Финансовый университет

Тема 7. Методы группировки данных

Лекция

19 октября 2021 года
ВМ/451 (ул. Верхняя Масловка, д. 15)

Студенты: Поток:УЦИ20-1
Преподаватель: Смирнов Михаил Викторович, доцент Департамента Анализа данных и машинного обучения Финансового университета при Правительстве Российской Федерации

Группировка данных

Данные можно представить как совокупность единичных наблюдений. Например, в наборе данных *FinancialSample3.csv* единичное наблюдение — это сделка по продаже конкретного товара.

In [1]:

```
import pandas as pd
fs=pd.read_csv("../Data/FinancialSample3.csv", sep=';')
fs.head()
```

Out[1]:

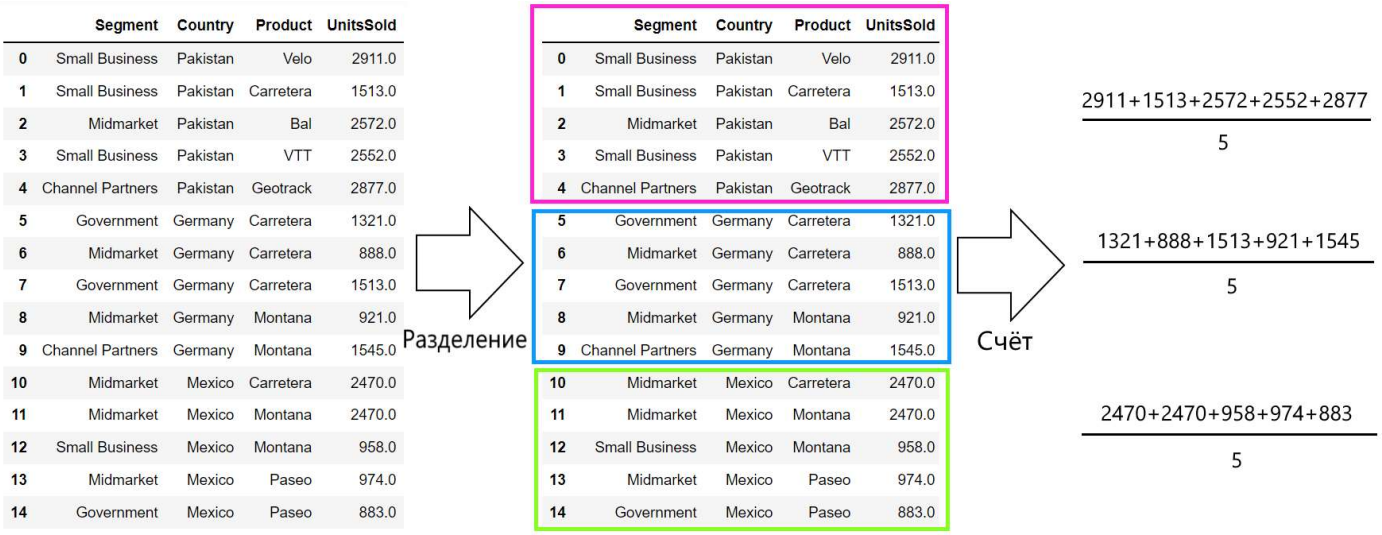
	Id	Segment	Country	Product	DiscountBand	UnitsSold	ManufacturingPrice	SalePrice
0	1	Government	Canada	Carretera	None	1618.5	3.0	20.0
1	2	Government	Germany	Carretera	None	1321.0	3.0	20.0
2	3	Midmarket	France	Carretera	None	2178.0	3.0	15.0
3	4	Midmarket	Germany	Carretera	None	888.0	3.0	15.0
4	5	Midmarket	Mexico	Carretera	None	2470.0	3.0	15.0

У каждого наблюдения есть какие-то атрибуты. Они могут быть разных типов:

- категориальные — например, название фирмы-производителя товара (признак *Product*);
- численные — например, размер партии товара в штуках (признак *UnitsSold*).

Решение задачи группировки предполагает разделение данных по некоторому признаку (атрибуту), после чего к каждому элементу этих разделенных данных мы можем применить агрегирующую операцию. Это такая операция, которая позволяет вычислить какой-либо показатель.

Затем мы можем оценить, как отличаются эти показатели в зависимости от признака, по которому было осуществлено разделение. Такое разделение мы можем назвать группировкой данных.



	Segment	Country	Product	UnitsSold
0	Small Business	Pakistan	Velo	2911.0
1	Small Business	Pakistan	Carretera	1513.0
2	Midmarket	Pakistan	Bal	2572.0
3	Small Business	Pakistan	VTT	2552.0
4	Channel Partners	Pakistan	Geotrack	2877.0
5	Government	Germany	Carretera	1321.0
6	Midmarket	Germany	Carretera	888.0
7	Government	Germany	Carretera	1513.0
8	Midmarket	Germany	Montana	921.0
9	Channel Partners	Germany	Montana	1545.0
10	Midmarket	Mexico	Carretera	2470.0
11	Midmarket	Mexico	Montana	2470.0
12	Small Business	Mexico	Montana	958.0
13	Midmarket	Mexico	Paseo	974.0
14	Government	Mexico	Paseo	883.0

Возьмем информацию о количестве (UnitsSold) проданного товара в пяти первых сделках для стран Пакистан, Германия, Мексика.

In [2]:

```
selected_columns=["Segment","Country","Product","UnitsSold"]
fs_pakistan = fs[fs["Country"]=="Pakistan"][selected_columns].head()
fs_germany = fs[fs["Country"]=="Germany"][selected_columns].head()
fs_mexico = fs[fs["Country"]=="Mexico"][selected_columns].head()
```

Найдем среднее значение количества товара в каждой группе

In [3]:

```
print(fs_pakistan["UnitsSold"].mean())
print(fs_germany["UnitsSold"].mean())
print(fs_mexico["UnitsSold"].mean())
```

```
2485.0
1237.6
1551.0
```

Объединим три этих таблицы в одну. Параметр `ignore_index=True` означает, что после объединения индекс надо перестроить.

In [4]:

```
deals = pd.concat([fs_pakistan, fs_germany, fs_mexico], axis=0, ignore_index=True)
deals
```

Out[4]:

	Segment	Country	Product	UnitsSold
0	Small Business	Pakistan	Velo	2911.0
1	Small Business	Pakistan	Carretera	1513.0
2	Midmarket	Pakistan	Bal	2572.0
3	Small Business	Pakistan	VTT	2552.0
4	Channel Partners	Pakistan	Geotrack	2877.0
5	Government	Germany	Carretera	1321.0
6	Midmarket	Germany	Carretera	888.0
7	Government	Germany	Carretera	1513.0
8	Midmarket	Germany	Montana	921.0
9	Channel Partners	Germany	Montana	1545.0
10	Midmarket	Mexico	Carretera	2470.0
11	Midmarket	Mexico	Montana	2470.0
12	Small Business	Mexico	Montana	958.0
13	Midmarket	Mexico	Paseo	974.0
14	Government	Mexico	Paseo	883.0

Применим функцию *groupby()*

In [5]:

```
deals.groupby(by="Country")["UnitsSold"].mean()
```

Out[5]:

```
Country
Germany     1237.6
Mexico      1551.0
Pakistan    2485.0
Name: UnitsSold, dtype: float64
```

Функция *groupby()*

Ранее мы уже рассмотрели простейшую группировку с помощью функции *value_counts()*, которая возвращает серию, содержащую количество уникальных значений в группах. Кратко вспомним, как работает *value_counts()* на примере. Пусть имеются сведения об успеваимости студентов: имя студента *Name*, учебная группа *Group*, оценка в баллах в первом *Mark_1* и втором *Mark_2* семестрах, а также признак *F_origin*, обозначающий, прибыл студент из другого города или страны *True* или нет *False*.

In [6]:

```
students=pd.DataFrame({'Name':
            ['Иван','Петр','Мария','Василий','Анна','Василиса','Дарья','Николай','Вероника'
            'Group':['У1','У1','У2','У2','У1','У2','У2','У2','У1','У1'],
            'Mark_1':[19,18,17,21,24,18,22,19,18,17],
            'Mark_2':[18,16,17,18,15,20,15,14,19,20],
            'F_origin':[False, False, True, True, False, False, False, True, True, True]})
```

Распечатаем датафрейм с данными о студентах, предварительно добавив столбец - суммарную оценку *Mark_kurs* по итогам двух семестров.

In [7]:

```
students['Mark_kurs'] = students['Mark_1'] + students['Mark_2']
students
```

Out[7]:

	Name	Group	Mark_1	Mark_2	F_origin	Mark_kurs
0	Иван	У1	19	18	False	37
1	Петр	У1	18	16	False	34
2	Мария	У2	17	17	True	34
3	Василий	У2	21	18	True	39
4	Анна	У1	24	15	False	39
5	Василиса	У2	18	20	False	38
6	Дарья	У2	22	15	False	37
7	Николай	У2	19	14	True	33
8	Вероника	У1	18	19	True	37
9	Майкл	У1	17	20	True	37

Применим *value_counts()* к столбцам *Group* и *Mark_1*

In [8]:

```
students['Group'].value_counts()
```

Out[8]:

```
У1    5
У2    5
Name: Group, dtype: int64
```

Результат: уникальными значениями столбца *Group* являются У1 и У2, которые встречаются 4 и 5 раз соответственно. Применим *value_counts()* к численному столбцу *Mark_1*.

In [9]:

```
students['Mark_1'].value_counts()
```

Out[9]:

```
18    3
19    2
17    2
24    1
22    1
21    1
Name: Mark_1, dtype: int64
```

Функция *value_counts()* сгруппировала студентов по группам и по оценкам. Предположим теперь, что нас интересует не количество студентов, а сумма баллов в группе студентов в первом семестре. Это можно узнать с помощью известного нам уже способа отбора по условию и применения статистической функции суммы.

In [10]:

```
print(students[students['Group']=='У1']['Mark_1'].sum())
print(students[students['Group']=='У2']['Mark_1'].sum())
```

```
96
97
```

Для таких целей удобнее применять функцию *groupby()*

In [11]:

```
students.groupby(['Group'])['Mark_1'].sum()
```

Out[11]:

```
Group
У1    96
У2    97
Name: Mark_1, dtype: int64
```

Рассмотрим более подробно работу функции *groupby()*. Выполним команду
`students.groupby(['Group'])`

In [12]:

```
students.groupby(['Group'])
```

Out[12]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001E2B7CE9A08>
```

Мы получили объект, который хранит сведения о том, какие строки относятся к каким значениям группируемого столбца. Чтобы распечатать эти сведения выполним эту же команду с атрибутом
`.groups`

In [13]:

```
students.groupby(['Group']).groups
```

Out[13]:

```
{'У1': Int64Index([0, 1, 4, 8, 9], dtype='int64'),
 'У2': Int64Index([2, 3, 5, 6, 7], dtype='int64')}
```

Теперь мы узнали, что к группе У1 относятся строки [0, 1, 4, 8, 9], а к У2 [2, 3, 5, 6, 7]. Эту информацию использует *pandas* для того, чтобы разбить данные на подгруппы и в каждой подгруппе рассчитать что-либо для нас.

Для того, чтобы провести суммирование (или выполнить другую агрегирующую функцию) по нескольким столбцам, их надо явно указать. Проведем суммирование баллов в каждой учебной группе отдельно в первом и во втором семестрах.

In [14]:

```
students_groupped = students.groupby(['Group'])[['Mark_1','Mark_2']].sum()
students_groupped
```

Out[14]:

Group	Mark_1	Mark_2
У1	96	88
У2	97	84

Объект *studens_groupped* - это *DataFrame*.

In [15]:

```
type(students_groupped)
```

Out[15]:

```
pandas.core.frame.DataFrame
```

Как известно, к элементам таблицы *pandas* можно получить доступ с помощью `.loc[]` и `iloc[]`. Например, чтобы получить сумму баллов студентов У2 во втором семестре введем команду

In [16]:

```
students_groupped.loc['У2','Mark_2']
```

Out[16]:

```
84
```

Если не уточнять, по какому столбцу проводится суммирование (или другая агрегирующая функция), то получим сумму по всем числовым столбцам

In [17]:

```
students_groupped = students.groupby(['Group']).sum()
students_groupped
```

Out[17]:

Group	Mark_1	Mark_2	F_origin	Mark_kurs
У1	96	88	2.0	184
У2	97	84	3.0	181

Студенты какой группы получили в сумме больше баллов? Для ответа отсортируем с помощью `.sort_values()` таблицу сгруппированных значений по убыванию суммы баллов.

In [18]:

```
scores=students_groupped.sort_values('Mark_kurs',ascending=False)
scores
```

Out[18]:

Group	Mark_1	Mark_2	F_origin	Mark_kurs
У1	96	88	2.0	184
У2	97	84	3.0	181

Наибольшее число баллов получили студенты первой группы. После сортировки по убыванию по столбцу *Mark_kurs* это значение содержится в первой строке столбца *Mark_kurs*. Отправим его на печать.

In [19]:

```
scores["Mark_kurs"].iloc[0]
```

Out[19]:

184

Выполнив команду

```
students.groupby(['Group'])[['Mark_1','Mark_2']].sum()
```

мы провели агрегирование сразу по двум столбцам. Но что, если мы хотим группировать также более чем по одному столбцу. Укажем второй параметр группировки - *F_origin* и найдем средний балл отдельно для местных и призжих студентов в каждой группе.

In [20]:

```
students.groupby(['Group', 'F_origin']).mean()
```

Out[20]:

Group	F_origin	Mark_1	Mark_2	Mark_kurs
У1	False	20.333333	16.333333	36.666667
	True	17.500000	19.500000	37.000000
У2	False	20.000000	17.500000	37.500000
	True	19.000000	16.333333	35.333333

Другие агрегирующие функции

Еще раз распечатаем набор данных.

In [21]:

```
fs.head()
```

Out[21]:

	Id	Segment	Country	Product	DiscountBand	UnitsSold	ManufacturingPrice	SalePrice
0	1	Government	Canada	Carretera	None	1618.5	3.0	20.0
1	2	Government	Germany	Carretera	None	1321.0	3.0	20.0
2	3	Midmarket	France	Carretera	None	2178.0	3.0	15.0
3	4	Midmarket	Germany	Carretera	None	888.0	3.0	15.0
4	5	Midmarket	Mexico	Carretera	None	2470.0	3.0	15.0

Также распечатаем его описание.

In [22]:

```
fs_description=pd.read_csv('../Data/FinancialSampleFields.csv', sep=';')
fs_description
```

Out[22]:

	Name	Meaning
0	Segment	Сегмент рынка
1	Country	Страна
2	Product	Сделано в
3	DiscountBand	Группа скидок
4	UnitsSold	Продано штук
5	ManufacturingPrice	Цена производства
6	SalePrice	Цена единицы товара
7	GrossSales	Валовая продажа
8	Discounts	Скидка
9	Sales	Стоимость партии товара
10	COGS	Себестоимость проданных товаров
11	Profit	Прибыль
12	Date	Дата
13	MonthNumber	Месяц номер
14	MonthName	Месяц
15	Year	Год

Функции *max(), min()*

Функция *max()* подсчитывает максимальное значение в серии. Найдем максимальную скидку с группировкой по стране.

In [23]:

```
fs.groupby("Country")[["Discounts"]].max()
```

Out[23]:

Country	Discounts
Albania	22.48
Canada	119756.00
Egypt	22.67
Finland	22.48
France	111375.00
Germany	106512.00
Mexico	149677.50
Pakistan	22.67
United States of America	125820.00

Функция *min()* подсчитывает минимальное значение в серии. Найдем минимальное число единиц товара с группировкой по сегменту рынка и отсортируем по убыванию.

In [24]:

```
fs.groupby('Segment')['UnitsSold'].min().sort_values(ascending=False)
```

Out[24]:

```
Segment
Agriculture         1316.0
Science             1256.0
Household           1255.0
Enterprise           330.0
Channel Partners     306.0
Midmarket            218.0
Small Business       214.0
Government           200.0
Name: UnitsSold, dtype: float64
```

Функция *nunique()*

Функция *nunique()* позволяет посчитать количество уникальных значений в серии. Её лучше всего применять к столбцам, в которых хранятся категориальные данные.

In [25]:

```
fs.groupby("Country")[["Segment", "Product", "DiscountBand"]].nunique()
```

Out[25]:

Country	Segment	Product	DiscountBand
Albania	8	8	5
Canada	8	9	6
Egypt	8	8	5
Finland	8	8	5
France	5	6	4
Germany	8	9	6
Mexico	8	9	6
Pakistan	8	8	5
United States of America	8	9	6

Функция *count()*

Позволяет посчитать количество непустых элементов в группе. Найдём число сделок в каждом сегменте. Число непустых значений будем подсчитывать по столбцу *Id*. Если конкретный столбец явно не указать, то *pandas* выведет на печать все столбцы таблицы.

In [26]:

```
fs.groupby('Segment')["Id"].count()
```

Out[26]:

```
Segment
Agriculture          85
Channel Partners    198
Enterprise          190
Government          389
Household            70
Midmarket           178
Science              88
Small Business      202
Name: Id, dtype: int64
```

Аналогичный результат получим с помощью *value_counts()*. Сравните:

In [27]:

```
fs['Segment'].value_counts()
```

Out[27]:

```
Government          389
Small Business      202
Channel Partners    198
Enterprise          190
Midmarket           178
Science              88
Agriculture          85
Household            70
Name: Segment, dtype: int64
```

Функция *median()*

Находит медианное значение. Рассчитаем среднее и медианное значение себестоимости и прибыли по сделкам каждого производителя в каждом сегменте. Для этого воспользуемся функцией *.agg()*

In [28]:

```
report=fs.groupby(['Product','Segment'])[['COGS','Profit']].agg(['mean','median'])
report
```

Out[28]:

		COGS		Profit	
		mean	median	mean	median
Product	**Segment**				
Amarilla	**Channel Partners**	5473.406250	5913.00	14379.281250	16163.400
	Enterprise	228230.000000	198240.00	-7929.375000	-5568.750
	Government	184157.083333	18048.75	52578.609762	13425.400
	Midmarket	15423.333333	16867.50	5300.454167	5238.850
	Small Business	348916.666667	293875.00	33940.083333	28424.000
...	**...**	...	...	...	...
Velo	**Government**	105087.503390	11928.70	30450.008831	5056.800
	Household	10799.600000	10252.90	3273.482222	2894.437
	Midmarket	12522.708000	10773.00	4162.885080	2878.610
	Science	10864.538462	10781.40	3133.781308	3276.240
	Small Business	218792.096429	40306.15	17125.088786	5368.498

69 rows × 4 columns

Функция *std()*

Рассчитывает стандартное отклонение. Найдем среднее, медианное значения и стандартное отклонение себестоимости и прибыли от продажи товара в каждом сегменте.

In [29]:

```
fs.groupby('Segment')[['COGS', 'Profit']].agg(['mean','median','std'])
```

Out[29]:

	COGS			Profit		
	mean	median	std	mean	median	std
Segment						
Agriculture	10600.100000	10623.20	2120.259169	2922.329200	2641.8000	950.974586
Channel Partners	7566.898485	7578.35	3562.078901	8236.115288	4752.0015	6952.042483
Enterprise	111395.994211	66360.00	124914.244317	-1769.628926	2293.1640	9780.839737
Government	108070.419280	12660.00	194388.604173	29995.488933	5166.0960	51387.453473
Household	10628.760000	10385.90	2056.484196	3146.672071	3080.9780	1140.580805
Midmarket	14472.774157	12460.00	7509.130571	4983.232983	3874.3850	3296.928288
Science	10408.586364	10550.40	1902.168898	3223.880773	3166.4285	1103.447176
Small Business	194919.771782	13902.35	240965.024680	22076.607757	5134.7140	29059.520725

Сводные таблицы

Что такое "сводная таблица" и чем она отличается от "плоской" таблицы? Как правило, значениями плоской таблицы являются атрибуты параметров объектов, расположенных в строках. Названия параметров расположены в столбцах. Например, каждая строка *Financial Sample* является объектом - сделкой по продаже товара. Параметры сделки указаны в столбцах. на пересечении строки и столбца содержится значение параметра для конкретного объекта.

Значениями сводной таблицы является результат вычисления агрегирующей функции - сгруппированные данные. При этом показатели, на основе которых происходит объединение данных, расположены в заголовках строк и столбцов.

Функция *pivot_table()*

Функция *pivot_table()* позволяет быстро и просто составлять сводные таблицы Согласно документации https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html (https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html)

```
pandas.pivot_table(data, values=None, index=None, columns=None, aggfunc='mean', fi
ll_value=None, margins=False, dropna=True, margins_name='All', observed=False)
```

параметрами функции являются:

- data - исходная таблица с данными
- index - строки
- columns - столбцы
- values - агрегируемый столбец исходной таблицы
- aggfunc - функция агрегации
- margins - вывод суммирующих строки и столбца (логическое значение)
- fill_value - значение, на которое следует заменять пустые значения

- dropna - удалять или нет результаты, в которых все значения пустые (логическое)

Составим сводную таблицу, в которой отобразим макимальную скидку, предоставленную каждым производителем в каждом сегменте рынка. Расположим названия сегментов по строкам, а производителей по столбцам. В качестве агрегируемого столбца укажем *Discounts* - столбец, в котором содержится значение скидки. Функция агрегации "max".

In [30]:

```
pd.pivot_table(fs, values='Discounts', index='Segment', columns=['Product'], aggfunc='max')
```

Out[30]:

Product	Amarilla	Bal	Carretera	Geotrack	Montana	Nice	Paseo	VTT	Ve
Segment									
Agriculture	NaN	22.10	20.21	21.72	21.34	19.08	20.78	22.29	
Channel Partners	4895.520	22.10	2556.84	22.48	3831.84	19.08	5314.32	3250.80	
Enterprise	49770.000	22.48	51881.25	22.48	31466.25	21.72	33563.75	55387.50	4
Government	78400.000	19.64	81445.00	21.34	109147.50	21.16	149677.50	98245.00	11
Household	NaN	19.27	22.48	22.29	22.67	22.10	18.70	22.67	
Midmarket	5279.175	22.48	5005.65	22.29	4830.00	22.67	5757.75	6974.10	
Science	NaN	22.67	22.29	22.29	22.29	20.59	22.48	21.72	
Small Business	111375.000	17.76	92763.00	22.67	102667.50	22.48	125820.00	106722.00	11

Добавим для производителей уровень группирвки по стране. Для большей наглядности ограничимся двумя странами - Германией и Францией.

In [31]:

```
fs_GF = fs[fs['Country'].isin(['Germany','France'])]
```

In [32]:

```
pd.pivot_table(fs_GF, values='Discounts',
               index='Segment', columns=['Country','Product'], aggfunc='max')
```

Out[32]:

Country	France						Germany		
Product	Amarilla	Carretera	Montana	Paseo	VTT	Velo	Amarilla	Bal	Ca
Segment									
Agriculture	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12.66	
Channel Partners	4158.00	1581.36	1967.28	3201.66	2412.72	2124.360	3088.80	13.22	
Enterprise	19703.75	33563.75	20891.25	33563.75	2180.00	12431.250	43518.75	15.68	5
Government	70462.00	81445.00	52479.00	94178.00	62769.00	43596.000	26698.00	19.08	3
Household	NaN	NaN	NaN	NaN	NaN	NaN	NaN	11.90	
Midmarket	3108.00	588.00	3108.00	3420.90	6974.10	7795.125	1309.50	20.02	
Science	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Small Business	111375.00	45801.00	63828.00	35748.00	21978.00	34839.000	18750.00	NaN	9

Доступ к элементам сводной таблицы

Как получить доступ к отдельному столбцу или столбцам? Извлечем из сводной таблицы значение максимальной скидки, предоставленной производителем *Amarilla* для правительственного сегмента Франции. Это число находится в строке *Government* и столбце *France Amarilla*.

Присвоим результату вычисления сводной таблицы имя *pvt* и выполним команду `type(pvt)` чтобы узнать, какого типа этот объект.

In [33]:

```
pvt=pd.pivot_table(fs_GF, values='Discounts',
               index='Segment', columns=['Country','Product'], aggfunc='max')
```

In [34]:

```
type(pvt)
```

Out[34]:

```
pandas.core.frame.DataFrame
```

pvt - таблица *pandas*, значит у нее есть атрибуты *columns* и *index*.

In [35]:

```
pvt.columns
```

Out[35]:

```
MultiIndex([( 'France',  'Amarilla'),
            ( 'France', 'Carretera'),
            ( 'France',   'Montana'),
            ( 'France',     'Paseo'),
            ( 'France',       'VTT'),
            ( 'France',      'Velo'),
            ('Germany',  'Amarilla'),
            ('Germany',       'Bal'),
            ('Germany', 'Carretera'),
            ('Germany',  'Geotrack'),
            ('Germany',   'Montana'),
            ('Germany',      'Nice'),
            ('Germany',     'Paseo'),
            ('Germany',       'VTT'),
            ('Germany',      'Velo')],
           names=['Country', 'Product'])
```

In [36]:

```
pvt.index
```

Out[36]:

```
Index(['Agriculture', 'Channel Partners', 'Enterprise', 'Government',
       'Household', 'Midmarket', 'Science', 'Small Business'],
      dtype='object', name='Segment')
```

Тогда применим `.loc[]` для доступа к искомому значению скидки

In [37]:

```
pvt.loc['Government', ( 'France',  'Amarilla')]
```

Out[37]:

```
70462.0
```

Консолидация: объединение данных с помощью сводной таблицы

Задача. Имеются csv-файлы с информацией об изменении цены акций.

1. GAZA (https://drive.google.com/file/d/1bw3d01WXw5_N1skmnXwZUHFeT_xTiU1c/view?usp=sharing)
2. KMAZ (https://drive.google.com/file/d/1JKendyxL4tF4-JrA7DzkHKzgSYjHtVkT/view?usp=sharing)
3. ROSN (https://drive.google.com/file/d/18lmO_nT9yaghfkLfe3UNkmqfwszZI1Si/view?usp=sharing)

Источник: finam.ru (https://www.finam.ru/)

Задача. Объединить эти данные в одну таблицу, в строках разместить дату, в столбцах код компании, в ячейках цену акции в соответствующий день.

In [38]:

```
GAZA=pd.read_csv('../Data/t_GAZA_200401_201130.csv',sep=';')
KMAZ=pd.read_csv('../Data/t_KMAZ_200401_201130.csv',sep=';')
ROSN=pd.read_csv('../Data/t_ROSN_200401_201130.csv',sep=';')
```

Ознакомимся кратко с этими источниками, отобразив первые пять строк.

In [39]:

```
GAZA.head()
```

Out[39]:

	<TICKER>	<PER>	<DATE>	<TIME>	<CLOSE>
0	GAZA	W	06/04/20	0	375.5
1	GAZA	W	13/04/20	0	386.5
2	GAZA	W	20/04/20	0	396.5
3	GAZA	W	27/04/20	0	387.0
4	GAZA	W	04/05/20	0	382.0

In [40]:

```
KMAZ.head()
```

Out[40]:

	<TICKER>	<PER>	<DATE>	<TIME>	<CLOSE>
0	KMAZ	W	06/04/20	0	54.1
1	KMAZ	W	13/04/20	0	53.0
2	KMAZ	W	20/04/20	0	54.9
3	KMAZ	W	27/04/20	0	55.5
4	KMAZ	W	04/05/20	0	56.0

In [41]:

```
ROSN.head()
```

Out[41]:

	<TICKER>	<PER>	<DATE>	<TIME>	<CLOSE>
0	ROSN	W	06/04/20	0	344.00
1	ROSN	W	13/04/20	0	313.90
2	ROSN	W	20/04/20	0	324.00
3	ROSN	W	27/04/20	0	335.65
4	ROSN	W	04/05/20	0	348.00

Объединение таблиц с помощью .concat()

In [42]:

```
# "Объединение" таблиц проводится с помощью .concat()
join=pd.concat([GAZA,KMAZ,ROSN], axis=0)

# в объединенной таблице преобразуем дату к типу даты
join['<DATE>']=pd.to_datetime(join['<DATE>'],dayfirst=True)

# и перестроим индекс
join.index=range(len(join))
join
```

Out[42]:

	<TICKER>	<PER>	<DATE>	<TIME>	<CLOSE>
0	GAZA	W	2020-04-06	0	375.50
1	GAZA	W	2020-04-13	0	386.50
2	GAZA	W	2020-04-20	0	396.50
3	GAZA	W	2020-04-27	0	387.00
4	GAZA	W	2020-05-04	0	382.00
...	...	...	...	...	...
97	ROSN	W	2020-10-26	0	349.80
98	ROSN	W	2020-11-02	0	382.35
99	ROSN	W	2020-11-09	0	419.35
100	ROSN	W	2020-11-16	0	464.80
101	ROSN	W	2020-11-23	0	461.20

102 rows × 5 columns

Проверка по числу строк и столбцов.

In [43]:

```
# Проверка. В объединенной таблице столько же столбцов, сколько в любой из исходных таблиц
# и столько строк, сколько в сумме у всех объединяесыь таблиц.
print(GAZA.shape)
print(KMAZ.shape)
print(ROSN.shape)
print(join.shape)
```

```
(34, 5)
(34, 5)
(34, 5)
(102, 5)
```

Краткая информация об объединенной таблице

In [44]:

```
join.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 102 entries, 0 to 101
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   <TICKER>  102 non-null    object
 1   <PER>     102 non-null    object
 2   <DATE>    102 non-null    datetime64[ns]
 3   <TIME>    102 non-null    int64
 4   <CLOSE>   102 non-null    float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(2)
memory usage: 4.1+ KB
```

Создадим сводную таблицу

In [45]:

```
pvt=join.pivot_table(index='<DATE>', columns='<TICKER>', values='<CLOSE>')
pvt
```

Out[45]:

<TICKER> <DATE>	GAZA	KMAZ	ROSN
2020-04-06	375.5	54.1	344.00
2020-04-13	386.5	53.0	313.90
2020-04-20	396.5	54.9	324.00
2020-04-27	387.0	55.5	335.65
2020-05-04	382.0	56.0	348.00
2020-05-11	377.5	54.3	350.05
2020-05-18	387.0	55.0	364.50
2020-05-25	381.0	55.9	376.20
2020-06-01	389.5	60.0	401.95
2020-06-08	384.5	57.3	376.10
2020-06-15	386.5	57.6	376.15
2020-06-22	390.0	57.0	367.05
2020-06-29	384.0	57.1	365.35
2020-07-06	390.0	56.2	364.00
2020-07-13	385.5	58.5	359.85
2020-07-20	395.0	58.1	359.75
2020-07-27	389.5	57.7	356.90
2020-08-03	395.0	65.7	367.15
2020-08-10	398.0	65.4	387.95
2020-08-17	412.5	64.4	382.80
2020-08-24	426.5	63.9	380.65
2020-08-31	405.5	61.8	372.50
2020-09-07	407.0	62.6	370.05
2020-09-14	408.0	63.2	376.50
2020-09-21	391.0	60.6	381.40
2020-09-28	389.0	59.3	380.45
2020-10-05	390.0	59.6	386.55
2020-10-12	391.5	62.0	382.95
2020-10-19	399.0	61.4	379.85
2020-10-26	387.5	57.7	349.80
2020-11-02	392.5	60.8	382.35
2020-11-09	390.5	61.2	419.35

<TICKER> <DATE>	GAZA	KMAZ	ROSN
2020-11-16	393.0	61.5	464.80
2020-11-23	393.0	63.9	461.20

Проведена консолидация данных: объединенная таблица содержит цену акций из всех трех источников, метки ндекса указывают на момент времени - день, в который совершалась торговля.

Проверка. Подсчитаем число строк в консолидированной таблице. Оно совпадает с числом строк в любой из исходных таблиц.

In [46]:

```
len(pvt)
```

Out[46]:

34

Выполните контрольные задания

Исходный набор данных

Загрузите в pandas.DataFrame набор данных *ListingAm.csv* и файл *ListingAmDescription.csv* с описанием полей этого набора данных.

In [47]:

```
# Ваш код здесь
```

Выполните задания

1. Найдите среднее число коек в каждом районе Амстердама.

In [48]:

```
# Ваш код здесь
```

2. Найдите среднюю оценку чистоты (качества уборки) помещения для каждого типа собственности.

In [49]:

```
# Ваш код здесь
```

3. Найдите среднюю цену размещения в для каждого типа комнаты в районе *Westerpark*.

In [50]:

```
# Ваш код здесь
```

4. С помощью *groupby()* найдите количество объектов размещения для каждого типа комнаты в *Noord-West*.

In [51]:

```
# Ваш код здесь
```

5. Выполните задание 4 с помощью *.value_counts()*

In [52]:

```
# Ваш код здесь
```

6. Найдите среднее и медианное значения числа спален и коек в разрезе по типу собственности в *Bos en Lommer*.

In [53]:

```
# Ваш код здесь
```

7. Используя результат предыдущего задания, определите:

   а) для каких типов собственности средние и медианные значения числа коек равны?
   б) для каких типов собственности медианное значение числа спален превышает среднее значение?

In [54]:

```
# Ваш код здесь
```

8. С помощью сводной таблицы найдите минимальную цену размещения каждого типа комнаты а каждом районе.

In [55]:

```
# Ваш код здесь
```

9. Какие владельцы владеют пятью и более объектами недвижимости? Распечатайте их идентификаторы вместе с именами и числом объектов в порядке убывания числа объектов.

In [56]:

```
# Ваш код здесь
```