Chapter 5 Grouping and Aggregating

In [1]:
```
import pandas as pd

# Read car_stocks dataset
car_stks = pd.read_csv("./datasets/car_stocks.csv")

# Print the dataframe
car_stks.head()
```

Out[1]:

	Symbol	Date	Open	High	Low	Close	Adj Close	Volume
0	RIVN	2021-11-10	106.750000	119.459999	95.199997	100.730003	100.730003	103679500
1	RIVN	2021-11-11	114.625000	125.000000	108.010002	122.989998	122.989998	83668200
2	RIVN	2021-11-12	128.645004	135.199997	125.250000	129.949997	129.949997	50437500
3	RIVN	2021-11-15	130.800003	152.529999	127.510002	149.360001	149.360001	64982300
4	RIVN	2021-11-16	163.800003	179.470001	153.779999	172.009995	172.009995	94036600

Find the average closing price of all possible values of column 'Symbol'

In [2]:
```
# Find the different possible values of Symbol in the dataframe

# Firstly, we can find the different possible values and their counts of a column u
# Dataframe.column.value_counts() method
car_stks.Symbol.value_counts()
```

Out[2]:
```
RIVN    13
LCID    13
GM      13
Name: Symbol, dtype: int64
```

In [3]:
```
# Find average closing price of RIVN
RIVN = car_stks["Symbol"] == "RIVN"
car_stks[RIVN].Close.mean()
```

Out[3]:
```
127.52307653846154
```

In [4]:
```
# Find average closing price of LCID
LCID = car_stks["Symbol"] == "LCID"
car_stks[LCID].Close.mean()
```

Out[4]:
```
49.82923061538462
```

In [5]:
```
# Find average closing price of GM
GM = car_stks["Symbol"] == "GM"
car_stks[GM].Close.mean()
```

Out[5]:
```
62.16461546153845
```

For a large dataset with numerous types of possible values, individually finding the average values for each type can be impractical. Is there any better alternative to this? The answer is **Grouping**.

Group by column

To group by values in a column we can make use of the following method:

Dataframe.groupby(by=Column): Returns a DataframeGroupBy object that rearranges the order of records to group them together based on the column passed. We can then apply various Dataframe methods to analyze these groups further.

Ex: df.groupby(by="age") or df.groupby("age")

In [6]:
```
# Grouping by the 'Symbol' column and finding the mean closing price for each group
grouped_data = car_stks.groupby("Symbol")
grouped_data["Close"].mean()

# This would calculate the mean closing price for each unique value in the 'Symbol'
```

Out[6]:
```
Symbol
GM       62.164615
LCID     49.829231
RIVN    127.523077
Name: Close, dtype: float64
```

In [7]:
```
# Read titanic dataset
titanic = pd.read_csv("./datasets/titanic.csv")

# Print the dataframe
titanic.head()
```

Out[7]:

	pclass	survived	name	gender	age	sibsp	parch	ticket	fare	cabin	em
0	1	1	Allen, Miss. Elisabeth Walton	female	29	0	0	24160	211.3375	B5	
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.55	C22 C26	
2	1	0	Allison, Miss. Helen Loraine	female	2	1	2	113781	151.55	C22 C26	
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30	1	2	113781	151.55	C22 C26	
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	1	2	113781	151.55	C22 C26	

In [8]:
```
# Find the datatype of age
titanic.age.dtype

# dtype('O') represents Python Object str
```

Out[8]:
```
dtype('O')
```

In [9]:
```
# Convert datatype from Python Object str to float64

# Replace '?' with None using replace method, in place
titanic.age.replace(['?'], [None], inplace=True)

# Set the datatype from Python Object to float64
titanic.age = titanic.age.astype("float")

# Confirm change in datatype
titanic.age.dtype
```

Out[9]:
```
dtype('float64')
```

In [10]:
```
# Create a shorter version of titanic dataframe including only the columns pclass,
tnc = titanic[["pclass", "survived", "gender", "age"]]

# Print the dataframe
tnc.head()
```

Out[10]:

	pclass	survived	gender	age
0	1	1	female	29.0000
1	1	1	male	0.9167
2	1	0	female	2.0000
3	1	0	male	30.0000
4	1	0	female	25.0000

Exploring various properties/methods on groups

As discussed above, we can group the records based on a column using the method **Dataframe.groupby(by=Column)** or simply **Dataframe.groupby(Column)**. It returns a DataframeGroupBy object that can be later used to perform data analysis on these groups.

1. **DataframeGroupBy.ngroups:** Returns the number of unique groups formed after the groupby() operation. It provides insight into the diversity of groups within the dataset.

Ex: df.groupby("age").ngroups

2. **DataframeGroupBy.groups:** Returns a dictionary where the keys are the unique group labels and the values are arrays of index labels corresponding to each group. It provides a convenient way to access the indexes of records within each group.

Ex: df.groupby("age").groups

3. **DataframeGroupBy.get_group(group_name):** Returns a Dataframe containing records of specified group.

Ex: df.groupby("age").get_group(14.0)

In [11]:
```
# Create a DataframeGroupBy Object on the column "gender"
gender_gbo = tnc.groupby(by="gender")

# Create a DataframeGroupBy Object on the column "age"
age_gbo = tnc.groupby(by="age")

# Create a DataframeGroupBy Object on the column "survived"
surv_gbo = tnc.groupby(by="survived")
```

In [12]:
```
# Print the various groups (genders)
gender_gbo.ngroups

# Two groups are formed (male and female)
```

Out[12]: 2

In [13]:
```
# Print the various groups (age)
```

```
age_gbo.ngroups

# 98 groups are formed
```

Out[13]: 98

In [14]:
```
# Print the indexes of each group (genders)
gender_gbo.groups

# Returns a dictionary with group label (female, male) as keys and the values are a
```

Out[14]:
```
{'female': [0, 2, 4, 6, 8, 11, 12, 13, 17, 18, 21, 23, 24, 27, 28, 32, 33, 35, 36,
41, 42, 43, 44, 48, 50, 55, 57, 59, 61, 63, 65, 66, 67, 69, 72, 73, 76, 78, 79, 8
2, 83, 85, 88, 90, 92, 95, 97, 98, 99, 102, 103, 104, 105, 107, 108, 111, 112, 11
3, 116, 117, 121, 122, 124, 127, 129, 130, 131, 134, 137, 139, 141, 144, 146, 149,
153, 155, 159, 160, 161, 163, 167, 168, 169, 176, 178, 180, 181, 182, 186, 187, 18
8, 190, 192, 193, 195, 198, 199, 204, 207, 208, ...], 'male': [1, 3, 5, 7, 9, 10,
14, 15, 16, 19, 20, 22, 25, 26, 29, 30, 31, 34, 37, 38, 39, 40, 45, 46, 47, 49, 5
1, 52, 53, 54, 56, 58, 60, 62, 64, 68, 70, 71, 74, 75, 77, 80, 81, 84, 86, 87, 89,
91, 93, 94, 96, 100, 101, 106, 109, 110, 114, 115, 118, 119, 120, 123, 125, 126, 1
28, 132, 133, 135, 136, 138, 140, 142, 143, 145, 147, 148, 150, 151, 152, 154, 15
6, 157, 158, 162, 164, 165, 166, 170, 171, 172, 173, 174, 175, 177, 179, 183, 184,
185, 189, 191, ...]}
```

In [15]:
```
# Print the indexes of each group (survived)
surv_gbo.groups

# Returns a dictionary with group labels (0, 1) as keys and the values are arrays o
```

Out[15]:
```
{0: [2, 3, 4, 7, 9, 10, 15, 16, 19, 25, 30, 34, 38, 39, 40, 45, 46, 51, 52, 53, 5
8, 60, 62, 70, 71, 74, 75, 77, 80, 81, 84, 89, 96, 101, 105, 106, 110, 114, 115, 1
18, 125, 126, 128, 132, 135, 138, 142, 147, 148, 150, 154, 156, 157, 158, 162, 16
6, 169, 171, 172, 173, 174, 175, 179, 184, 185, 189, 191, 194, 197, 200, 201, 203,
205, 206, 210, 211, 212, 215, 217, 221, 222, 223, 224, 225, 226, 228, 232, 234, 23
6, 237, 239, 241, 243, 244, 246, 248, 252, 262, 266, 267, ...], 1: [0, 1, 5, 6, 8,
11, 12, 13, 14, 17, 18, 20, 21, 22, 23, 24, 26, 27, 28, 29, 31, 32, 33, 35, 36, 3
7, 41, 42, 43, 44, 47, 48, 49, 50, 54, 55, 56, 57, 59, 61, 63, 64, 65, 66, 67, 68,
69, 72, 73, 76, 78, 79, 82, 83, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 97, 98, 9
9, 100, 102, 103, 104, 107, 108, 109, 111, 112, 113, 116, 117, 119, 120, 121, 122,
123, 124, 127, 129, 130, 131, 133, 134, 136, 137, 139, 140, 141, 143, 144, 145, 14
6, ...]}
```

In [16]:
```
# Print male passenger records
gender_gbo.get_group("male")
```

Out[16]:

	pclass	survived	gender	age
1	1	1	male	0.9167
3	1	0	male	30.0000
5	1	1	male	48.0000
7	1	0	male	39.0000
9	1	0	male	71.0000
...	...	...	...	...
1302	3	0	male	NaN
1303	3	0	male	NaN
1306	3	0	male	26.5000
1307	3	0	male	27.0000
1308	3	0	male	29.0000

843 rows × 4 columns

In [17]:
```
# Print records of who survived
surv_gbo.get_group(1)
```

Out[17]:

	pclass	survived	gender	age
0	1	1	female	29.0000
1	1	1	male	0.9167
5	1	1	male	48.0000
6	1	1	female	63.0000
8	1	1	female	53.0000
...	...	...	...	...
1261	3	1	female	63.0000
1277	3	1	male	22.0000
1286	3	1	female	38.0000
1290	3	1	female	47.0000
1300	3	1	female	15.0000

500 rows × 4 columns

Aggregation methods

The aggregation methods provide versatile tools for summarizing and analyzing data within grouped contexts, offering insights into various statistical properties of the grouped data.

Method Name	Description
count()	Counts the number of non-null values in each group.
sum()	Computes the sum of values in each group.
mean()	Calculates the mean (average) of values in each group.
median()	Calculates the median (middle value) of values in each group.
min()	Finds the minimum value in each group.
max()	Finds the maximum value in each group.
std()	Computes the standard deviation of values in each group.
var()	Computes the variance of values in each group.
first()	Retrieves the first value in each group.
last()	Retrieves the last value in each group.
agg() or aggregate()	Allows for applying custom or multiple aggregation functions simultaneously.

In [18]:
```
# Print titanic dataframe
tnc.head()
```

Out[18]:

	pclass	survived	gender	age
0	1	1	female	29.0000
1	1	1	male	0.9167
2	1	0	female	2.0000
3	1	0	male	30.0000
4	1	0	female	25.0000

Count of all columns of passengers grouped by gender

In [19]:
```
gender_gbo.count()
```

Out[19]:

gender	pclass	survived	age
female	466	466	388
male	843	843	658

Count of ages of passengers grouped by gender

In [20]:
```
gender_gbo["age"].count()
```

Out[20]:
```
gender
female    388
male      658
Name: age, dtype: int64
```

Sum of all columns of passengers grouped by gender

In [21]:
```
gender_gbo.sum()
```

Out[21]:

gender	pclass	survived	age
female	1004	339	11130.5834
male	2000	161	20125.0833

Sum of ages of passengers grouped by gender

In [22]:
```
gender_gbo["age"].sum()
```

Out[22]:
```
gender
female    11130.5834
male      20125.0833
Name: age, dtype: float64
```

Mean of all columns of passengers grouped by gender

In [23]:
```
gender_gbo.mean()
```

Out[23]:

gender	pclass	survived	age
female	2.154506	0.727468	28.687071
male	2.372479	0.190985	30.585233

Mean of ages of passengers grouped by gender

In [24]:
```
gender_gbo["age"].mean()
```

Out[24]:
```
gender
female    28.687071
male      30.585233
Name: age, dtype: float64
```

Median of all columns of passengers grouped by gender

In [25]:
```
gender_gbo.median()
```

Out[25]:

gender	pclass	survived	age
female	2.0	1.0	27.0
male	3.0	0.0	28.0

Median of ages of passengers grouped by gender

In [26]:
```
gender_gbo["age"].median()
```

Out[26]:
```
gender
female    27.0
male      28.0
Name: age, dtype: float64
```

Min of all columns of passengers grouped by gender

In [27]:
```
gender_gbo.min()
```

Out[27]:

gender	pclass	survived	age
female	1	0	0.1667
male	1	0	0.3333

Min of ages of passengers grouped by gender

In [28]:
```
gender_gbo["age"].min()
```

Out[28]:
```
gender
female    0.1667
male      0.3333
Name: age, dtype: float64
```

Max of all columns of passengers grouped by gender

In [29]:
```
gender_gbo.max()
```

Out[29]:

gender	pclass	survived	age
female	3	1	76.0
male	3	1	80.0

Max of ages of passengers grouped by gender

In [30]:
```
gender_gbo["age"].max()
```

Out[30]:
```
gender
female    76.0
male      80.0
Name: age, dtype: float64
```

Standard Deviation of all columns of passengers grouped by gender

In [31]:
```
gender_gbo.std()
```

Out[31]:

gender	pclass	survived	age
female	0.866181	0.445741	14.576995
male	0.811908	0.393310	14.280571

Standard Deviation of ages of passengers grouped by gender

In [32]:
```
gender_gbo["age"].std()
```

Out[32]:
```
gender
female    14.576995
male      14.280571
Name: age, dtype: float64
```

Variance of all columns of passengers grouped by gender

In [33]:
```
gender_gbo.var()
```

Out[33]:

gender	pclass	survived	age
female	0.750270	0.198685	212.488788
male	0.659194	0.154693	203.934709

Variance of ages of passengers grouped by gender

In [34]:
```
gender_gbo["age"].var()
```

Out[34]:
```
gender
female    212.488788
male      203.934709
Name: age, dtype: float64
```

First records of all columns of passengers grouped by gender

In [35]:
```
gender_gbo.first()
```

Out[35]:

gender	pclass	survived	age
female	1	1	29.0000
male	1	1	0.9167

First record of column "age" passengers grouped by gender

In [36]:
```
gender_gbo["age"].first()
```

Out[36]:
```
gender
female    29.0000
male       0.9167
Name: age, dtype: float64
```

Last records of all columns passengers grouped by gender

In [37]:
```
gender_gbo.last()
```

Out[37]:

gender	pclass	survived	age
female	3	0	14.5
male	3	0	29.0

Last record of column "age" passengers grouped by gender

In [38]:
```
gender_gbo["age"].last()
```

Out[38]:
```
gender
female    14.5
male      29.0
Name: age, dtype: float64
```

Multiple aggregate functions combined using agg() method

In [39]:
```
gender_gbo["age"].agg(["min", "max", "count", "mean", "median", "std", "var", "firs
```

Out[39]:

gender	min	max	count	mean	median	std	var	first	last
female	0.1667	76.0	388	28.687071	27.0	14.576995	212.488788	29.0000	14.5
male	0.3333	80.0	658	30.585233	28.0	14.280571	203.934709	0.9167	29.0