

# Получение и предобработка данных. Первичная работа с объектом DataFrame

## Цель занятия

После освоения темы:

- вы узнаете виды, источники и способы хранения данных (csv, tsv-файлы и другие);
- узнаете структуры данных и инструменты, предоставляемые библиотекой `Pandas` для работы с данными;
- сможете выполнять выгрузку данных с использованием библиотеки `Pandas`;
- сможете проводить предварительную обработку данных с использованием библиотеки `Pandas` — получать информацию о `DataFrame`, работать со строками и столбцами;
- сможете осуществлять группировку и агрегацию таблиц с использованием `Pandas`.

## План занятия

1. [Виды и источники данных](#)
2. [Предобработка данных](#)
3. [Первичная работа с DataFrame](#)
4. [Введение в агрегирование и сводные таблицы](#)

## Используемые термины

**Источник данных** — ресурс, из которого мы берем данные для анализа.

**Репозиторий** — место хранения данных.

**.csv (comma-separated values)** — значения, разделенные запятыми.

**.tsv (tabulation-separated values)** — разделителем выступает табуляция.

**.sav** — расширение баз данных формата пакета SPSS (часто используется опросными компаниями, социологическими службами).

**.dta** — расширение пакета Stata (научные исследования).

**.xls, .xlsx** — расширения Excel.

**.json** — текстовый формат обмена данными, основанный на JavaScript.

**.rmd** — формат сохранения данных пакета R Studio (на базе языка R, популярного для статистики).

**Data pre-processing** — исследование структуры данных на предмет возможности протестировать необходимые аналитические гипотезы и соответствующая коррекция имеющейся базы данных.

## Конспект занятия

### 1. Виды и источники данных

В статистике встречаются данные двух видов:

- количественные (например, визуализация метрик в процентах или дробях);
- качественные.

Python может анализировать количественные данные, а также качественные при условии, что они переведены в количественные.

Как правило, количественные данные анализировать проще. Качественные необходимо перевести в ту или иную количественную презентацию.

Также можем разделить данные на два типа:

- первичные (источник данных — мы сами);
- вторичные (собраны кем-то до нас, за нас).

Собирать первичные данные — отдельное искусство (за пределами программы курса). Источников данных огромное множество, и в дальнейшем мы рассмотрим некоторые примеры.

Вместе с тем, для конкретных задач анализа мы можем комбинировать первичные и вторичные данные.

## Источники данных

**Источник данных** — это ресурс, из которого мы берем данные для анализа.

Обсудим следующие группы ресурсов:

- репозитории,
- статистика,
- опросы.

Многие сайты и разделы сайтов специализируются на хранении данных различного типа. Такие сайты можно назвать репозиториями.

**Репозиторий** — место хранения данных.

[Kaggle](#) — сайт-система организации конкурсов по исследованию данных, а также социальная сеть специалистов по обработке данных и машинному обучению.

Kaggle хранит коллекций самых различных данных, которые использовались в соревнованиях. Наборы данных хорошо подходят для тренировочных целей, так как они уже предобработаны.

[GitHub](#) — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. На подобных репозиториях ученые и аналитики данных часто публикуют базы данных своих исследований и проектов на персональных репозиториях. Ключевые научные журналы могут даже вводить это как специальное требование, чтобы исследовательскую стратегию могли реплицировать другие коллеги.

[Harvard Dataverse](#) — еще один специальный репозиторий, на котором публикуются репликационные материалы (в том числе и база данных) об уже опубликованных научных исследованиях.

**Статистические данные** достаточно часто нам необходимы для проведения тех или исследований и проектов. Примеры социально-экономической и демографической статистики:

Россия	Мир
<ul style="list-style-type: none"><li>• <a href="#">Росстат</a></li><li>• <a href="#">Демоскоп</a></li><li>• <a href="#">Портал открытых данных РФ</a> и <a href="#">Москвы</a></li><li>• ИНИД</li></ul>	<ul style="list-style-type: none"><li>• Всемирный банк</li><li>• John Hopkins Coronavirus resource center</li></ul>

Как правило, важная социальная и демографическая статистика преимущественно агрегируется либо на уровне страны, либо на региональном уровне, позволяя нам проанализировать информацию о каком-то человеке. В данном случае на помощь приходят опросы.

**Опросные данные** — достаточно распространенный источник данных для последующего анализа. Для нахождения первичных опросных данных наиболее релевантной стратегией будет посещение сайтов конкретных социологических или опросных служб.

Россия	Мир
<ul style="list-style-type: none"><li>• ВЦИОМ,</li><li>• ФОМ,</li><li>• более мелкие маркетинговые агентства, которые тоже могут публиковать данные.</li></ul>	<ul style="list-style-type: none"><li>• Gallup,</li><li>• World Values Survey,</li><li>• Life in Transition Survey.</li></ul>

## Поиск данных. Достоверность используемых данных

Процесс поиска данных — достаточно креативный процесс, но его можно разбить на этапы:

1. Определиться с тематикой (идеальная модель вашей базы данных).
2. Подумать, какие субъекты могут собирать такие данные:
  - коммерческие компании;
  - некоммерческие организации (НКО);

- университеты;
- исследовательские центры;
- волонтеры-энтузиасты или независимые ученые, аналитики.

На втором этапе можно строить свой поиск через:

- Междисциплинарные репозитории.
- Сайты конкретного субъекта (при сложностях обратите внимание на контакты). Можно обратиться к сайтам конкретного субъекта источника данных, и в случае затруднений напрямую написать представителям центра. Зачастую люди открыты и могут представить большие массивы данных.
- Общение с экспертами напрямую. Они могут обладать более глубокими знаниями выбранной аналитической области, могут подсказать и направить.
- Социальные сети. В них есть тематические группы, где в том числе можно задать вопросы, получить обратную связь от коллег.

Обязательно проверьте, достоверны ли выбранные данные и источники. Не стоит доверять:

- сомнительной выборке (сотрудники Пятерочки считают, что это лучший магазин);
- представлению наиболее привлекательных данных;
- сравнению без конкретики;
- вводящим в заблуждение графикам;
- неправильно составленным вопросам для анкеты.

Стоит вникнуть в методологию сбора опросных данных (какая анкета, какие вопросы) и проверять, соотносятся вопросы с реальностью или нет.

## Форматы хранения данных

Данные могут храниться с разными расширениями. Одно из самых распространенных расширений — **.csv** (comma-separated values — значения, разделенные запятыми). Иногда разделитель в .csv может быть другим знаком препинания — например, точкой с запятой.

Первичная репрезентация данных в формате .csv:

2000,Sydney,XVII,

2004,Athens,XIII,

2008,Beijing,XIX,

Как данные могут интерпретироваться различными средствами аналитики данных:

2000	Sydney	XVII
2004	Athens	XIII
2008	Beijing	XIX

Особым подвидом данных .csv является расширение .tsv (tabulation-separated values), в котором разделителем выступает табуляция.

Другие расширения данных:

- **.sav** — расширение баз данных формата пакета SPSS (часто используется опросными компаниями, социологическими службами).
- **.dta** — расширение пакета Stata (научные исследования).
- **.xls, .xlsx** — расширения Excel.
- **.json** — текстовый формат обмена данными, основанный на JavaScript.
- **.rmd** — формат сохранения данных пакета R Studio (на базе языка R, популярного для статистики).

## 2. Предобработка данных

Посмотрим, каким образом можно выгружать основные типы данных в интерпретатор Python. Будем использовать функционал библиотеки Pandas.

Установим библиотеку:

```
!pip install pandas
```

Импортируем ее:

```
import pandas as pd
```

Теперь можно выгружать данные в систему.

Для наиболее популярных расширений можно использовать методы `read_csv()`, `read_stata()`, `read_spss()`, `read_excel()`. Полный список встроенных методов чтения файлов приведен в [документации библиотеки](#).

Чтобы избежать ошибок в выгрузке данных, еще раз проверьте себя:

1. Импортированы все необходимые библиотеки (например, `pandas`).
2. Правильно прописан путь к файлу.

Для самопроверки используйте встроенную библиотеку `os`:

- импортируем ее: `import os` — отвечает за работу с операционной системой;
  - `os.abspath()` — метод, показывающий путь к файлу;
  - `os.getcwd()` — показывает текущую рабочую директорию;
  - `os.chdir()` — позволяет изменить текущую рабочую директорию.
3. Нет ошибки в расширении, в структуре файла и кодировке.

## Предобработка данных. Получение информации о датасете

Важным этапом после выгрузки данных будет предварительная обработка (`data pre-processing`).

**Data pre-processing** — это исследование структуры данных на предмет возможности протестировать необходимые аналитические гипотезы и соответствующая коррекция имеющейся базы данных.

Предобработка включает:

- Исследование исходно располагаемых признаков (колонок таблицы).
- Установление структур данных, используемых по умолчанию.
- Инспекция данных на предмет наличия пропущенных значений и дубликатов и модификации датасета.

**Пример.** Рассмотрим основной функционал предобработки данных в библиотеке `pandas` с использованием [датасета](#) с информацией о пассажирах Титаника.

Датасет находится в расширении `.csv`, значит, будем использовать метод `read_csv()`, чтобы загрузить данные в нашу рабочую среду. Внутри метода необходимо добавить путь к файлу в кавычках:

```
pd.read_csv('train.csv')
```

Также сохраним датасет в переменную `df`:

```
df = pd.read_csv('train.csv')
```

После загрузки данных в среду посмотрим на наши данные. Для этого будем использовать метод `head()`, по умолчанию выводящий первые 5 строк нашей базы данных:

```
df.head()
```

Получим:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Для вывода большего количества строчек нужно передать конкретное число в скобках для метода `head()`.

Сверху видим названия наших колонок: `PassengerID` (анонимный идентификационный номер пассажира), `Survived` (0 — не выжил, 1 — выжил), `Pclass` (класс проезда на Титанике — первый, второй или третий), `Name`, `Sex`, `Age`, `SibSp`, `Parch`, `Ticket`, `Fare`, `Cabin`, `Embarked`.

Выделенные жирным шрифтом числа в левой колонке — индексы строк. По умолчанию система индексации начинается с нуля.

С помощью метода `info()` можем посмотреть на основную информацию о датасете:

```
df.info()
```



Получим ряд важных характеристик:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     891 non-null   int64
1   Survived        891 non-null   int64
2   Pclass          891 non-null   int64
3   Name            891 non-null   object
4   Sex             891 non-null   object
5   Age             714 non-null   float64
6   SibSp           891 non-null   int64
7   Parch           891 non-null   int64
8   Ticket          891 non-null   object
9   Fare            891 non-null   float64
10  Cabin           204 non-null   object
11  Embarked        889 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

В полученной выдаче мы видим ряд важных характеристик. Прежде всего тип данных — для Pandas используется специальный тип данных `DataFrame`. Число 891 — количество наблюдений, то есть 891 пассажир. Общее количество колонок в таблице 12. Далее приведена таблица с информацией о колонках: индекс, название, количество полных наблюдений, тип данных. Также метод `info()` выводит количество оперативной памяти, которую занимает наш датасет.

Метод `set_axis()` позволяет менять названия строк и столбцов. В случае данных Титаника для нас это не актуально, поэтому создадим искусственный датасет:

```
df1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
```

Получим:

	A	B
0	1	4
1	2	5
2	3	6

Чтобы переименовать строчки, мы можем воспользоваться следующим кодом:

```
df1 = df1.set_axis(['a', 'b', 'c'], axis='index')
```

В метод `set_axis` мы передаем список новых названий (`a`, `b`, `c`) и устанавливаем параметр `axis='index'`, т.е. показывая, что мы изменяем названия строк.

В результате получим:

	A	B
a	1	4
b	2	5
c	3	6

То же самое можем проделать над столбцами:

```
df1 = df1.set_axis(['I', 'II'], axis='columns')
```

	I	II
a	1	4
b	2	5
c	3	6

## Работа с пропущенными значениями и дубликатами

Как бы тщательно мы не собирали данные, все равно можно столкнуться с пропущенными значениями. Рассмотрим ряд вспомогательных методов.

Чтобы проверить, не попался ли «кот в мешке», мы можем использовать специальные функции:

- Метод `isnull()` проверяет, является ли ячейка в датафрейме пропущенным значением. На выходе метод выдает тип данных `bool`.
- Чтобы посмотреть количество пропусков по колонкам, сагрегируем данные с помощью метода `sum()` (вернемся опять к данным Титаника):

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

Из данных по Титанику получаем:

```

PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age             177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin           687
Embarked         2
dtype: int64

```

В большинстве колонок у нас нет пропущенных значений, за исключением колонок Age, Cabin, Embarked.

Какие простые способы работы с пропущенными значениями существуют?

Метод `dropna()` по умолчанию удаляет все строки, где есть хотя бы одно пропущенное значение в колонках.

Синтаксис метода:

```
df.dropna(how = , subset = , inplace = )
```

Код	Значение
<code>df</code>	Имя датафрейма
<code>how =</code>	Как необходимо удалить пропуски: <ul style="list-style-type: none"> <li>если <code>how = 'any'</code>, <code>dropna</code> удалит строку, если хотя бы одно значение в колонках пропущено;</li> <li>если <code>how = 'all'</code>, <code>dropna</code> удалит строку, если в ней пропущены значения во всех колонках</li> </ul>
<code>subset =</code>	Список колонок, которые необходимо рассмотреть
<code>inplace =</code>	Будет ли операция замены пропусков произведена автоматически (без необходимости переприсвоения работы метода)

Вовсе не обязательно удалять наблюдения с пропущенными значениями.

Альтернатива — замена пропусков по какому-то закону и/или в связи с нашими содержательными соображениями.

Например, если мы предполагаем, что неизвестная информация по возрасту в данных Титаника не связана с каким-то «умыслом», можно попробовать заменить все пропуски на средние значения:

```
df["Age"].fillna(df["Age"].mean(), inplace = True)
```

Код	Значение
<code>df["Age"]</code>	Часть датасета, в которой необходимо осуществить замену
<code>fillna</code>	Метод <code>fillna()</code> , который позволяет заменять пропуски
<code>df["Age"].mean()</code>	Чем заменить пропуски? В данном случае — средним возрастом по выборке
<code>inplace = True</code>	Обновляем датафрейм

Иногда вследствие человеческого фактора и/или ошибки агрегирования данных в датафрейме могут появляться строки-дубликаты. В таком случае можно использовать метод `duplicated()`, который проверяет, имеется ли у строки дубликат.

### 3. Первичная работа с DataFrame

В Pandas есть два главных типа данных — `DataFrame` и `Series`. Тип данных `Series` — достаточно простой. Как правило, он передается с помощью списка, состоящего из последовательности каких-то символов.

Сконцентрируемся на типе данных `DataFrame`.

**DataFrame** — это таблица (со столбцами и строками).

Создать `DataFrame` можно с помощью функции `pd.DataFrame`, куда мы передадим словарь, где ключом словаря будет название колонки, а значением — соответствующие значения ячеек для каждой из колонок:

```
pd.DataFrame({'Column 1': ['Black', 50], \
              'Column 2': [155, ['orange', 'apple']]})
```

В результате получим:

	Column 1	Column 2
0	Black	155
1	50	[orange, apple]

## Базовые атрибуты датафрейма

По аналогии с NumPy в датафреймах Pandas существуют свои атрибуты. Рассмотрим их на примере. Предположим, у нас есть данные о продажах лимонов и яблок в двух магазинах, представленные в переменной `df` в виде датафрейма Pandas:

	fruit	shop	pl	Q	P	total
0	lemons	Shop A	online	1	5	5
1	lemons	Shop A	online	2	4	8
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15
4	apples	Shop A	online	3	6	18
5	apples	Shop A	offline	4	6	24
6	apples	Shop A	offline	5	8	40
7	apples	Shop B	online	6	9	54
8	apples	Shop B	offline	7	9	63
9	apples	Shop B	offline	4	3	12
10	apples	Shop A	offline	4	3	12

Основные атрибуты датафрейма Pandas:

- `ndim` — число измерений;
- `shape` — размерность;
- `size` — размер (число ячеек);
- `columns` — информация о названиях колонок;
- `axes` — информация о названии осей;
- `dtypes` — типы данных для каждой из колонок.

Посмотрим на атрибуты нашего DataFrame:

Команда	Результат выполнения
<code>print(df.ndim)</code>	2
<code>print(df.shape)</code>	(11, 6)
<code>print(df.size)</code>	66
<code>print(df.columns)</code>	Index(['fruit', 'shop', 'pl', 'Q', 'P', 'total'], dtype='object')
<code>print(df.axes)</code>	[RangeIndex(start=0, stop=11, step=1), Index(['fruit', 'shop', 'pl', 'Q', 'P', 'total'], dtype='object')]
<code>print(df.dtypes)</code>	fruit            object shop            object pl               object Q                int64 P                int64 total            int64 dtype:           object

## Первичное исследование датафрейма

Для первичного исследования датафрейма будут полезны три функции:

- `head(n)` — вывод первых n строк датафрейма;
- `tail(n)` — вывод последних n строк датафрейма;
- `describe()` — вывод описательных статистик (только для колонок типа `float` и `int`).

Рассмотрим эти функции на нашем примере:

	fruit	shop	pl	Q	P	total
0	lemons	Shop A	online	1	5	5
1	lemons	Shop A	online	2	4	8
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15
4	apples	Shop A	online	3	6	18
5	apples	Shop A	offline	4	6	24
6	apples	Shop A	offline	5	8	40
7	apples	Shop B	online	6	9	54
8	apples	Shop B	offline	7	9	63
9	apples	Shop B	offline	4	3	12
10	apples	Shop A	offline	4	3	12

Результат выполнения `df.head()`

	fruit	shop	pl	Q	P	total
0	lemons	Shop A	online	1	5	5
1	lemons	Shop A	online	2	4	8
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15
4	apples	Shop A	online	3	6	18

Результат выполнения `df.tail()`

	fruit	shop	pl	Q	P	total
6	apples	Shop A	offline	5	8	40
7	apples	Shop B	online	6	9	54
8	apples	Shop B	offline	7	9	63
9	apples	Shop B	offline	4	3	12
10	apples	Shop A	offline	4	3	12

Результат выполнения `df.describe()` – мы можем видеть, что в исходной таблице есть только три колонки с типом данных `int` или `float`, для них выводятся описательные характеристики:

- `count` – количество не пропущенных значений,
- `mean` – среднее арифметическое значение,
- `std` – стандартное отклонение,
- `min` – минимальное значение,
- `25%` – нижняя квартиль,
- `50%` – медиана,
- `75%` – верхняя квартиль,
- `max` – максимальное значение.

	Q	P	total
count	11.000000	11.000000	11.000000
mean	3.727273	5.727273	23.727273
std	1.793929	2.148996	19.733681
min	1.000000	3.000000	5.000000
25%	2.500000	4.500000	11.000000
50%	4.000000	5.000000	15.000000
75%	4.500000	7.000000	32.000000
max	7.000000	9.000000	63.000000

## Индексирование

Мы можем сослаться на колонку без использования специфических методов, двумя способами – `df.column_name` и `df['column_name']`.

Чтобы сослаться на конкретную ячейку, нужно сначала сослаться на название колонки, а потом – на индекс строки:

- `df.column_name[index];`
- `df['column_name'][index].`

Выведем значение первой строки колонки `total`.

Первый способ: `df.total[0]`

Второй способ: `df['total'][0]`



В обоих случаях будет выведено значение 5:

	fruit	shop	pl	Q	P	total
0	lemons	Shop A	online	1	5	5
1	lemons	Shop A	online	2	4	8
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15
4	apples	Shop A	online	3	6	18
5	apples	Shop A	offline	4	6	24
6	apples	Shop A	offline	5	8	40
7	apples	Shop B	online	6	9	54
8	apples	Shop B	offline	7	9	63
9	apples	Shop B	offline	4	3	12
10	apples	Shop A	offline	4	3	12

Теперь сошлемся не на один элемент колонки, а на несколько: `df['total'][0:2]`

Результат выполнения:

```
0    5
1    8
Name: Q, dtype: int64
```

	fruit	shop	pl	Q	P	total
0	lemons	Shop A	online	1	5	5
1	lemons	Shop A	online	2	4	8
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15
4	apples	Shop A	online	3	6	18
5	apples	Shop A	offline	4	6	24
6	apples	Shop A	offline	5	8	40
7	apples	Shop B	online	6	9	54
8	apples	Shop B	offline	7	9	63
9	apples	Shop B	offline	4	3	12
10	apples	Shop A	offline	4	3	12

Рассмотрим способы срезов `DataFrame` в `Pandas`. Есть два основных способа осуществления срезов:

- по индексу:

```
iloc[start:stop:step, start:stop:step]
```

- по лейблу:

```
loc[start:stop:step, [column_names]]
```

Рассмотрим пример. Пусть мы хотим вывести элементы первой строки первого столбца, то есть значение 'lemons'. Мы можем сделать это двумя способами:

- `df.iloc[0,0]`
- `df.loc[0, 'fruit']`

Для того, чтобы вывести срез, который состоит более чем из одной ячейки для метода `iloc` необходимо передать последовательность индексов:

```
df.iloc[0:2,0:2]
```

В результате на экран будут выведены первые две строки первых двух столбцов:

	fruit	shop
0	lemons	Shop A
1	lemons	Shop A

При использовании метода `loc` для строк мы передаем интервал индексов, а для столбцов — список, включающий названия колонок:

```
df.loc[0:2,['fruit', 'shop']]
```

Результат выполнения программы:

	fruit	shop
0	lemons	Shop A
1	lemons	Shop A
2	lemons	Shop A

Pandas поддерживает срез по условию, который называется фильтрацией. Схема осуществления среза по условию:

```
df.loc[condition_2 &/| condition_2 &/|...&/| condition_3]
```

Например, нужно выбрать транзакции с лимонами на общую стоимость более 8 у. е.:

```
df.loc[(df.total > 8) & (df.fruit == 'lemons')]
```

Результат выполнения:

	fruit	shop	pl	Q	P	total
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15

#### 4. Введение в агрегирование и сводные таблицы

**Агрегирование** — обобщение вложенных структур данных. Является достаточно важным процессом трансформации данных.

По сути, агрегирование — это переход от более низкого уровня данных к более высокому.

- Пример 1. Проведен межстрановой опрос потребителей товара X. Нам нужно понять тренды по странам в целом. Для этого агрегируем данные индивидуального уровня на страновой уровень. Меняется единица анализа (строка таблицы): от респондента к стране.
- Пример 2. Изучение межрегиональной вариации в баллах выпускных экзаменов в стране. Данные собраны на уровне каждой школы в отдельности. Необходимо обобщить данные до уровня регионов. Меняется единица анализа: от школы к региону.

Агрегирование можно осуществить с помощью метода `groupby()`.

Общая схема может выглядеть так:

```
groupby('grouping_variable')['aggregation_variable'].method_aggregation()
```

В показанной схеме:

- `grouping_variable` — переменная группировки;
- `aggregation_variable` — переменная агрегирования;
- `method_aggregation()` — метод агрегирования.

Возьмем датасет с транзакциями по покупкам. Допустим, мы хотим узнать среднюю стоимость покупок фруктов для каждого магазина.

	fruit	shop	pl	Q	P	total
0	lemons	Shop A	online	1	5	5
1	lemons	Shop A	online	2	4	8
2	lemons	Shop A	offline	2	5	10
3	lemons	Shop B	online	3	5	15
4	apples	Shop A	online	3	6	18
5	apples	Shop A	offline	4	6	24
6	apples	Shop A	offline	5	8	40
7	apples	Shop B	online	6	9	54
8	apples	Shop B	offline	7	9	63
9	apples	Shop B	offline	4	3	12
10	apples	Shop A	offline	4	3	12

То есть мы переходим от уровня транзакций до уровня конкретного магазина. И таким образом хотим обобщить данные, агрегировать их.

Реализация:

```
df.groupby('shop')['total'].mean().reset_index()
```

Результат выполнения:

	shop	total
0	Shop A	16.714286
1	Shop B	36.000000

Хорошим описательным методом агрегирования данных являются сводные таблицы.

Сводные таблицы удобно реализовывать с помощью метода `pivot_table()`. Синтаксически он более удобен, чем `groupby()`, и хорошо подходит для более сложных методов агрегирования.

Возьмем данные по транзакциям покупок фруктов. Допустим, нам нужно получить сумму стоимостей покупок для каждого вида товара в каждом магазине с разбиениями по платформе:

```
pd.pivot_table(df, \
                values='total', \
                index=['fruit', 'shop'], \
                columns=['pl'], \
                aggfunc=np.sum)
```

Результат выполнения:

		pl	offline	online
fruit	shop			
apples	Shop A		76.0	18.0
	Shop B		75.0	54.0
lemons	Shop A		10.0	13.0
	Shop B		NaN	15.0

Приведем другие методы агрегирования, которые могут быть полезны при осуществлении операций – как агрегирования с помощью `groupby`, так и сведения данных с помощью `pivot_table`.

Метод	Описание
<code>count()</code>	Общее число наблюдений
<code>first(), last()</code>	Первое и последнее наблюдения
<code>mean(), median()</code>	Среднее и медиана
<code>min(), max()</code>	Минимум и максимум
<code>std(), var()</code>	Стандартное отклонение и дисперсия
<code>mad()</code>	Среднее абсолютное отклонение деление
<code>prod()</code>	Произведение всех наблюдений
<code>sum()</code>	Сумма всех наблюдений

## Дополнительные материалы для самостоятельного изучения

1. [Find Open Datasets and Machine Learning Projects | Kaggle](#)
2. [GitHub: Let's build from here · GitHub](#)
3. [Harvard Dataverse](#)
4. [Федеральная служба государственной статистики](#)
5. [Приложение Демоскопа Weekly](#)

- 22