**- Fills missing values with backward filling

Pandas gives us functionality to check if there are missing values in the dataset and methods to handle them.

There are different ways to address the missing values. The first is to ignore the missing values and work with the rest of the data if there are enough.

The alternative is to use data imputation. There are different methods that can be used for data imputation. If we are dealing with temporal data we can use the previous or the next value. Another common way is to calculate the mean or median of the existing observations. However, when there are many missing variables, mean or median results can result in a loss of variation in the data.

We will now see how we can handle missing values in Pandas.

The first step is to check if there are any null values in the dataset.

A very useful method is the `isnull()` that returns the boolean value for every data point. With the `sum()` we can also see how many missing values we have per column

In [3]:

```
df.isnull()
```

Out[3]:

	date	population
0	False	False
1	False	False
2	False	True
3	False	False
4	False	True
5	False	False
6	False	False
7	False	False
8	False	False
9	False	False
10	False	False
11	False	False
12	False	False
13	False	False
14	False	False
15	False	True
16	False	False
17	False	False

	date	population
18	False	False
19	False	False
20	False	False
21	False	False
22	False	False
23	False	True
24	False	False
25	False	False
26	False	False
27	False	False
28	False	False
29	False	False
30	False	False
31	False	False
32	False	False
33	False	False
34	False	False
35	False	False
36	False	False
37	False	False
38	False	False
39	False	False

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
date          0
population    4
dtype: int64
```

Remove with dropna

One way of dealing with the missing data is to remove them. Pandas provides the `dropna()` function that can drop all of those rows which have any missing data. Let's print the 15 first rows to see the result

In [5]:

```
df.dropna().head(15)
```

Out[5]:

	date	population
0	1980	14148415.0
1	1981	14223763.0
3	1983	14365385.0

	date	population
5	1985	14513949.0
6	1986	14595755.0
7	1987	14682649.0
8	1988	14774038.0
9	1989	14868655.0
10	1990	14965448.0
11	1991	15064519.0
12	1992	15165862.0
13	1993	15268006.0
14	1994	15369120.0
16	1996	15563255.0
17	1997	15655475.0

We observe that some of the rows (e.g., 2, 4) are now missing.

Fillna function

One of the useful functions that Pandas has for working with missing values is the filling function called `fillna()` . This function takes a number of parameters. You can pass in a single value which is called a scalar value to change all of the missing data to one value.

Let's fill the missing values with 0

In [6]:

```
df1 = df.fillna(0)
df1.head(12)
```

Out[6]:

	date	population
0	1980	14148415.0
1	1981	14223763.0
2	1982	0.0
3	1983	14365385.0
4	1984	0.0
5	1985	14513949.0
6	1986	14595755.0
7	1987	14682649.0
8	1988	14774038.0
9	1989	14868655.0
10	1990	14965448.0
11	1991	15064519.0

The `fillna()` method can also take values that indicate if we want to fill the missing values with the values of the previous or the next item row.

`ffill()` is for forward filling and it updates an `na` value for a particular cell with the value from the previous row. `bfill()` for backward filling which fills the missing values with the next valid value.

We can set the parameter method to `ffil()`. We will make a new dataframe `df1` for that.

In [7]:
```
df1 = df.fillna(method = 'ffill')
df1.head()
```

Out[7]:

	date	population
0	1980	14148415.0
1	1981	14223763.0
2	1982	14223763.0
3	1983	14365385.0
4	1984	14365385.0

Now we can set the parameter method to `bfill`. We will make a new dataframe `df1` for that.

In [8]:
```
df1 = df.fillna(method = 'bfill')
df1.head(10)
```

Out[8]:

	date	population
0	1980	14148415.0
1	1981	14223763.0
2	1982	14365385.0
3	1983	14365385.0
4	1984	14513949.0
5	1985	14513949.0
6	1986	14595755.0
7	1987	14682649.0
8	1988	14774038.0
9	1989	14868655.0

2. Descriptive statistics

Key points:

- **df.describe()** - Generates descriptive statistics of the DataFrame
- **df[col_name].describe()** - Generates descriptive statistics of a column
- **df[col_name].mean()** - Returns the mean of a column
- **df[col_name].sum()** - Returns the sum of a column
- **df.col_name1[df['col_name2'] > < = x].mean()** - Returns the mean of a column based on a condition

There is a large number of methods for computing descriptive statistics and other related operations on Series and DataFrames. Most of these are aggregations like `sum()`, `mean()`, and `quantile()`.

The function describe()

The `describe()` function generates a range of descriptive statistics of a dataset's distribution. With `describe()` we can view the count, mean, std, min, max and quartiles of the numerical data of the DataFrame.

Let's first load some data for that. We will use the `supermarket_sales.csv` file for that. Let's read the file and print some basic statistics

In [9]:
```
df = pd.read_csv('../data/supermarket_sales.csv')
df.head()
```

Out[9]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634

In [10]:
```
df.describe()
```

Out[10]:

	Unit price	Quantity	Tax 5%	Total
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	55.672130	5.510000	15.379369	322.966749
std	26.494628	2.923431	11.708825	245.885335
min	10.080000	1.000000	0.508500	10.678500
25%	32.875000	3.000000	5.924875	124.422375
50%	55.230000	5.000000	12.088000	253.848000
75%	77.935000	8.000000	22.445250	471.350250
max	99.960000	10.000000	49.650000	1042.650000

If we want the descriptive statistics of only one column (e.g., Quantity), we can first get the data of the column and then apply the `describe()` function:

In [11]:
```
df.Quantity.describe()
```

Out[11]:
```
count    1000.000000
mean        5.510000
std         2.923431
min         1.000000
25%         3.000000
50%         5.000000
75%         8.000000
max        10.000000
Name: Quantity, dtype: float64
```

There are also functions that can be used to get a specific statistic of a set of data. For example, if we want to calculate the mean of the Quantity we can do it with the `mean()` function:

In [12]:
```
df.Quantity.mean()
```

Out[12]:
```
5.51
```

Or if we want to get the sum of the Quantity we do it with the `sum()` function:

In [13]:
```
df.Quantity.sum()
```

Out[13]:
```
5510
```

Conditional statistics

We can also put some boolean expressions to apply the function on data that meet a spcific condition. Let's say we want to calculate the mean Quantity bought from customers that have bought at least 5 items.

In [14]:
```
df.Quantity[df['Quantity'] > 6].mean()
```

Out[14]:
```
8.57286432160804
```

3. Group by

Key points:

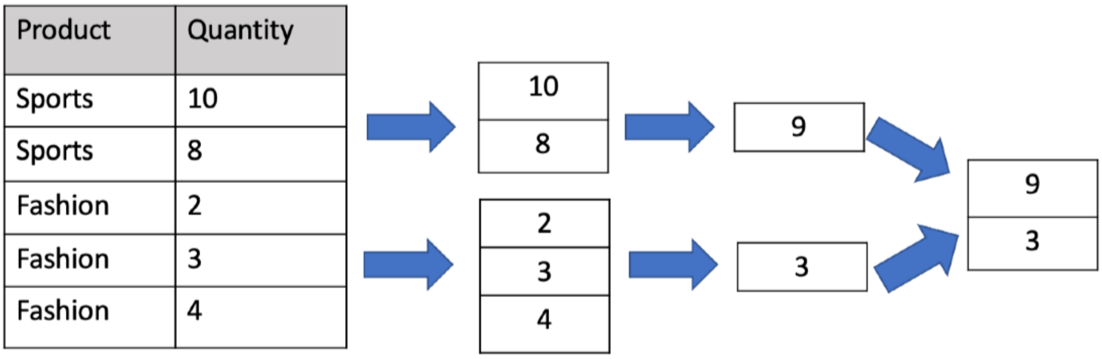
- **df.groupby('col_name')** - Splits the data into groups based on a column
- **df.groupby('col_name').groups** - Returns the groups created after groupby
- **df_product.get_group('group_name').head()** - Prints the first rows of a specific group
- **df.groupby('col_name').size()** - Prints the size of groups
- **df.groupby(['col_name'])['col_name'].mean()** - Returns the mean of a specific column after groupby operation
- **df.groupby(['col_name','col_name'])['col_name'].describe()** - Returns descriptive statistics of of a specific column after groupby operation

The `Groupby` operation that is undoubtedly one of the most powerful functionalities of `Pandas`. Groupby allows adopting a split-apply-combine approach to a data set. This approach is often used to slice and dice data in such a way that a data analyst can answer a specific question.

On a high-level groupby allows to:

1. Split the data based on column(s)/condition(s) into groups;
2. Apply a function/transformation to all the groups and
3. Combine the results into an output

![groupBy.png]



Let's say we are interested to split the data based on their product line. We can do that with the group by operation.

In [15]:

```
df_product = df.groupby('Product line')
df_product.groups
```

Out[15]:

```
{'Electronic accessories': [1, 5, 6, 11, 12, 20, 23, 37, 45, 48, 55, 59, 73, 7
5, 95, 97, 102, 105, 109, 120, 133, 136, 156, 172, 173, 194, 201, 202, 206, 20
9, 210, 217, 220, 222, 227, 228, 231, 238, 246, 248, 256, 258, 259, 260, 290,
291, 292, 295, 296, 303, 304, 305, 308, 314, 317, 329, 335, 338, 340, 346, 34
8, 351, 354, 358, 366, 369, 370, 379, 381, 392, 399, 419, 421, 432, 439, 450,
451, 454, 457, 458, 469, 474, 477, 479, 481, 496, 505, 513, 520, 532, 543, 54
9, 553, 554, 560, 562, 563, 600, 610, 617, ...], 'Fashion accessories': [10, 2
6, 27, 30, 49, 52, 53, 67, 71, 76, 77, 86, 100, 101, 106, 112, 115, 116, 117,
124, 127, 130, 135, 146, 150, 152, 167, 177, 180, 191, 195, 208, 218, 223, 23
0, 233, 237, 239, 242, 247, 251, 255, 261, 262, 275, 277, 278, 300, 309, 311,
323, 332, 336, 345, 350, 352, 356, 365, 371, 373, 375, 378, 388, 390, 391, 40
3, 404, 407, 409, 422, 423, 424, 425, 430, 433, 434, 443, 447, 455, 472, 486,
487, 490, 491, 494, 501, 512, 515, 526, 527, 531, 536, 538, 546, 550, 551, 55
6, 564, 567, 568, ...], 'Food and beverages': [9, 13, 18, 28, 34, 43, 47, 50,
51, 70, 72, 78, 81, 82, 83, 87, 98, 103, 108, 118, 128, 143, 153, 155, 160, 16
2, 164, 168, 171, 174, 176, 178, 181, 185, 192, 199, 211, 219, 221, 224, 240,
249, 250, 267, 288, 293, 302, 312, 315, 316, 320, 326, 327, 331, 333, 339, 34
3, 355, 360, 361, 362, 364, 382, 383, 384, 386, 389, 396, 400, 406, 420, 427,
431, 438, 440, 446, 452, 456, 459, 460, 461, 463, 464, 468, 480, 497, 507, 52
4, 528, 533, 539, 544, 557, 558, 561, 565, 572, 573, 576, 577, ...], 'Health a
nd beauty': [0, 3, 8, 14, 16, 21, 29, 33, 38, 44, 46, 57, 64, 65, 66, 69, 79,
80, 89, 93, 94, 96, 104, 111, 134, 141, 142, 145, 147, 149, 158, 165, 170, 17
9, 183, 196, 198, 203, 205, 226, 232, 234, 236, 241, 271, 274, 283, 284, 285,
294, 301, 313, 318, 319, 321, 322, 328, 341, 342, 349, 387, 394, 395, 398, 41
0, 412, 415, 417, 418, 426, 445, 448, 453, 466, 473, 475, 492, 508, 516, 523,
530, 541, 552, 578, 579, 581, 585, 589, 590, 595, 627, 635, 636, 646, 651, 66
7, 668, 672, 673, 678, ...], 'Home and lifestyle': [2, 7, 19, 22, 25, 39, 40,
41, 54, 56, 58, 61, 74, 90, 99, 113, 114, 119, 123, 125, 137, 144, 148, 157, 1
66, 175, 186, 187, 188, 189, 190, 193, 197, 204, 207, 212, 215, 229, 243, 244,
245, 253, 254, 257, 266, 268, 269, 272, 273, 276, 280, 281, 286, 289, 297, 29
8, 299, 307, 324, 330, 347, 353, 363, 367, 372, 374, 376, 397, 401, 402, 408,
414, 416, 429, 437, 442, 470, 483, 488, 489, 493, 502, 509, 511, 517, 518, 52
1, 522, 534, 535, 537, 540, 545, 555, 559, 570, 591, 599, 605, 622, ...], 'Spo
rts and travel': [4, 15, 17, 24, 31, 32, 35, 36, 42, 60, 62, 63, 68, 84, 85, 8
8, 91, 92, 107, 110, 121, 122, 126, 129, 131, 132, 138, 139, 140, 151, 154, 15
```

```
9, 161, 163, 169, 182, 184, 200, 213, 214, 216, 225, 235, 252, 263, 264, 265,
270, 279, 282, 287, 306, 310, 325, 334, 337, 344, 357, 359, 368, 377, 380, 38
5, 393, 405, 411, 413, 428, 435, 436, 441, 444, 449, 462, 465, 467, 471, 476,
478, 482, 484, 485, 495, 498, 499, 500, 503, 504, 506, 510, 514, 519, 525, 52
9, 542, 547, 548, 566, 569, 571, ...]}
```

We can access one of the groups with the `get_group()` method. Let's see the five first items of the Fashion accessories group:

In [16]:

```
df_product.get_group('Fashion accessories').head()
```

Out[16]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
10	351-62-0822	B	Mandalay	Member	Female	Fashion accessories	14.48	4	2.8960	6(
26	649-29-6775	B	Mandalay	Normal	Male	Fashion accessories	33.52	1	1.6760	3!
27	189-17-4241	A	Yangon	Normal	Female	Fashion accessories	87.67	2	8.7670	18.
30	871-79-8483	B	Mandalay	Normal	Male	Fashion accessories	94.13	5	23.5325	49.
49	574-22-5561	C	Naypyitaw	Member	Female	Fashion accessories	82.63	10	41.3150	86

Let's also see the how many items there are per product line group. We can do that with the `size()`

In [17]:

```
df_product.size()
```

Out[17]:

```
Product line
Electronic accessories    170
Fashion accessories       178
Food and beverages        174
Health and beauty         152
Home and lifestyle        160
Sports and travel         166
dtype: int64
```

Let's say we want to get the mean Quantity per product line. Then we can do it by first grouping by adding the column name after the grouping

In [18]:

```
df_product['Quantity'].mean()
```

Out[18]:

```
Product line
Electronic accessories    5.711765
Fashion accessories       5.067416
Food and beverages        5.471264
Health and beauty         5.618421
Home and lifestyle        5.693750
Sports and travel         5.542169
Name: Quantity, dtype: float64
```

And then we want to get all the general statistics of Quantity per Customer type and per

Gender.

In [19]:
```
df.groupby(['Customer type','Gender'])['Quantity'].describe()
```

Out[19]:

Customer type	Gender	count	mean	std	min	25%	50%	75%	max
Member	Female	261.0	5.716475	2.937175	1.0	3.0	6.0	8.0	10.0
	Male	240.0	5.387500	2.984593	1.0	3.0	5.0	8.0	10.0
Normal	Female	240.0	5.737500	2.836155	1.0	4.0	6.0	8.0	10.0
	Male	259.0	5.204633	2.914914	1.0	3.0	5.0	7.0	10.0

4. Map and apply a function

Key points:

- **df['col_name'].map({})** - Substitutes each value in a Series with another value
- **df['col_name'].apply(function)** - Applies a given function to each item of the given column
- **df['col_name'].apply(lambda)** - Applies a lambda function to each item of the given column

`Map` is used for substituting each value in a series with another value, that may be derived from a function.

Let's say we want to create an additional column to our DataFrame that will contain the value 0 for females and 1 for males. The new column is called `Gender_num`.

In [20]:
```
df['Gender_num'] = df['Gender'].map({'Female':0, 'Male':1})
```

In [21]:
```
df[['Gender',"Gender_num"]].head()
```

Out[21]:

	Gender	Gender_num
0	Female	0
1	Female	0
2	Male	1
3	Male	1
4	Male	1

With the `apply()` function we can apply a function along a row or a column of the DataFrame.

In [22]:
```
def freeItems(x):
    if x <= 5:
        x = x +1
    else:
        x = x + 2
    return x
```

In [23]:
```
df['New_Quantity'] = df['Quantity'].apply(freeItems)
df.loc[0:4, ['Quantity', "New_Quantity"]]
```

Out[23]:

	Quantity	New_Quantity
0	7	9
1	5	6
2	7	9
3	8	10
4	7	9

A `lambda` function is a small function containing a single expression. `Lambda` functions can also act as anonymous functions where they don't require any name. These are very helpful when we have to perform small tasks with less code.

Let's see how we can do the above example without an additional function.

In [24]:
```
df['New_Quantity'] = df['Quantity'].apply(lambda x: x + 1
                                          if x <= 5 else x + 2)
df[['Quantity', "New_Quantity"]].head()
```

Out[24]:

	Quantity	New_Quantity
0	7	9
1	5	6
2	7	9
3	8	10
4	7	9

5. Correlations

Key points:

- **df['column1'].corr(df['column2'])** - Calculates correlation between column1 and column2
- **df1.corr(df2, method='spearman')** - Calculates spearman correlation between column1 and column2
- **df.corr(method ='pearson')** - Calculates pearson correlation among all numerical columns of df

Correlation coefficients quantify the association between variables or features of a dataset. Python has great tools that you can use to calculate them.

`Pearson r correlation` is the most widely used correlation statistic to measure the degree of the relationship between linearly related variables. For example, in the stock market, if we want to measure how two stocks are related to each other, Pearson r correlation is used to measure the degree of relationship between the two.

`Spearman rank correlation` is a non-parametric test that is used to measure the degree of association between two variables. The Spearman rank correlation test does not carry any assumptions about the distribution of the data and is the appropriate correlation analysis when the variables are measured on a scale that is at least ordinal.

Let's say we want to calculate the correlation between Unit price and Quantity

In [25]:
```
df['Quantity'].corr(df['Unit price'])
```

Out[25]: `0.010777564342497298`

Or the correlation between Branch and Total

In [26]:
```
df['Branch'].corr(df['Total'], method='spearman')
```

Out[26]: `0.019624351879191697`

It is also possible to get the correlation among all the numerical columns of a DataFrame

In [27]:
```
df.corr(method ='pearson')
```

Out[27]:

	Unit price	Quantity	Tax 5%	Total	Gender_num	New_Quantity
Unit price	1.000000	0.010778	0.633962	0.633962	0.015445	0.014053
Quantity	0.010778	1.000000	0.705510	0.705510	-0.074258	0.997260
Tax 5%	0.633962	0.705510	1.000000	1.000000	-0.049451	0.705720
Total	0.633962	0.705510	1.000000	1.000000	-0.049451	0.705720
Gender_num	0.015445	-0.074258	-0.049451	-0.049451	1.000000	-0.071323
New_Quantity	0.014053	0.997260	0.705720	0.705720	-0.071323	1.000000

Summary

In this unit, we explored functions that can be used on the data of Pandas Series and DataFrames.

First, we explored ways to deal with missing values in the DataFrames. Next, we worked with group by that splits that data based on values that a column contains. Also, we applied methods on the data with map and apply functions.

Finally, we calculated the correlation between two different variables/features that can also be part of a DataFrame.

Exercises

```
1. Read the file winemag-data_first50k.csv and print some of the
information of the DataFrame to get familiar with it
```

In [28]:
```
df = pd.read_csv("../data/winemag-data_first50k.csv")
df.head()
```

Out[28]:

	Unnamed: 0	country	description	designation	points	price	province	region_1	region_2
0	0	US	This tremendous 100% varietal wine hails from ...	Martha's Vineyard	96	235.0	California	Napa Valley	Napa
1	1	Spain	Ripe aromas of fig, blackberry and cassis are ...	Carodorum Selección Especial Reserva	96	110.0	Northern Spain	Toro	NaN
2	2	US	Mac Watson honors the memory of a wine once ma...	Special Selected Late Harvest	96	90.0	California	Knights Valley	Sonoma
3	3	US	This spent 20 months in 30% new French oak, an...	Reserve	96	65.0	Oregon	Willamette Valley	Willamette Valley
4	4	France	This is the top wine from La Bégude, named aft...	La Brûlade	95	66.0	Provence	Bandol	NaN

In [29]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Unnamed: 0   49999 non-null  int64  
 1   country      49997 non-null  object 
 2   description  49999 non-null  object 
 3   designation  35657 non-null  object 
 4   points       49999 non-null  int64  
 5   price        45430 non-null  float64
 6   province     49997 non-null  object 
 7   region_1     41879 non-null  object 
 8   region_2     19744 non-null  object 
 9   variety      49999 non-null  object 
 10  winery       49999 non-null  object 
dtypes: float64(1), int64(2), object(8)
memory usage: 4.2+ MB
```

2. Print the number of missing values per column

In [30]:

```
df.isnull().sum()
```

Out[30]:

```
Unnamed: 0         0
country            2
```

```
description        0
designation    14342
points             0
price           4569
province           2
region_1        8120
region_2       30255
variety            0
winery             0
dtype: int64
```

3. Drop the rows that have missing values in province and check that they have been dropped

In [31]:
```
df.shape
```

Out[31]:
```
(49999, 11)
```

In [32]:
```
df = df.dropna(subset=["province"])
df.shape
```

Out[32]:
```
(49997, 11)
```

In [33]:
```
df.isnull().sum()
```

Out[33]:
```
Unnamed: 0         0
country            0
description        0
designation    14342
points             0
price           4569
province           0
region_1        8118
region_2       30253
variety            0
winery             0
dtype: int64
```

4. Drop columns designation, region_1 and region_2 and save the new DataFrame to df. Print the columns of the new DataFrame to check whether they have been dropped

In [34]:
```
df=df.drop(columns=['designation', 'region_1','region_2'])
df.columns
```

Out[34]:
```
Index(['Unnamed: 0', 'country', 'description', 'points', 'price', 'province',
       'variety', 'winery'],
      dtype='object')
```

5. Get the first ten rows of price that have null price value

In [35]:
```
df[df.price.isnull()].head(10)
```

Out[35]:

Unnamed: 0	country	description	points	price	province	variety	winery

	Unnamed: 0	country	description	points	price	province	variety	winery
32	32	Italy	Underbrush, scorched earth, menthol and plum s...	90	NaN	Tuscany	Sangiovese	Abbadia Ardenga
56	56	France	Delicious while also young and textured, this ...	90	NaN	Loire Valley	Sauvignon Blanc	Domaine Vacheron
72	72	Italy	This offers aromas of red rose, wild berry, da...	91	NaN	Piedmont	Nebbiolo	Silvano Bolmida
82	82	Italy	Berry, baking spice, dried iris, mint and a hi...	91	NaN	Piedmont	Nebbiolo	Ceste
116	116	Spain	Aromas of brandied cherry and crème de cassis ...	86	NaN	Levante	Monastrell	Casa de la Ermita
242	242	France	A tight and herbaceous wine that is crisp, min...	88	NaN	Bordeaux	Bordeaux-style White Blend	Château Ferran
261	261	France	This fresh and fruity sparkling wine is crisp ...	88	NaN	Loire Valley	Chenin Blanc-Chardonnay	Musset-Roullier
282	282	France	The estate wine from Château du Cèdre is anyth...	92	NaN	Southwest France	Malbec	Château du Cèdre
294	294	France	A ripe, wood-aged wine, it's richly smoky and ...	91	NaN	Southwest France	Gros and Petit Manseng	Lionel Osmin & Cie
323	323	Spain	Ripe pure black-fruit aromas are touched up by...	94	NaN	Northern Spain	Red Blend	Matarromera

```
6. Find the mean price for the wines that have at least 87 points
```

In [36]:

```
df.price[df['points'] > 86].mean()
```

Out[36]: `40.898689902739854`

7. Group the DataFrame by points and return the mean price per group

In [37]:
```
points = df.groupby('points')
points.groups

points['price'].mean()
```

Out[37]:
```
points
80      15.977273
81      17.862191
82      19.245392
83      18.125000
84      18.797662
85      19.755679
86      21.989537
87      24.748044
88      28.221291
89      32.527373
90      37.225691
91      43.488601
92      52.027128
93      66.374562
94      80.196429
95     107.958633
96     137.594737
97     231.738318
98     273.225000
99     333.437500
100    532.571429
Name: price, dtype: float64
```

8. Fill the missing values of prices with the mean value per group. Save the result to the price column

In [38]:
```
df['price']=df.groupby('points')['price'].apply(lambda x:x.fillna(x.mean()))
```

9. Print one of the rows that had missing value in price and check the row again

In [39]:
```
df.loc[32]
```

Out[39]:
```
Unnamed: 0                                                    32
country                                                    Italy
description    Underbrush, scorched earth, menthol and plum s...
points                                                        90
price                                                  37.225691
province                                                 Tuscany
variety                                               Sangiovese
winery                                           Abbadia Ardenga
Name: 32, dtype: object
```

10. Group by country and check the size per group

In [40]:
```
df.groupby('country').size()
```

Out[40]:
```
country
Albania                      2
Argentina                 1520
Australia                 1015
```

```
Austria                      1269
Bosnia and Herzegovina          2
Brazil                          7
Bulgaria                       27
Canada                         92
Chile                        1561
China                           1
Croatia                        22
Cyprus                          3
England                         9
France                       7738
Georgia                        11
Germany                      1069
Greece                        195
Hungary                        48
India                           7
Israel                        279
Italy                        8637
Japan                           1
Lebanon                        14
Lithuania                       4
Macedonia                       5
Mexico                         29
Moldova                        25
Montenegro                      1
Morocco                        11
New Zealand                   550
Portugal                     2166
Romania                        33
Serbia                         12
Slovenia                       53
South Africa                  516
South Korea                     2
Spain                        2665
Switzerland                     2
Turkey                         45
US                          20321
Ukraine                         5
Uruguay                        23
dtype: int64
```

11. Calculate the correlation between the price and points

In [41]:
```
df['points'].corr(df['price'])
```

Out[41]:
```
0.43969208119304333
```

12. Calculate the correlation between price and points per country only if the country has 50 items

In [43]:
```
df.groupby('country')[['price','points']].corr(min_periods=50)
```

Out[43]:

		price	points
country			
Albania	**price**	NaN	NaN
	points	NaN	NaN
Argentina	**price**	1.000000	0.555047
	points	0.555047	1.000000
Australia	**price**	1.000000	0.506596

		price	points
country			
...	**...**	...	...
US	**points**	0.424138	1.000000
Ukraine	**price**	NaN	NaN
	points	NaN	NaN
Uruguay	**price**	NaN	NaN
	points	NaN	NaN

84 rows × 2 columns

13. Bonus:

Another widely used statistical test is the t test that tells you how significant the differences between groups are. In other words it lets you know if those differences (measured in means) could have happened by chance. For the t-test we need the `scipy` library. SciPy is an open-source Python library which is used to solve scientific and mathematical problems.

We want to see if there is any statistical difference in the wine price between Argentina and Chile and then between Italy and Chile. Then:

In [44]:
```
import scipy.stats as stats
```

In [45]:
```
stats.ttest_ind(df[df['country'] == 'Argentina'].price,
                df[df['country'] == 'Chile'].price)
```

Out[45]:
```
Ttest_indResult(statistic=-0.37878485749090124, pvalue=0.7048737494373603)
```

In [46]:
```
stats.ttest_ind(df[df['country'] == 'Italy'].price,
                df[df['country'] == 'Chile'].price)
```

Out[46]:
```
Ttest_indResult(statistic=16.71967707747564, pvalue=6.270455620734883e-62)
```

In [ ]:
```

```