

## **Лабораторная работа №**

### **Минимальные требования к оборудованию**

Для оптимального студенческого опыта рекомендуется следующая аппаратная конфигурация:

- ОС: Windows 7 SP1 32/64-bit, Windows 8.1 32/64-bit или Windows 10 32/64-bit, Ubuntu 14.04 или позже, или macOS Sierra или позже
- Процессор: Двухъядерный или лучше
- Память: RAM НА 4 ГБ
- Устройство хранения данных: свободное место на 10 ГБ

### **Требования к программному обеспечению**

Необходимо следующее программное обеспечение, установленное заранее:

- Браузер: Google Chrome или Mozilla Firefox
- Conda
- JupyterLab и Jupyter Notebook
- Sublime Text (последняя версия), IDE Atom (последняя версия) или другие подобные приложения текстового редактора
- Python 3
- Следующие библиотеки Python: NumPy, pandas, Matplotlib, seaborn, geoplotlib, Bokeh и squarify

### **Соглашения**

Кодовые комбинации в тексте, именах таблицы базы данных, именах папок, именах файлов, расширениях файла, путях, фиктивных URL, вводе данных пользователем и дескрипторах Twitter показывают следующим образом: “axis=0 горизонтален, и axis=1 является вертикальным, поэтому если результат для каждой строки, следует выбрать axis=1 “.

Блок кода установлен следующим образом:

```
# индексация первого значения второй строки (1-я строка, 1-е значение)
first_val_first_row = dataset [0] [0]
np.mean (first_val_first_row)
```

Новые условия и важные слова показывают полужирным:

“Чтобы сделать выводы из визуализируемых данных, нужно обработать данные и преобразовать их в самое лучшее представление. Это то место, где **пререкание данных** используется”.

## Установка

Необходимо установить Python 3.6, pip и другие библиотеки, которые понадобятся в процессе обучения. Шаги по установке представлены ниже:

### Установка Python

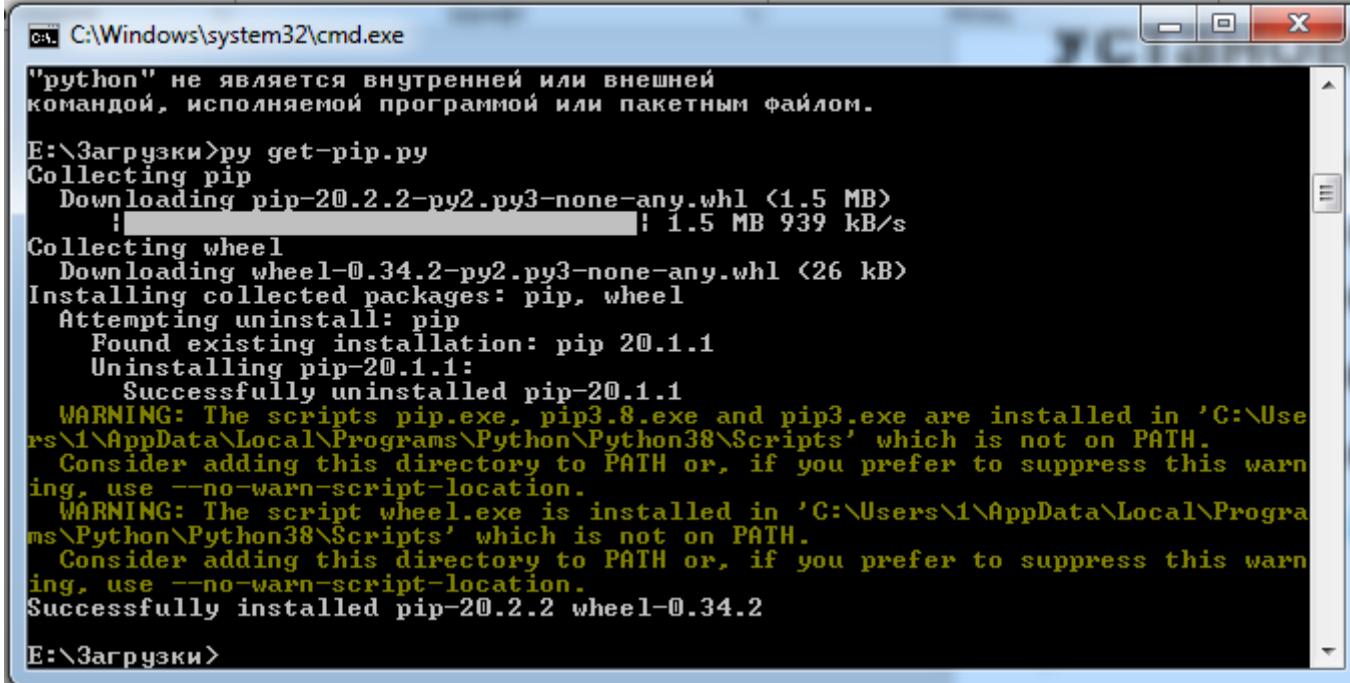
Установить Python 3.6 рекомендуется следуя инструкциям в этой ссылке:

<https://realpython.com/installing-Python/>.

### Установка pip

1. Для установки pip необходимо перейти к следующей ссылке и загрузить **get-pip.py** файл: <https://pip.pypa.io/en/stable/installing/>.
2. Затем использовать следующую команду для установки его:

python get-pip.py или py get-pip.py на Windows



```
C:\Windows\system32\cmd.exe
"python" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.

E:\Загрузки>py get-pip.py
Collecting pip
  Downloading pip-20.2.2-py2.py3-none-any.whl (1.5 MB)
    ! [██████████] 1.5 MB 939 kB/s
Collecting wheel
  Downloading wheel-0.34.2-py2.py3-none-any.whl (26 kB)
Installing collected packages: pip, wheel
  Attempting uninstall: pip
    Found existing installation: pip 20.1.1
    Uninstalling pip-20.1.1:
      Successfully uninstalled pip-20.1.1
      WARNING: The scripts pip.exe, pip3.8.exe and pip3.exe are installed in 'C:\Users\1\AppData\Local\Programs\Python\Python38\Scripts' which is not on PATH.
      Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
      WARNING: The script wheel.exe is installed in 'C:\Users\1\AppData\Local\Programs\Python\Python38\Scripts' which is not on PATH.
      Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pip-20.2.2 wheel-0.34.2

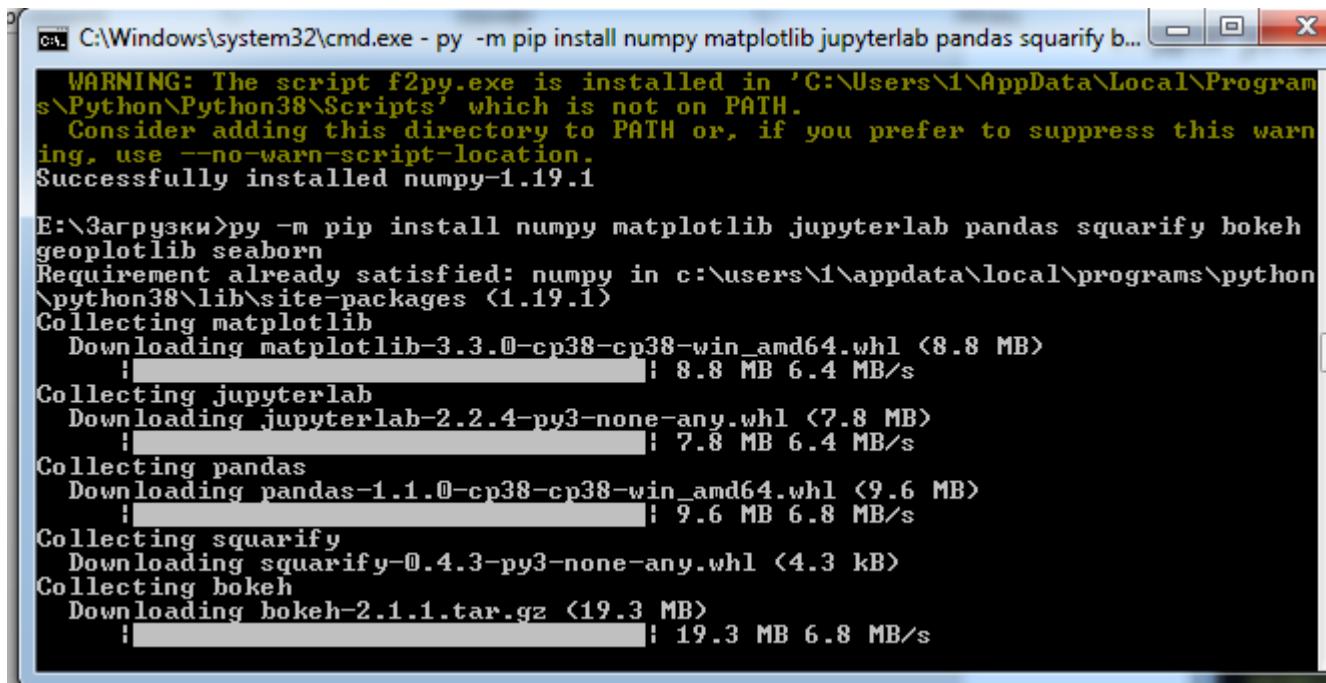
E:\Загрузки>
```

Возможно, следовало использовать **python3**, **get-pip.py** команду, из-за предыдущих версий Python на компьютере, которые уже используют команду **python**.

## Установка библиотек

Используя команду **pip**, необходимо установить следующие библиотеки:

```
python -m pip install -user numpy matplotlib jupyterlab pandas squarify
bokeh geoplotlib seaborn
py -m pip install numpy matplotlib jupyterlab pandas squarify bokeh
geoplotlib seaborn на Windows
```



```
c:\ C:\Windows\system32\cmd.exe - py -m pip install numpy matplotlib jupyterlab pandas squarify b...
WARNING: The script f2py.exe is installed in 'C:\Users\1\AppData\Local\Programs\Python\Python38\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed numpy-1.19.1

E:\Загрузки>py -m pip install numpy matplotlib jupyterlab pandas squarify bokeh
geoplotlib seaborn
Requirement already satisfied: numpy in c:\users\1\appdata\local\programs\python\python38\lib\site-packages (1.19.1)
Collecting matplotlib
  Downloading matplotlib-3.3.0-cp38-cp38-win_amd64.whl (8.8 MB)
    |██████████| 8.8 MB 6.4 MB/s
Collecting jupyterlab
  Downloading jupyterlab-2.2.4-py3-none-any.whl (7.8 MB)
    |██████████| 7.8 MB 6.4 MB/s
Collecting pandas
  Downloading pandas-1.1.0-cp38-cp38-win_amd64.whl (9.6 MB)
    |██████████| 9.6 MB 6.8 MB/s
Collecting squarify
  Downloading squarify-0.4.3-py3-none-any.whl (4.3 kB)
Collecting bokeh
  Downloading bokeh-2.1.1.tar.gz (19.3 MB)
    |██████████| 19.3 MB 6.8 MB/s
```

## Работа с JupyterLab и Jupyter Notebook

Работа над различными упражнениями и операциями выполняется в JupyterLab. Эти упражнения и операции могут быть загружены со связанного репозитория GitHub. Загрузить репозиторий можно отсюда:

<https://github.com/TrainingByPackt/Data-Visualization-with-Python>.

Можно загрузить его через GitHub или как заархивированную папку путем нажатия на зеленую кнопку

**Clone or download** на верхней правой стороне.

Для открытия Jupyter Notebooks необходимо перейти в каталог с терминалом. Чтобы сделать это, следует ввести:

**CD Data-Visualization-with-Python / <текущая глава>.**

Например:

**CD Data-Visualization-with-Python/chapter01/**

Для завершения процесса необходимо выполнить следующие шаги:

1. Для достижения каждого действия и осуществления необходимо использовать **CD** еще раз для входа в каждую папку:

**cd Activity01**

2. Как только удалось попасть в нужную папку, следует вызвать **jupyter-lab**

для запуска JupyterLab. Точно так же для Jupyter Notebook, необходимо вызвать **jupyter notebook**.

## Импорт библиотек Python

Импорт библиотек в Python очень прост:

1. Для импорта библиотек, таких как NumPy и pandas, необходимо выполнить следующий код. Это импортирует целую **numpy** библиотеку в текущий файл:

```
# import numpy  
import numpy
```

2. Можно использовать **np** вместо **numpy** в коде для вызова методов от **numpy**:

```
# import numpy and assign alias np  
import numpy as np
```

3. В более поздних главах частичный импорт будет присутствовать, как показано в следующем коде. Это загружает только метод **средний** из библиотеки:

```
from numpy import mean # only import the mean method of numpy
```

## Установка пакета кода

Необходимо скопировать пакет кода для класса в папку **C:/Code**.

## **Важность визуализации данных и исследования данных**

### **Изучение целей**

К концу этой главы будут изучены вопросы:

- Важности визуализации данных
- Вычисления основных статистических значений, таких как медиана, среднее, и различие
- Использование NumPy для пререкания данных
- Использование pandas для пререкания данных

В этой главе будет рассказано о основных операциях NumPy и pandas.

### **Введение**

В отличие от машин, люди обычно не снабжаются для интерпретации большой информации от случайного набора чисел и сообщений в данной части данных. В то время как они могут знать то, из чего в основном состоят данные, они могли бы нуждаться в помощи для понимания их полностью. В силу логических возможностей человек понимает вещи лучше всего посредством обработки визуальной информации. Когда данные представлены визуально, вероятность понимания сложных сборок и чисел увеличивается.

Python недавно появился в качестве языка программирования, который работает хорошо для анализа данных. Python имеет приложения через data science, которые преобразовывают данные в применимый формат, анализируют их и извлекают полезные заключения из данных для представления их в лучшем виде. Это обеспечивает библиотеки визуализации данных, которые могут помочь собрать графические представления быстро.

В этой книге будет показано, как использовать Python в сочетании с различными библиотеками, таким как NumPy, pandas, Matplotlib, seaborn, и

**geoplotlib**, для создания эффективной визуализации данных с помощью реальных данных. Помимо этого, будет рассказано о функциях различных типов диаграмм и сравнение их преимуществ и недостатков. Это поможет выбрать тип диаграммы, который подходит для визуализации данных. Изучая основы, можно покрыть более сложные понятия, такие как интерактивная визуализация и как **bokeh** может использоваться для создания анимированной визуализации, которая рассказывает историю.

## **Введение в визуализацию данных**

Компьютеры и смартфоны хранят данные, такие как имена и номера в цифровом формате.

**Представление данных** относится к форме, в которой можно сохранить, обработать и передать данные.

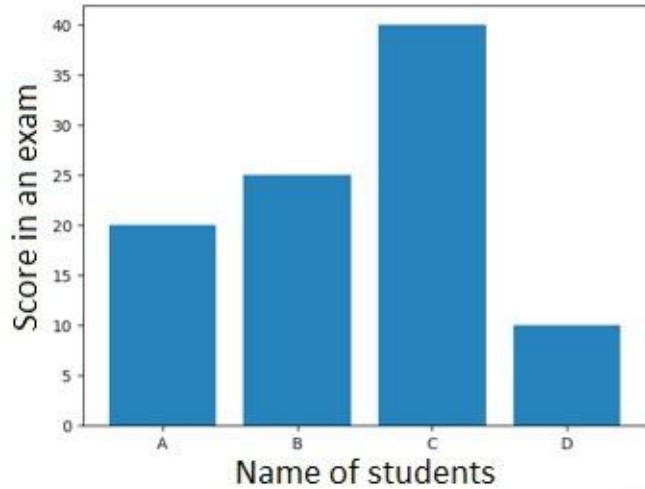
Представления могут рассказать историю и передать ключевые открытия аудитории. Соответственно не моделируя информацию для использования его для создания значимых результатов ее значение уменьшается. Создание представлений помогает достигнуть более ясной, более краткой, и прямой перспективы информации, облегчающей для любого понимание данных.

Информация не действительно эквивалентна данным. Представления являются полезным аппаратом для нахождения понимания спрятанному в данных. Таким образом представления преобразовывают информацию в полезные данные.

## **Важность визуализации данных**

Визуальные данные очень легко понять по сравнению с данными в любой другой форме. Вместо того, чтобы просто смотреть на данные в столбцах электронной таблицы Excel, возможно получить лучшее представление о том, что имеющиеся данные содержат при помощи визуализации.

Например, легко видеть, что шаблон появляется из числовых данных, которые это дано в следующем графике:



**Рисунок 1.1: Простой пример визуализации данных**

Визуализация данных имеет много преимуществ, таких как:

- Сложные данные могут быть понятными
- Может быть создано простое визуальное представление выбросов, целевых аудиторий и будущих рынков
- Рассказывание истории может быть сделано с помощью панелей инструментов и анимаций
- Данные могут быть исследованы посредством интерактивной визуализации

## Теоретическая часть

### Пререкание данных (Data Wrangling)

Чтобы сделать выводы из визуализируемых данных, необходимо обработать данные и преобразовать их в самое лучшее представление. Это то, где **пререкание данных** используется.

Посмотрев на следующие данные процесса пререкания данных, легко понять, как точные и действенные данные могут быть получены для бизнес-аналитиков, чтобы продолжить работать. Данные «Обязательства Сотрудника» находятся в своей необработанной форме первоначально. Это импортируется как DataFrame и позже убирается. Убранные данные преобразовываются в соответствующие графики, от которых может быть смоделировано понимание. На основе этого понимания можно передать конечные результаты. Например, обязательство сотрудника может быть измерено на основе собранных необработанных данных из обзоров обратной связи, срока сотрудника, интервью выхода, индивидуальной встречи, и так далее. Эти данные убраны и превращены в графики на основе параметров, таких как направления, вера в лидерство и объем продвижений. Проценты, то есть, понимание, сгенерированное от графиков, помогают достигнуть результата, который должен определить меру мотивации сотрудника:

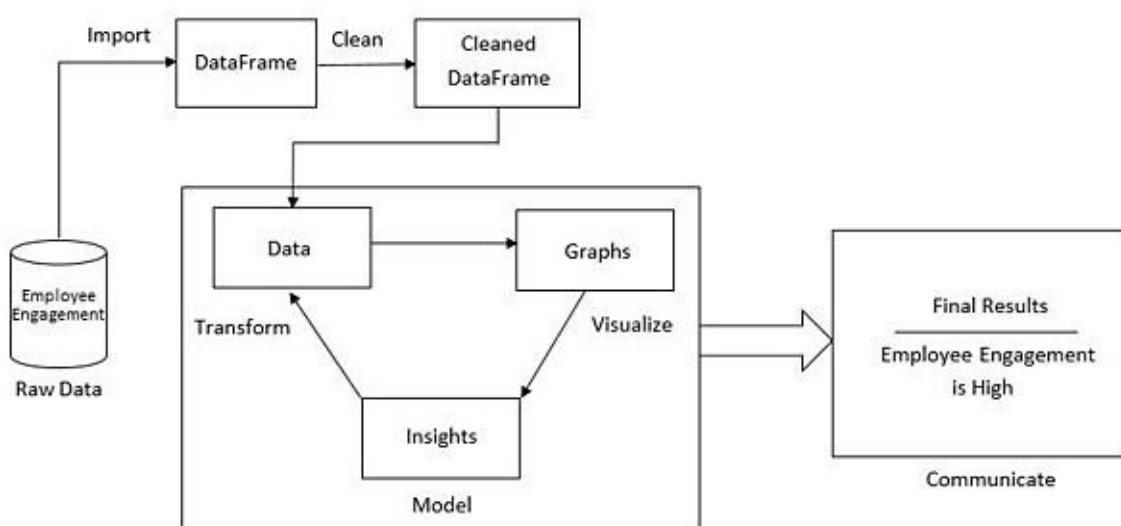


Рисунок 1.2: Процесс пререкания данных для «измерения»

## **мотивации сотрудника**

### **Инструменты и библиотеки для визуализации**

Существует несколько подходов к созданию визуализации данных. В зависимости от background-а возможно использовать инструмент декодирования, такой как **Tableau**, который дает возможность получить хорошее представление данных. Помимо Python, который будет использоваться в этой книге, **MATLAB** и **R** в большой степени используются в анализе данных.

Однако Python является самым популярным языком в промышленности. Его простота использования и скорость, на которой можно управлять и визуализировать данные, объединенные с доступностью многих библиотек, делают Python лучшим выбором.

### ***Примечание***

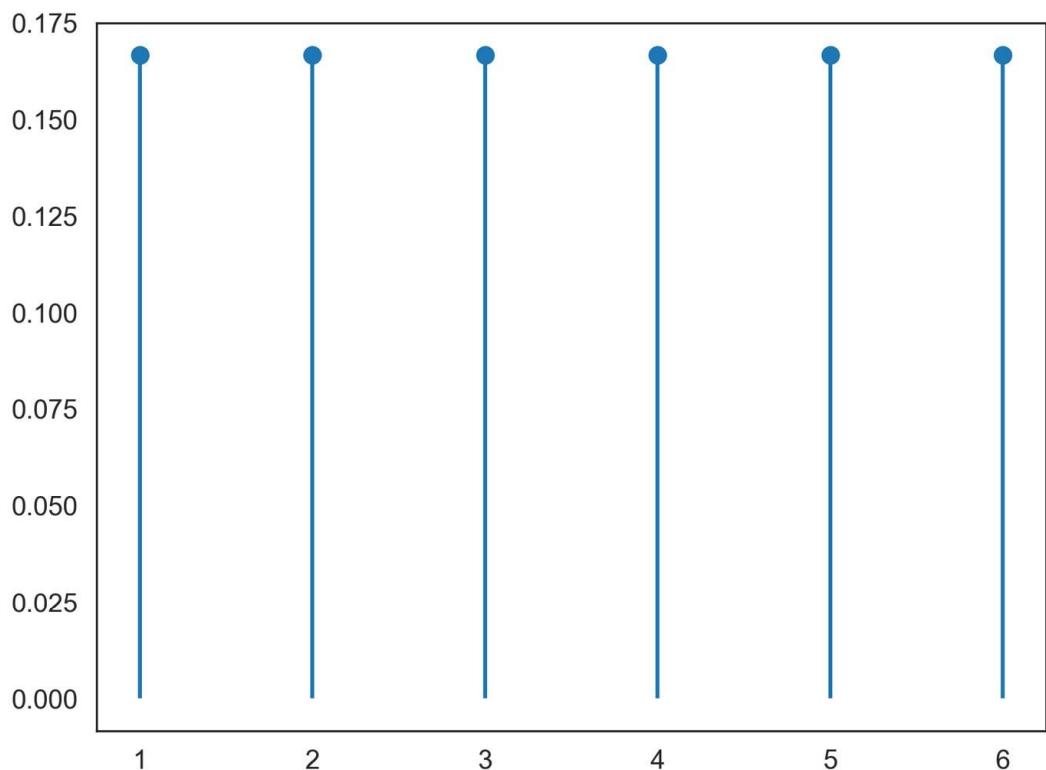
*MATLAB (<https://www.mathworks.com/products/matlab.html>), R (<https://www.r-project.org>), и Tableau (<https://www.tableau.com>) не часть этого курса, тут описаны только выделенные инструменты и библиотеки для Python.*

## **Обзор статистики**

**Статистика** является комбинацией анализа, набора, интерпретации и представления числовых данных. **Вероятность** является мерой вероятности, что событие будет иметь место и определяется количественно как число между 0 и 1.

**Распределение вероятностей** является функцией, которая обеспечивает вероятности для каждого возможного события. Распределение вероятностей часто используется для статистического анализа. Чем выше вероятность, тем более вероятно событие. Существует два типа распределения вероятностей, а именно, распределения дискретной вероятности и непрерывного распределения вероятностей.

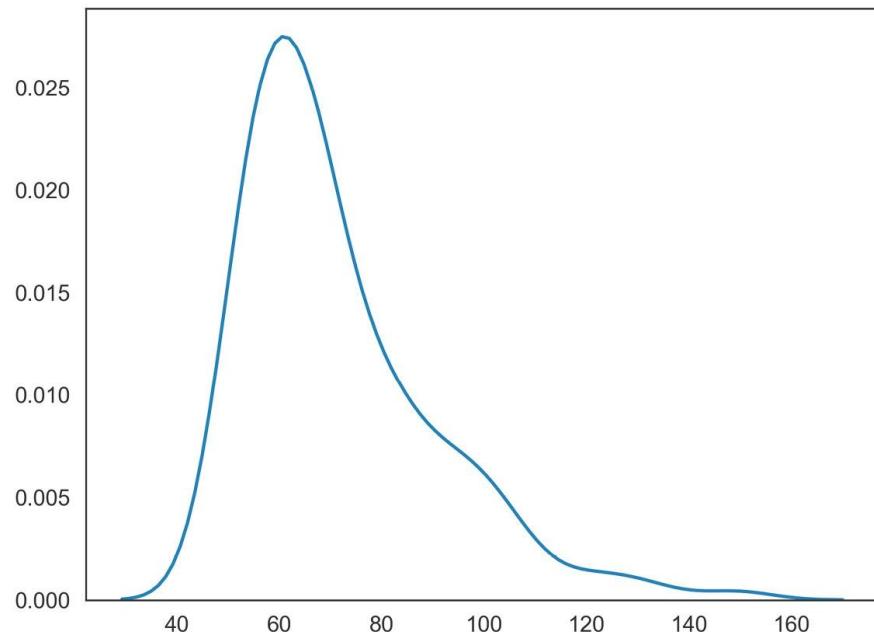
**Распределение дискретной вероятности** показывает все значения, что случайная переменная может взять, вместе с их вероятностью. Следующая схема иллюстрирует пример распределения дискретной вероятности. Если есть 6-сторонняя кость, можно прокрутить каждое число между 1 и 6. Есть шесть событий, которые могут произойти на основе прокрученного числа. Существует равная вероятность прокрутки любого из чисел, и отдельная вероятность любого этих шести появлений событий является  $1/6$ :



**Рисунок 1.3: Распределение дискретной вероятности для бросков кости**

**Непрерывное распределение вероятностей** определяет вероятности каждого возможного значения непрерывной случайной переменной. Следующая схема иллюстрирует пример непрерывного распределения вероятностей. Этот пример иллюстрирует распределение времени необходимое на дорогу домой. В большинстве случаев, приблизительно 60

минут необходимо, но иногда меньше времени необходимо, потому что нет никакого трафика, и иногда намного больше времени необходимо, если существуют пробки:



**Рисунок 1.4: Непрерывное распределение вероятностей в течение времени взятое для времени, необходимого на дорогу домой**

### **Меры центральной тенденции**

Меры центральной тенденции часто называют **средними числами** и описывают центральные или типичные значения для распределения вероятностей. Существует три вида средних чисел, которые будут обсуждены в этой главе:

- **Средний:** среднее арифметическое, которое вычисляется путем подведения всех измерений и деления суммы количеством наблюдений. Среднее вычисляется следующим образом:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$

**Медиана:** Это - среднее значение представленного набора данных.

Если будет четное число наблюдений, медиана будет средним числом

- о двух средних значений. Медиана менее подвержена выбросам по сравнению со средним, где выбросы являются отличными значениями в данных.

**Режим:** Последняя мера центральной тенденции, режим определяется как самое частое значение. Может быть больше чем один режим в случаях, где несколько значений являются одинаково частыми.

### Пример:

Игральная кость была прокручена десять раз, и были получены следующие числа: 4, 5, 4, 3, 4, 2, 1, 1, 2, и 1.

Среднее вычисляется путем подведения всех событий и деления их количеством наблюдений:  $(4+5+4+3+4+2+1+1+2+1)/10=2.7$ .

Для вычисления медианы броски кости должны быть заказаны согласно их значениям. Заказанные значения следующие: 1, 1, 1, 2, 2, 3, 4, 4, 4, 5. Так как есть четное число бросков кости, следует взять среднее число двух средних значений. Среднее число двух средних значений  $(2+3)/2=2.5$ .

Режимы равняются 1 и 4, так как они - два самых частых события.

## Меры дисперсии

**Дисперсия**, также названная **изменчивостью**, является степенью, до которой распределение вероятностей расширено или сжато.

Различные меры дисперсии следующие:

**Различие:** различие является математическим ожиданием отклонения

- о в квадрате от среднего. Это описывает, как далеко ряд чисел распространен из их среднего. Различие вычисляется следующим образом:

$$Var(X) = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

- о **Стандартное отклонение:** это - квадратный корень различия.
- о **Диапазон:** Вот в чем разница между самыми большими и самыми маленькими значениями в наборе данных.

- **Межквартильный размах:** Также названный **midspread** или **middle 50%**, это разница между 75-ми и 25-ми процентилями, или между верхними и более низкими квартилями.

## Корреляция

До сих пор были обсуждены только меры для единственных переменных. Напротив, **корреляция** описывает статистические отношения между двумя переменными:

В положительной корреляции обе переменные перемещаются в то же направление

В отрицательной корреляции переменные перемещаются в противоположные направления

В нулевой корреляции не связаны переменные

### Пример:

Съемщик хочет найти достойную квартиру в аренду, которая не является слишком дорогой по сравнению с другими квартирами, которые он нашел. Другие квартиры, которые он нашел на веб-сайте, оценены следующим образом: 700\$, 850\$, 1,500\$ и 750\$ в месяц:

о Среднее  $\frac{\$700 + \$850 + \$1500 + \$750}{4} = \$950$

о Медиана  $\frac{\$750 + \$850}{2} = \$800.$

Стандартное отклонение

о  $\sqrt{\frac{(\$700 - \$950)^2 + (\$850 - \$950)^2 + (\$1500 - \$950)^2 + (\$750 - \$950)^2}{4}} = \$322.10$

Диапазон  $\$1500 - \$700 = \$800$

- Медиана является лучшей статистической мерой в этом случае,
- о так как это менее подвержена выбросам (арендованная сумма 1,500\$).

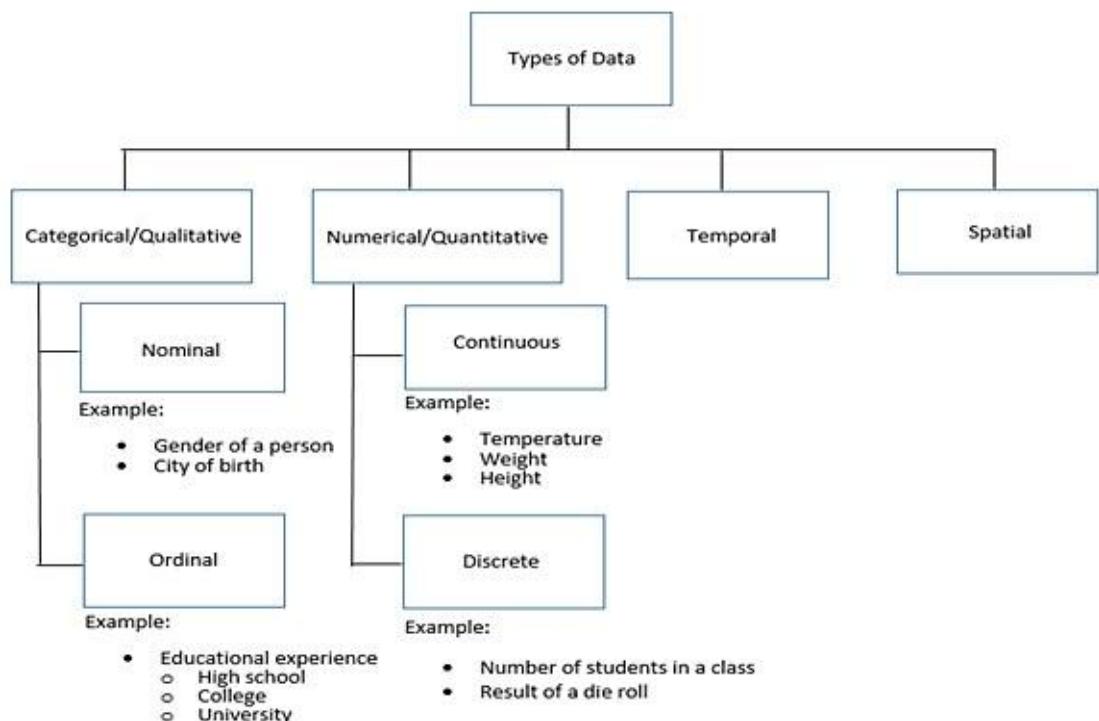
## Типы данных

Важно понять, с какими данными имеется дело так, чтобы можно было выбрать и правильную статистическую меру, и правильную визуализацию.

Следует разделять данные как категориальные/качественные и числовые/количественные. Категориальные данные описывают

характеристики, например, цвет объекта или пол человека. Можно далее разделить категориальные данные на номинальные и порядковые данные. В отличие от номинальных данных, порядковые данные имеют порядок. Числовые данные могут быть разделены на дискретные и текущие данные. Говоря о дискретных данных, если данные могут только иметь определенные значения, тогда как текущие данные могут принять любое значение (иногда ограничиваемый диапазоном).

Другой аспект для рассмотрения - имеют ли данные временный домен – другими словами, это связывается со временем, или это изменяется со временем? Если данные связываются с местоположением, могло бы быть интересно показать пространственные отношения:



**Рисунок 1.5: Классификация типов данных**

### Сводная статистика

В реальных приложениях приходится часто встречаться с огромными наборами данных, поэтому, **сводные статистические данные** используются для суммирования важных аспектов данных. Они необходимы для передачи больших объемов информации компактным и

простым способом.

Изучив меры центральной тенденции и дисперсии, которые являются сводной статистикой, важно знать, что меры центральной тенденции показывают центральную точку в ряде значений данных, тогда как меры дисперсионного показа - насколько распространены значения данных.

Следующая таблица показывает, как мера центральной тенденции подходит лучше всего для конкретного типа данных:

Data type	Ideal measure of central tendency
Nominal	Mode
Ordinal	Median
Numerical	Mean/Median

**Рисунок 1.6: Самые подходящие меры центральной тенденции для различных типов данных**

## NumPy

При обработке данных часто нужен способ работать с многомерными массивами. Необходимо применить некоторые основные математические и статистические операции на те данные. Именно здесь раскрывается NumPy в лучшем виде. Он оказывает поддержку для больших n-мерных массивов и является встроенной поддержкой многих высокоуровневых математических и статистических операций.

### *Примечание*

*Перед NumPy была библиотека под названием Numeric. Однако она больше не используется, поскольку подпись NumPy ndarray позволяет производительнее обрабатывать большие и высоко-размерные матрицы.*

Те ndarrays являются сущностью NumPy. Они - то, что делает его быстрее, чем использование встроенных списков Python. Кроме встроенного типа данных списка, ndarrays обеспечивают шаговое представление памяти (например, `int []` в Java). Так как они гомогенно вводятся, означая, что все элементы должны иметь тот же тип, шаг последователен, который приводит к меньшему количеству потерь памяти и лучшему времени доступа.

Шаг является количеством местоположений между началом двух смежных элементов в массиве. Они обычно измеряются в байтах или в единицах размера элементов массива. Шаг может быть больше или равняться размеру элемента, но не меньший, иначе это пересекло бы ячейку памяти следующего элемента.

### *Примечание*

*Следует помнить, что массивы NumPy имеют “определенный” тип данных. Это означает, что нельзя вставить строки в массив целого типа. NumPy главным образом используется с типами данных двойной точности.*

## **Упражнение 1: загрузка демонстрационного набора данных и вычисление среднего**

### **Примечание**

*Все упражнения и операции будут разработаны в Jupyter Notebook.*

В этом упражнении следует загрузить **normal\_distribution.csv** набор данных и вычислить среднюю из каждой строки и каждого столбца в ней:

1. Открыть **exercise01.ipynb Jupyter Notebook** из папки **Lesson01** для реализации этого упражнения.

Чтобы сделать это, необходимо перейти к пути этого файла. В терминале командной строки ввести **jupyter-lab**.

2. Теперь видно, что окно браузера открывается, показывая содержание каталога, которое было названо в предыдущей команте. Нажатие на **exercise01.ipynb** откроет notebook.

3. Notebook для Chapter01 должен теперь быть открыт и готов к изменениям. После краткого введения необходимо видеть ячейку, которая импортирует необходимые зависимости. В этом случае импортируйте **numpy** с псевдонимом:

```
#      importing the necessary dependencies
import numpy as np
```

3. Нужно искать ячейку, которая имеет комментарий, “loading the dataset”. В это место нужно вставить вызов метода **genfromtxt**. Этот метод помогает в загрузке данных из данного текста или .csv файла.

Полная строка должна быть похожей на это:

```
#  loading the dataset
dataset =
np.genfromtxt ('./data/normal_distribution.csv', delimiter= ',')
```

5. Если все работает как ожидалось, генерация должна пройти без любой ошибки или вывода. Простое выполнение ячейки, которая

возвращает значение, такое как ndarray GI, будет использовать форматирование Jupyter, которое выглядит лучше и, в большинстве случаев, отображает больше информации, чем использование **print**:

```
#      looking at the dataset  
dataset
```

Вывод предыдущего кода следующий:

```
array([[ 99.14931546, 104.03852715, 107.43534677, 97.85230675,  
       98.74986914, 98.80833412, 96.81964892, 98.56783189],  
      [ 92.02628776, 97.10439252, 99.32066924, 97.24584816,  
       92.9267508 , 92.65657752, 105.7197853 , 101.23162942],  
      [ 95.66253664, 95.17750125, 90.93318132, 110.18889465,  
       98.80084371, 105.95297652, 98.37481387, 106.54654286],  
      [ 91.37294597, 100.96781394, 100.40118279, 113.42090475,  
       105.48508838, 91.6604946 , 106.1472841 , 95.08715803],  
      [101.20862522, 103.5730309 , 100.28690912, 105.85269352,  
       93.37126331, 108.57980357, 100.79478953, 94.20019732],  
      [102.80387079, 98.29687616, 93.24376389, 97.24130034,  
       89.03452725, 96.2832753 , 104.60344836, 101.13442416],  
      [106.71751618, 102.97585605, 98.45723272, 100.72418901,  
       106.39798503, 95.46493436, 94.35373179, 106.83273763],  
      [ 96.02548256, 102.82360856, 106.47551845, 101.34745901,  
       102.45651798, 98.74767493, 97.57544275, 92.5748759 ],  
      [105.30350449, 92.87730812, 103.19258339, 104.40518318,  
       101.29326772, 100.85447132, 101.2226037 , 106.03868807],  
      [110.44484313, 93.87155456, 101.5363647 , 97.65393524,  
       92.75048583, 101.72074646, 96.96851209, 103.29147111],  
      [101.3514185 , 100.37372248, 106.6471081 , 100.61742813,  
       105.0320535 , 99.35999981, 98.87007532, 95.85284217],  
      [ 97.21315663, 107.02874163, 102.17642112, 96.74630281,
```

**Рисунок 1.7: первые строки normal\_distribution.csv файла**

6. Для получения быстрого обзора набора данных распечатаем его “форму”. Это даст вывод формы (строки, столбцы). Распечатайте форму с помощью команды **dataset.shape**. Можно также назвать строки как экземпляры и столбцы как функции. Это означает, что набор данных имеет 24 экземпляра и 8 функций:

```
#      printing the shape of our dataset  
dataset.shape
```

Вывод предыдущего кода следующий:

```
(24, 8)
```

**Рисунок 1.8: Форма набора данных**

7. Вычисление среднего является следующим шагом, как только загружен и

проверен набор данных.

К первой строке в **массиве NumPy** можно получить доступ путем простой индексации его с нулем, как это: **dataset [0]**. Как было упомянуто ранее, NumPy имеет некоторые встроенные функции для вычислений такого как среднее. Поэтому можно просто назвать **np.mean ()** и передать в строке набора данных для получения результата. Печатный вывод должен выглядеть следующим образом:

```
#      calculating the mean for the first row  
np.mean (dataset [0])
```

Вывод предыдущего кода: следующим образом:

100.177647525

#### **Рисунок 1.10: Представление элементов в первом столбце**

8. Можно также сделать то же самое для первого столбца при помощи **np.mean ()** в сочетании со столбцом, индексирующим **dataset [:, 0]**:

```
#      calculating the mean for the first column  
np.mean(dataset[:, 0])
```

Вывод предыдущего кода следующий:

99.76743510416668

#### **Рисунок 1.10: Представление элементов в первом столбце**

9. Если нужно получить среднее для каждой строки, агрегированной в списке, то можно использовать инструменты **оси(axis)** NumPy. Путем простой передачи параметра **оси** в **np.mean ()** вызов, можно определить размер, на котором будут агрегированы данные.  
**axis=0** горизонтален, и **axis=1** является вертикальным, поэтому если необходимо иметь результат для каждой строки, следует выбрать **axis=1**:

```
#      mean for each row  
np.mean (dataset, axis=1)
```

Вывод предыдущего кода следующий:

```
array([100.17764752, 97.27899259, 100.20466135, 100.56785907,
       100.98341406, 97.83018578, 101.49052285, 99.75332252,
       101.89845125, 99.77973914, 101.013081 , 100.54961696,
       98.48256886, 98.49816126, 101.85956927, 97.05201872,
       102.62147483, 101.21177037, 99.58777968, 98.96533534,
       103.85792812, 101.89050288, 99.07192574, 99.34233101])
```

**Рисунок 1.11: Представление элементов для каждой строки**

Если нужно иметь результат каждого столбца, следует выбрать **axis=0**.

```
# mean for each column
np.mean(dataset, axis=0)
```

Вывод предыдущего кода следующий:

```
array([ 99.7674351 , 99.61229127, 101.14584656, 101.8449316 ,
       99.04871791, 99.67838931, 99.7848489 , 100.44049274])
```

**Рисунок 1.12: Представление элементов для каждого столбца**

10. Как последняя задача этого упражнения, необходимо найти среднее из целой матрицы. Можно было бы подвести итог всех значений, которые были получены на предыдущих шагах, но NumPy позволяет просто сделать это вычисление:

```
# calculating the mean for the whole matrix
np.mean(dataset)
```

Вывод предыдущего кода следующий:

---

100.16536917390624

**Рисунок 1.13: Представление элементов для полного набора данных**

## **Действие 1: Использовать NumPy для вычисления среднего, медианы, различия и стандартного отклонения для данных чисел**

В этом действии будут использованы изученные навыки, нужно будет импортировать наборы данных и выполнять некоторые основные вычисления (среднее, медиана, различие и стандартное отклонение) для вычисления задач.

Для закрепления навыков и более плотного знакомства с NumPy необходимо выполнить следующие шаги:

1. Открыть **notebook activity01.ipynb Jupyter** из папки **Lesson01** для реализации этого действия.
2. Теперь, импортировать **numpy** в свой Jupyter Notebook и дать ему псевдоним **np**.
3. Загрузить **normal\_distribution.csv** набор данных при помощи **genfromtxt** метода **numpy**.
4. Удовостериться, что все работает, взглянув на ndarray.
5. Учитывая набор данных, использовать встроенные **numpy** методы.
6. Используя эти методы, сначала вычислить среднюю из третьей строки, последнего столбца и пересечения первых 3 строк и первых 3 столбцов.
7. Затем вычислить медиану последней строки, последних 3 столбцов и каждой строки.
8. Вычислить различие каждого столбца, пересечение последних 2 строк, и первых 2 столбцов.
9. Вычислить стандартное отклонение для набора данных.

## **Основные операции NumPy**

В этом разделе будут изучены основные операции NumPy, такие как индексация, разрезание, разделение и итерация и реализуем их в действии.

### **Индексация**

Индексация элементов в **массиве** NumPy, на высоком уровне, работает точно так же, как и со встроенными списками Python. Поэтому можно индексировать элементы в многомерных матрицах:

```
dataset [0] # индексирует единственный элемент в наиболее удаленном размере  
dataset [-1] # индексирует в обратном порядке в наиболее удаленном размере  
dataset [1, 1] # индексируют единственный элемент в двумерных данных  
dataset [-1,-1] # индексируют в обратном порядке в двумерных данных
```

## Разрезание

Разрезание также было адаптировано из списков Python. Способность легко нарезать части списков в новый ndarrays очень полезна при обработке больших объемов данных:

```
dataset [1:3] # строки 1 и 2  
dataset [:2:2] # 2x2 подмножество данных  
dataset [-1::-1] # последнюю строку с инвертированными элементами  
dataset [-5:-1:6:2] # делятся 4 строки, любой элемент к индексу 6
```

## Разделение

Разделение данных может быть полезным во многих ситуациях от графического изображения только половины данных временного ряда до разделению данных тестирования и обучающих данных для алгоритмов машинного обучения.

Существует два способа разделить данные, горизонтально и вертикально.

Горизонтальное разделение может быть сделано с **hsplit** методом.

Вертикальное разделение может быть сделано с **vsplit** методом:

```
np.hsplit (dataset, (3)) # разделение горизонтально в 3 равных
```

*списках*

```
np.vsplit (dataset, (2)) # разделение вертикально в 2 равных  
списках
```

## Итерация

Выполнение итерации структур данных NumPy, ndarrays, также возможно.

Это переступание через целый список данных один за другим, посещая каждый элемент в ndarray однажды. Рассмотрим, что у них может быть несколько размеров, от чего индексация становится более сложной.

**nditer** является многомерным объектом итератора, который выполняет итерации по данному количеству массивов:

```
# iterating over whole dataset (each value in each row)  
for x in np.nditer (dataset):  
    print(x)
```

**ndenumerate** даст этот индекс, таким образом возвращая **(0, 1)** для второго значения в первой строке:

```
# iterating over whole dataset with indices matching the position in the  
dataset  
for index, value in np.ndenumerate (dataset):  
    print (index, value)
```

## Действие 2: индексация, разрезание, разделение, и итерация

В этом действии будут использованы функции NumPy, чтобы индексировать, нарезать, разделить, и выполнить итерации ndarrays для консолидации того, что было изучено. Клиент хочет, чтобы было доказано, что набор данных распределяется вокруг среднего значения 100. Для этого необходимо:

1. Открыть **activity02.ipynb Jupyter Notebook** из папки **Lesson01** для реализации этого действия.
2. Теперь, импортировать **numpy** в свой Jupyter Notebook и дать ему псевдоним **np**.
3. Загрузить **normal\_distribution.csv** набора данных, используя NumPy.  
Удовствериться, что все работает, взглянув на ndarray, как в предыдущем действии. Следуйте описанию задачи в notebook.
4. С загруженным набором данных использовать ранее обсужденную функцию индексации, чтобы индексировать вторую строку набора данных (вторая строка), индексировать последний элемент набора данных (последняя строка), индексировать первое значение второй строки (вторая строка, сначала оценить), и индексировать последнее значение предпоследней строки (использующий объединенный доступ).
5. Создание подсписков входа требует разрезания. Нарежьте пересечение четырех элементов ( $2 \times 2$ ) первых двух строк и первых двух столбцов, выберите каждый второй элемент пятой строки и инвертируйте порядок записи, выбрав первые две строки в обратном порядке.
6. Если нужен меньший набор данных, можно также использовать разделение для эффективного деления набора данных. Используйте это понятие, чтобы разделить набор данных горизонтально на три равных части и разделить набор данных вертикально на индексе 2.
7. Последняя задача в этом действии будет выполнять итерации по полному набору данных. Используйте ранее изученные методы

для итерации по целому набору данных с индексами, соответствующими положению в наборе данных.

## Усовершенствованные операции NumPy

В этом разделе будут описаны операции NumPy, такие как фильтрация, сортировка, объединение и изменение, и реализуем их в действии.

### Фильтрация

Фильтрация является очень мощным инструментом, который может использоваться для чистки данных, если нужно избежать значений изолированной части. Также полезно для того, чтобы получить лучшее понимание данных.

В дополнение к **dataset [dataset > 10]** нотация стенографии, можно использовать встроенный метод **извлечения** NumPy, который делает то же самое с помощью различной нотации, но дает больший контроль с более сложными примерами.

Если необходимо только извлечь индексы значений, которые соответствуют заданному условию, следует использовать встроенный метод **where**. Например, **np.where (dataset > 5)** возвратит список индексов значений от начального набора данных, которые больше, чем 5:

```
dataset [dataset > 10] # значения, большие, чем 10  
np.extract ((dataset <3), dataset) # альтернатива – значения,  
меньшие, чем 3  
dataset [(dataset > 5) & (dataset <10)] # оценивает большие 5 и  
меньшие 10  
np.where (dataset > 5) # индексы значений, большие, чем 5 (строки  
и столбцы)
```

### Сортировка

Сортировка каждой строки набора данных может быть действительно

полезной. Используя NumPy, возможно также отсортировать столбцы.

Кроме того, **argsort** дает возможность получить список индексов, которые привели бы к отсортированному списку:

```
np.sort (dataset) # значения отсортированы на последней оси  
np.sort (dataset, axis=0) # значения, отсортированные на оси 0  
np.argsort (dataset) # индексы значений в отсортированном списке
```

## Объединение

Объединение строк и столбцов на существующий набор данных может быть полезной, если есть два набора данных одного и того же размера, сохраненного в различные файлы.

Имея два набора данных, следует использовать **vstack** для “складывания” **dataset\_1** сверху **dataset\_2**, который даст объединенный набор данных со всеми строками от **dataset\_1**, сопровождаемого всеми строками от **dataset\_2**. Используя **hstack**, выполняется складывание наборов данных “друг с другом”, подразумевая, что элементы от первой строки **dataset\_1** будут сопровождаться элементами первой строки **dataset\_2**. Это будет применено к каждой строке:

```
np.vstack ([dataset_1, dataset_2]) # комбинируют наборы  
данных вертикально  
np.hstack ([dataset_1, dataset_2]) # комбинируют наборы  
данных горизонтально  
np.stack ([dataset_1, dataset_2], axis=0) # комбинируют  
наборы данных на оси 0
```

## Изменение

Изменение может быть крайне важно для некоторых алгоритмов. В зависимости от природы данных это могло бы помочь уменьшить размерность для создания лучшей визуализации:

```
dataset.reshape (-1, 2) # изменяют набор данных к двум столбцам x
```

*строки*

*np.reshape (dataset, (1,-1)) # изменяют набор данных к одной строке  
x столбцы*

Здесь, **-1** - неизвестное измерение, которое NumPy определяет автоматически. NumPy сначала выяснит длину любого данного массива и остальных измерений и таким образом удостоверится, что это удовлетворяет данному упомянутому стандарту.

### **Действие 3: фильтрация, сортировка, объединение, и изменение**

Это последнее действие для NumPy, которое обеспечивает некоторые более сложные задачи закрепления приобретенных знаний. Это также объединит большинство ранее изученных методов как резюме. Для этого необходимо выполнить следующее:

1. Открыть **activity03.ipynb Jupyter notebook** из папки **Lesson01**.
2. NumPy будет единственной необходимой зависимостью для этого действия, поэтому необходимо удостовериться, что библиотека импортирована.
3. Снова загрузить **normal\_distribution.csv** набор данных NumPy.  
Удостовериться, что все работает, взглянув на ndarray.
4. После загрузки набора данных использовать ранее изученные функции фильтрации для фильтрации для значений, которые больше, чем 105, фильтр для значений, которые находятся между 90 - 95, и получают индексы значений, которые имеют дельту (различие) меньше чем 1 - 100.
5. Сортировка данных является важной функцией при попытке отобразить данные по порядковому столбцу типа данных. Следует использовать **сортировку** для сортировки значений для каждой строки, значения вида для каждого столбца, чтобы получить индексы положений для каждой строки и получить 3 самых маленьких значения для каждой строки (остающиеся значения не отсортированы).
6. Используя данные разделения из *Activity02* этой главы, можно также использовать ранее изученные сочетающие функции, чтобы добавить вторую половину первого столбца назад, добавить второй столбец к объединенному набору данных и добавить третий столбец к объединенному набору данных.
7. Использовать изменяющиеся функции, чтобы изменить набор данных в одномерном списке со всеми значениями и изменить набор данных в матрицу только с двумя столбцами.

Затем, будет изучаться библиотека pandas, которая принесет несколько преимуществ при работе с данными, которые более сложны, чем простые многомерные числовые данные. Pandas также поддерживают различные типы данных в наборах данных, подразумевая, что могут быть столбцы, которые содержат строки и другие, которые имеют числа.

Сам NumPy, как было рассмотрено, имеет некоторые действительно мощные инструменты. Некоторые из них еще более мощны, когда объединены с pandas.

## pandas

Библиотека Python **pandas** предлагает структуры данных и методы для управления различными типами данных, такими как числовые и временные. Эти операции просты в использовании и высоко оптимизированы по производительности.

Форматы данных, такие как **CSV**, **JSON** и базы данных могут использоваться для создания **DataFrame**. DataFrames являются внутренним представлением данных и очень похожи на таблицы, но мощнее. Импорт и чтение файлов и данных в оперативной памяти абстрагированы в удобный для пользователя интерфейс. Когда дело доходит до обработки недостающих данных pandas предоставляют встроенные решения, чтобы очистить и увеличить данные, подразумевая, что это заполняет отсутствующие значения с рыночной стоимостью.

Интегрированная индексация и основанное на маркировке разрезание в сочетании с индексацией (что уже было изучено с NumPy) делают данные простыми для обработки. Более сложные методы, такие как **изменение**, **поворот** и **расплавление** данных, вместе с возможностью легкого **присоединения** и **слияния** данных, обеспечивают мощный функционал для обработки данных.

Если работа происходит с **данными временного ряда**, операции, такие как **генерация диапазона дат**, **преобразование частоты**, и **движущаяся**

**оконная статистика** могут обеспечить усовершенствованный интерфейс для пререкания.

### **Примечание**

*Инструкции по установке pandas могут быть найдены здесь:*

<https://pandas.pydata.org/>.

## **Преимущества pandas перед NumPy**

Некоторые преимущества pandas:

**Высокий уровень абстракции:** у панд более высокий уровень абстракции, чем у NumPy, который дает ему более простой интерфейс для пользователей для взаимодействия с. Это

- абстрагирует некоторые более сложные понятия и облегчает использование и понимание.
- **Интуитивно понятный:** Много методов, таких как присоединение, выбор, и загрузка файлов, применимы без большой интуиции и не вникая в большую часть мощной библиотеки pandas.

**Быстрее обработка:** внутреннее представление DataFrames позволяет более быстро обрабатывать некоторые операции. Конечно, это всегда зависит от данных и их структуры.

- **Легкий дизайн DataFrames:** DataFrames разработаны для операций с и на большие наборы данных.

## **Недостатки pandas**

Некоторые недостатки pandas:

**Менее применимый:** из-за его более высокой абстракции, обычно менее применимо, чем NumPy. Особенно, когда используется за пределами объема абстракции, операции быстро становятся очень сложными и тяжело выполнимыми.

**Больше дискового пространства:** из-за упомянутого внутреннего

- представления DataFrame панды занимают больше дискового пространства взамен на более производительное выполнение, использование памяти для сложных операций может быть проблемой.

**Проблемы производительности:** Особенно при выполнении тяжелых соединений, который не рекомендуется выполнять, неправильное использование памяти может стать основополагающим и привести к проблемам производительности.

**Скрытая сложность:** сравнительно простой интерфейс имеет другую сторону также. Менее опытные пользователи часто склонны

- злоупотреблять методами и выполнять их несколько раз вместо того, чтобы снова использовать то, что они уже вычислили. Эта скрытая сложность заставляет пользователей думать, что сами операции просты, что на самом деле не так.

### ***Примечание***

*Всегда следует думать о том, как правильно настроить рабочие процессы вместо того, чтобы чрезмерно использовать операции.*

## **Упражнение 2: загрузка демонстрационного набора данных и вычисление среднего**

В этом упражнении будет использован **world\_population.csv** набор данных для вычисления среднего из некоторых строк и столбцов. Этот набор данных содержит ежегодную плотность населения для каждой страны. Необходимо использовать pandas для получения некоторого действительно быстрого и легкого понимания:

1. Открыть **exercise02.ipynb Jupyter Notebook** из папки **Lesson01** для реализации этого упражнения.
2. Импортировать библиотеки панд:

```
# импорт необходимых зависимостей
```

```
import pandas as pd
```

3. После импорта pandas использовать **read\_csv** метод для загрузки упомянутого набора данных. Использовать первый столбец, содержащий названия страны, как индекс. Использовать **index\_col** параметр для этого. Полная строка должна быть похожей на это:

```
# загрузка набора данных
```

```
dataset = pd.read_csv ('./data/world_population.csv',
```

```
index_col=0)
```

4. Как прежде, требуется взглянуть на данные, импортированные путем простого написания имени набора данных в следующей ячейке. Pandas используют структуру данных под названием **DataFrames**. Нужно распечатать некоторые строки, чтобы не заполнять экран. Pandas **DataFrames** имеют два метода, **head ()** и **tail ()**. Оба берут число, **n**, в качестве параметра, который описывает, сколько строк должно быть возвращено:

### **Примечание**

*Значение, такое как DataFrame, будет использовать форматирование Jupyter, которое выглядит лучше и, в большинстве случаев, отображает большие информации, чем использование print.*

```
# рассмотрение набора данных
```

```
dataset.head()
```

Вывод предыдущего кода следующий:

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965
Aruba	ABW	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	307.972222	312.366667	314.983333	316.827778	318.666667
Andorra	AND	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	30.587234	32.714894	34.914894	37.170213	39.470213
Afghanistan	AFG	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	14.038148	14.312061	14.599692	14.901579	15.218206
Angola	AGO	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	4.305195	4.384299	4.464433	4.544558	4.624228
Albania	ALB	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	60.576642	62.456898	64.329234	66.209307	68.058066

5 rows × 60 columns

**Рисунок 1.14: Первые пять строк набора данных**

5. Для получения быстрого обзора набора данных следует распечатать его форму.

Это даст вывод в форме (**строки, столбцы**). Распечатать форму с помощью команды **dataset.shape**. Это работает точно так же, как и с NumPy ndarrays:

```
# печать формы набора данных
```

```
dataset.shape
```

Вывод предыдущего кода следующий:

(264, 60)

**Рисунок 1.15: Форма набора данных**

6. Вычисление среднего является следующим шагом, как только был

загружен и проверен набор данных.

Индексация строк работает немного по-другому. На данный момент нужно только индексировать столбец с 1961 годом.

У DataFrames есть встроенные функции для вычислений **среднего**.

Это означает, что можно просто назвать **dataset.mean()** для получения результата.

Печатный вывод должен вызваться следующим образом:

```
# вычисление среднего для 1961 столбца  
dataset[“1961”].mean()
```

Вывод предыдущего кода следующий:

---

176.91514132840538

### Рисунок 1.16: Средний из элементов в 1961 столбце

7. Только для наблюдения различия в плотности населения за эти годы следует сделать то же самое со столбцом на 2015 год (население, более чем удвоенное в данном диапазоне времени):

```
# вычисление среднего для 2015 столбца  
# dataset[“2015”].mean()
```

Вывод предыдущего кода следующий:

---

368.7066010400187

### Рисунок 1.17: Средний из элементов в 2015 столбце

8. Если требуется получить среднее для каждой страны (строка), можно использовать инструменты **axis (оси)** панд. Путем простой передачи параметра **axis** в **dataset.mean()** вызов, определить размер, на котором будут агрегированы данные.

**axis=0** горизонтален (для каждого столбца), и **axis=1** является вертикальным (на строку), поэтому если нужно иметь результат для каждой строки, следует выбрать **axis=1**. Так как имеется 264 строки в

наборе данных, следует ограничить сумму возвращенных стран.

10. Как было обсуждено ранее, можно использовать **head (10)** и **tail (10)** методы с параметром 10:

```
# среднее для каждой страны (строка)
```

```
dataset.mean (axis=1).head (10)
```

Вывод предыдущего кода следующий:

```
Country Name
Aruba           413.944949
Andorra         106.838839
Afghanistan    25.373379
Angola          9.649583
Albania         99.159197
Arab World      16.118586
United Arab Emirates 31.321721
Argentina       11.634028
Armenia         103.415539
American Samoa 211.855636
dtype: float64
```

**Рисунок 1.18: Среднее из элементов в первых 10 странах (строки)**

Код для хвоста:

```
# среднее для каждого столбца
```

```
dataset.mean (axis=0).tail (10)
```

Вывод предыдущего кода следующий:

```
2007    331.995474
2008    338.688417
2009    343.649206
2010    347.967029
2011    351.942027
2012    357.787305
2013    360.985726
2014    364.849194
2015    368.706601
2016        NaN
dtype: float64
```

**Рисунок 1.19: Среднее из элементов в течение прошлых 10 лет (столбцы)**

9. Последняя задача этого упражнения состоит в том, чтобы вычислить средний из всего DataFrame. По умолчанию используется **axis=0**, что

означает, что будет получен тот же результат как прежде:

```
# вычисление среднего для целой матрицы  
dataset.mean()
```

Вывод предыдущего кода следующий:

1960	NaN
1961	176.915141
1962	180.703231
1963	184.572413
1964	188.461797
1965	192.412363
1966	196.145042
1967	200.118063
1968	203.879464
1969	207.336102
1970	210.607871
1971	213.489694
1972	215.998475
1973	218.438708
1974	220.621210
1975	223.046375
1976	224.960258
1977	227.006734
1978	229.187306
1979	232.510772
1980	236.185357
1981	240.789508
1982	246.175178
1983	251.342389
1984	256.647822

**Рисунок 1.20:** Среднее из элементов для каждого столбца

## **Действие 4: Использовать pandas, чтобы вычислить среднее, медиану, и различие для данных чисел**

В этом действии следует использовать ранее освоенные навыки импорта наборов данных и выполнения некоторых основных вычислений, применить их для решения задачи.

Для того что закрепить полученные навыки и ближе познакомиться с pandas, следует:

1. Открыть Jupyter Notebook **activity04.ipynb** из папки **Lesson01** для реализации этого действия.
2. Импортировать библиотеку в начале действия, чтобы быть в состоянии использовать их.
3. Загрузить **world\_population.csv** набор данных с помощью **read\_csv** метода панд.
4. Учитывая набор данных, использовать встроенные методы панд для вычисления среднего из третьей строки, последней строки и страны Германия.
5. Затем вычислить медиану последней строки, последних 3 строк и первых 10 стран.
6. Наконец, вычислить различие последних 5 столбцов.

## **Основные операции pandas**

Этот раздел посвящен изучению основных операций панд, таких как индексация, разрезание и итерация.

### **Индексация**

Индексация с пандами немного сложнее, чем с NumPy. Можно получить доступ к столбцу с одиночной скобкой. Для использования индексов строк для доступа к ним нужен **iloc** метод. Если нужно получить доступ к ним **index\_col** (который был установлен в вызове **read\_csv**), следует использовать метод **loc**:

```
dataset[“2000”] # индексирует столбец 2000 года  
dataset.iloc[-1] # индексируют последнюю строку  
dataset.loc[“Германия”] # индексируют строку с индексом Германия  
dataset[[“2015”]].loc[[“Германия”]] # индексирует строку Германия и  
столбец 2015
```

## **Разрезание**

Разрезание с пандами работает мощнее. Можно использовать синтаксис разрезания значения по умолчанию, который уже был рассмотрен с NumPy или использовать мультивыбор. Если требуется нарезать различные строки или столбцы по имени, можно просто передать список в скобку:

```
dataset.iloc [0:10] # часть первых 10 строк  
dataset.loc [[“Германия”, “Индия”]] # часть строк Германия и Индия  
# подмножество Германии и Индии с годами 1970/90  
dataset.loc [[“Германия”, “Индия”]] [[“1970”, “1990”]]
```

## **Итерация**

Итерация DataFrames также возможна. Считается, что у них может быть несколько размеров и dtypes. Выполнение итераций по каждой строке должно быть сделано отдельно:

```
# итерация целого набора данных  
for index, row in dataset.iterrows ():  
print (index, row)
```

## **Ряд**

Ряд панд является одномерным маркированным массивом, который способен содержать любого типа данные. Возможно создать ряд путем загрузки наборов данных из .csv файла, электронной таблицы Excel или базы данных SQL. Существует много различных способов создать их. Например:

Массивы NumPy:

```
# импорт
pandas
import pandas
as pd
# импорт
numpy
import numpy
as np
# создание массива numpy
numarr = np.array(['p', 'y', 't', 'h', 'o', 'n'])
ser = pd.Series(numarr)
print(ser)
```

о

списки pandas

```
# импорт
pandas
import pandas
as pd
# создание списка панд
plist = ['p', 'y', 't', 'h', 'o', 'n']
ser = pd.Series(plist)
print(ser)
```

## **Действие 5: Индексация, разрезание и итерация с использованием pandas**

В этом действии используются ранее обсужденные функции панд, чтобы индексировать, нарезать, и выполнить итерации DataFrames с помощью ряда pandas. Для получения некоторого понимания набора данных необходимо явно индексировать, нарезать, и выполнить итерации данных. Например, можно сравнить несколько стран с точки зрения роста плотности населения. Требуется отобразить плотность населения Германии, Сингапура, Соединенных Штатов и Индии в течение лет 1970, 1990, и 2010, для этого нужно:

1. Открыть **activity05.ipynb Jupyter notebook** из папки **Lesson01** для реализации этого действия.
2. Прежде, чем загрузить набор данных, необходимо импортировать pandas. Использовать псевдоним **pd**, чтобы обращаться к пандам.
3. Загрузить **world\_population.csv** набор данных с помощью панд. Удовестовериться, что все работает, взглянув на DataFrames.
4. С загруженным набором данных использовать ранее обсужденную функцию индексации для индексации строки для США, предпоследней строки, столбца 2000 года как ряда и плотности населения для Индии в 2000.
5. Создание подсписков набора данных требует разрезания.

Использовать это понятие, чтобы нарезать страны в строках 2 - 5, нарезать страны Германия, Сингапур, Соединенные Штаты, и Индия и часть Германия, Сингапур, Соединенные Штаты и Индия с их плотностью населения в течение лет 1970, 1990, и 2010.

6. Последняя задача в этом действии будет выполнять итерации по первым трем странам в наборе данных. Использовать ранее изученный итеративный метод, чтобы выполнить итерации по целому набору данных и распечатать имя, код страны и население в течение 1970, 1990, и 2010.

## **Усовершенствованные операции pandas**

В этом разделе будут изучены усовершенствованные операции панд, такие как фильтрация, сортировка, и изменение, реализуем их в действии.

### **Фильтрация**

Фильтрация pandas имеет более высокоуровневый интерфейс, чем NumPy. Можно все еще использовать “простую”, основанную на скобках **условную фильтрацию**. Однако можно использовать более сложные запросы, например, строки фильтра на основе регулярного выражения:

```
dataset.filter (items = ["1990"]) # только столбец 1994  
dataset [(dataset ["1990"] <10)] # страны с плотность <10 в  
1999  
dataset.filter (like = "8", axis=1) # годы, содержащие 8  
dataset.filter (regex = "a$", axis=0) # страны, заканчивающиеся а
```

### **Сортировка**

Сортировка каждой строки или столбца на основе данной строки, или столбца поможет получить лучшее понимание своих данных и найти рейтинг данного набора данных. С пандами сделать это довольно легко. Сортировка в порядке по возрастанию и порядке по убыванию может быть сделана с помощью параметра, известного как **ascending (возрастание)**. Конечно, можно сделать более сложную сортировку путем вставки больше чем одного значения в = [] список. Данные подстраиваются к типу данных первого значения:

```
dataset.sort_values (= ["1999"]) # значения, отсортированные к 1999  
набор данных sort_values (= ["1994"], ascending=False) # значения,  
отсортированные к 1999 в порядке убывания
```

### **Изменение**

Изменение может быть крайне важно для более легкой визуализации и

алгоритмов. Однако в зависимости от данных, использовать это может быть действительно тяжело:

```
dataset.pivot (index = [“1999”] * len (dataset),  
columns = “Country Code”, values = “1999”)
```

### **Примечание**

*Изменение является очень сложной темой. Если есть желание погрузиться глубже в эту тему, это – полезный ресурс для начала работы:*

[https://bit.ly/2SjWzaB.](https://bit.ly/2SjWzaB)

## Действие 6: фильтрация, сортировка и изменение

Это последнее действие для панд, оно обеспечивает решение некоторых более сложных задач и также комбинирует большинство методов, изученных ранее как резюме. После этого действия студенты должны быть в состоянии использовать pandas, кодировать и понимать его логику:

1. Открыть **activity06.ipynb** Jupyter notebook из папки **Lesson01** для реализации этого действия. Вернуться к введение этой главы для получения инструкций относительно того, как открыть Jupyter Notebook in JupyterLab.
2. Фильтрация, сортировка и изменение являются всеми методами панд. Прежде чем использовать их, необходимо импортировать pandas.
3. Снова загрузить **world\_population.csv** набор данных с помощью панд и удостовериться, что все работает, взглянув на DataFrames:

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963
Aruba	ABW	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	307.972222	312.366667	314.983333
Andorra	AND	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	30.587234	32.714894	34.914894

**Рисунок 1.21: Рассмотрение первых двух столбцов из набора данных**

4. После загрузки набора данных использовать ранее изученные функции фильтрации для получения 1961, 2000, и 2015 годов и всех стран, которые имели большую плотность населения, чем 500 в 2000 году. Кроме того, использовать фильтрацию для получения нового набора данных, содержащих 2000 год и позже, страны, которые начинаются с

- А и стран, содержащих слово “земля”.
5. Сортировка данных является важной функцией при попытке отобразить данные по порядковому столбцу типа данных. Использовать операцию сортировки для сортировки значений в порядке возрастания к 1961, в порядке возрастания к 2015, и в порядке убывания к 2015.
  6. Если входной набор данных не удовлетворяет потребностям, следует изменить его. Использовать изменяющие функции для изменения набора данных к 2015 как строка и код страны как столбцы.

## Сводка

NumPy и pandas являются существенными инструментами для пререкания данных. Их удобные для пользователя интерфейсы и производительная реализация делают обработку данных легкой. Даже при том, что они обеспечивают только немного понимания наборов данных, они крайне ценные для пререкания, увеличения и очистки наборов данных. Навыки работы с этими библиотеками улучшат качество визуализации.

В этой главе были изучены основы NumPy и pandas, понятия статистики. Даже при том, что статистические покрытые понятия являются очень простыми, они необходимы для обогащения визуализации информацией, которая, в большинстве случаев, непосредственно не предоставляется в наборах данных. Этот практический опыт поможет реализовать упражнения и операции в следующих главах.

## Практическая часть

### Действие 1: Использовать NumPy для вычисления среднего, медианы, различия и стандартного отклонения для данных чисел

#### Решение:

Требуется использовать NumPy для вычисления среднего, среднего, различия и стандартного отклонения:

1. Импортировать необходимые библиотеки:

```
# импорт необходимых зависимостей
```

```
import numpy as np
```

```
In [1]: import numpy as np
```

Рисунок 1.1: Импорт numpy

2. Загрузить **normal\_distribution.csv** набор данных при помощи **genfromtxt** метода NumPy:

```
# загрузка набора данных
```

```
dataset = np.genfromtxt('./data/normal_distribution.csv', delimiter=',')
```

3. Во-первых, необходимо распечатать подмножество первых двух строк набора данных:

```
# рассмотрение первых двух строк набора данных набора данных
```

```
dataset[0:2]
```

Вывод предыдущего кода следующий:

```
In [3]: dataset = np.genfromtxt('normal_distribution.csv', delimiter=',')
dataset[0:2]
```

```
Out[3]: array([[ 99.14931546, 104.03852715, 107.43534677, 97.85230675,
   98.74986914, 98.80833412, 96.81964892, 98.56783189],
   [ 92.02628776, 97.10439252, 99.32066924, 97.24584816,
   92.9267508 , 92.65657752, 105.7197853 , 101.23162942]])
```

Рисунок 1.2: Первые две строки набора данных

4. Как только известно, что набор данных был успешно загружен, можно начать решать первую задачу, которая вычисляет среднее из третьей строки.

К третьей строке можно получить доступ путем индексации **dataset [2]**:

# вычисление среднего из третьей строки

*np.mean (набор данных [2])*

Вывод предыдущего кода следующий:

In [8]: np.mean(dataset[2])

Out[8]: 100.20466135250001

**Рисунок 1.3: Среднее третьей строки**

5. Последний элемент ndarray может быть индексирован тем же путем. **dataset [-1]** даст последний столбец каждой строки:

# вычисление среднего из последнего столбца

*np.mean (dataset [-1])*

Вывод предыдущего кода следующий:

In [9]: np.mean (dataset [-1])

Out[9]: 100.20115344206522

**Рисунок 1.4: Среднее последнего столбца**

6. Дважды индексирующий механизм NumPy дает хороший интерфейс для извлечения подвыбора. В этой задаче необходимо получить субматрицу первых трех элементов каждой строки первых трех столбцов:

# вычисление среднего из пересечения первых 3 строк и первых 3 столбцов

*np.mean (dataset [0:3, 0:3])*

Вывод предыдущего кода следующий:

In [10]: np.mean (dataset [0:3, 0:3])

Out[10]: 97.87197312333333

### Рисунок 1.5: Среднее пересечения

7. Перемещаясь в следующий набор задач, которые предполагают поиск медианы, можно увидеть, что API последователен между различными методами.:

```
# вычисление медианы последней строки  
# np.median (dataset [-1])
```

Вывод предыдущего кода следующий:

```
In [11]: np.median (dataset [-1])  
Out[11]: 99.18748092
```

### Рисунок 1.6: Медиана последней строки

8. Индексирующий реверс также работает при определении диапазона. Так, если необходимо получить последние три столбца, можно использовать **dataset [:-3:]**:

```
# вычисление медианы последних 3 столбцов  
np.median (dataset [:-3:])
```

Вывод предыдущего кода следующий:

```
In [12]: np.median (dataset [:-3:])  
Out[12]: 100.38337885
```

### Рисунок 1.7: Медиана последних 3 столбцов

9. Как было видно в предыдущем осуществлении, можно агрегировать значения вдоль **оси**. Если нужно вычислить строки, следует использовать **axis=1**:

```
# вычисление медианы каждой строки  
np.median (dataset, axis=1)
```

Вывод предыдущего кода следующий:

```
In [13]: np.median (dataset, axis=1)

Out[13]: array([ 98.77910163,  97.17512034,  98.58782879, 100.68449836,
   101.00170737,  97.76908825, 101.85002253, 100.04756697,
   102.24292555,  99.59514997, 100.4955753 ,  99.8860714 ,
   99.00647994,  98.67276177, 102.44376222,  96.61933565,
  104.0968893 , 100.72023043,  98.70877396,  99.75008654,
 104.89344428, 101.00634942,  98.30543801,  99.18748092])
```

**Рисунок 1.8: Использование axis для вычисления медианы каждой строки**

10. Последний метод, который будет здесь изучен – различие (variance). Снова NumPy предоставляет API, который делает выполнение этого действия простым в реализации. Для вычисления различия для каждого столбца следует использовать **axis 0**:

```
# вычисление различия каждого столбца
np.var (dataset, axis=0)
```

Вывод предыдущего кода следующий:

```
In [14]: np.var (dataset, axis=0)

Out[14]: array([23.64757465, 29.78886109, 20.50542011, 26.03204443, 28.38853175,
   19.09960817, 17.67291174, 16.17923204])
```

**Рисунок 1.9: Различие через каждый столбец**

11. Теперь для подмножества данных (2x2).

#### **Примечание**

*Маленькое подмножество набора данных не отображает атрибуты целого.*

```
# вычисление различия пересечения последних 2 строк и первые 2
столбца
```

```
np.var (dataset[-2:, :2])
```

Вывод предыдущего кода следующий:

```
In [15]: np.var(dataset[-2:, :2])  
Out[15]: 4.674691991769191
```

**Рисунок 1.10: Различие маленького подмножества набора данных**

12. Значения различия могли бы казаться немного странными сначала.

Можно всегда возвращаться к *Мерам Дисперсионной* темы для оценки того, что было получено.

Просто следует помнить, что различие не является стандартным отклонением:

```
# вычисление стандартного отклонения для набора данных  
np.std(dataset)
```

Вывод предыдущего кода следующие:

```
In [16]: np.std(dataset)  
Out[16]: 4.838197554269257
```

**Рисунок 1.11: Стандартное отклонение полного набора данных**

## Действие 2: индексация, разрезание, разделение, и Итерация

### Решение:

Требуется использовать функции NumPy, чтобы индексировать, нарезать, разделить, и выполнить итерации ndarrays.

### Индексация

1. Импортировать необходимые библиотеки:

```
# импорт необходимых зависимостей
import numpy as np
```

2. Загрузить **normal\_distribution.csv** использование набора данных NumPy.

Удостовериться, что все работает, взглянув на ndarray, как в предыдущем действии:

```
# загрузка набора данных
dataset = np.genfromtxt ('./data/normal_distribution.csv', delimiter = ',')
In [17]: dataset = np.genfromtxt ('normal_distribution.csv', delimiter = ',')
```

**Рисунок 1.12: Загрузка набора данных**

3. Во-первых, необходимо использовать простую индексацию для 2 строки.

Для более ясного понимания все элементы сохраняются в переменной:

```
# индексация второй строки набора данных (вторая строка)
second_row = dataset [1]
np.mean (second_row)
```

Вывод предыдущего кода следующий:

```
In [18]: second_row = dataset [1]
          np.mean (second_row)
Out[18]: 97.27899259
```

**Рисунок 1.13: Среднее второй строки**

4. Теперь нужно инвертировать, индексировать последнюю строку и вычислить

среднее из той строки. Всегда следует помнить, что, отрицательное число индексирует список от конца:

```
# индексация последнего элемента набора данных (на строку),  
last_row = dataset [-1]  
np.mean (last_row)
```

Вывод предыдущего кода следующий:

```
In [19]: last_row = dataset [-1]  
np.mean (last_row)  
  
Out[19]: 99.34233100624999
```

**Рисунок 1.14: Вычисление среднего из последней строки**

5. К двумерным данным можно получить доступ то же как со списком Python при помощи **[0] [0]**, где первая из скобок получает доступ к строке и вторая доступ к столбцу.

Однако можно также использовать разделенную запятой нотацию, как **[0, 0]**:

```
# индексация первого значения второй строки (1-я строка, 1-е значение)  
first_val_first_row = dataset [0] [0]  
np.mean (first_val_first_row)
```

Вывод предыдущего кода следующий:

```
In [20]: first_val_first_row = dataset [0] [0]  
np.mean (first_val_first_row)  
  
Out[20]: 99.14931546
```

**Рисунок 1.15: Вычисление среднего от единственного значения не выдает ошибку**

6. К последнему значению предпоследней строки можно легко получить доступ с обратной индексацией. Необходимо помнить, что **-1** означает последний элемент:

```
# индексация последнего значения предпоследней строки (используется
# объединенный синтаксис для доступа),
last_val_second_last_row = dataset [-2,-1]
np.mean (last_val_second_last_row)
```

Вывод предыдущего кода следующий:

```
In [21]: last_val_second_last_row = dataset [-2,-1]
          np.mean (last_val_second_last_row)

Out[21]: 103.83852459
```

**Рисунок 1.16: Использование разделенной запятой нотации**

### Разрезание

7. Для создания 2x2 матрицы, которая начинается во второй строке и втором столбце следует использовать [1:3, 1:3]:

```
# разрезание пересечения 4 элементов (2x2) первых двух строк и
# первых двух столбцов
subsection_2x2 = dataset [1:3, 1:3]
np.mean (subsection_2x2)
```

Вывод предыдущего кода следующий:

```
In [22]: subsection_2x2 = dataset [1:3, 1:3]
          np.mean (subsection_2x2)

Out[22]: 95.63393608250001
```

**Рисунок 1.17: Среднее из 2x2 подмножества**

8. Третье значение позволяет выбирать определенные значения как любой другой элемент путем обеспечения значения 2. Это означает, что пропускаются значения между и берется только каждый второй элемент из используемого списка. В этой задаче требуется иметь любой элемент, таким образом, нужно обеспечить индексацию:: 2, которая, как уже обсуждено, возьмет любой элемент целого списка:

```
# выбор каждого второго элемента пятой строки  
every_other_elem = набор dataset [6:: 2]  
np.mean (every_other_elem)
```

Вывод предыдущего кода следующий:

```
In [88]: every_other_elem = dataset [6:: 2]  
np.mean (every_other_elem)  
  
Out[88]: 1960      NaN  
1961    143.833038  
1962    147.509427  
1963    151.935963  
1964    156.853728  
1965    161.980054  
1966    167.315544  
1967    172.833493  
1968    178.121462  
1969    182.792661  
1970    186.460030  
1971    189.025209  
1972    190.578209  
1973    191.392632  
1974    191.933234  
1975    192.466299  
1976    193.006849  
1977    193.581695  
1978    194.531075  
1979    196.289190  
1980    199.463299  
1981    203.428933  
1982    208.500133  
1983    214.424742  
1984    220.888183  
... 227 231 1 1
```

**Рисунок 1.18: Выбор каждого другого седьмого элемента в строке**

9. Отрицательные числа могут также использоваться для инвертирования элементов:

```
# инвертирование порядка записи, первые две строки в обратном  
порядке  
reversed_last_row = dataset [-1::-1]  
np.mean (reversed_last_row)
```

Вывод предыдущего кода следующий:

```
In [24]: reversed_last_row = dataset [-1::-1]
np.mean (reversed_last_row)
```

```
Out[24]: 100.16536917390624
```

**Рисунок 1.19:** Часть последней строки с элементами в обратном порядке

### Разделение

- Горизонтальное разделение данных может быть сделано **hsplit** методом. Следует обратить внимание, что, если набор данных не может быть разделен с данным количеством частей, то выдастся ошибка:

*# разделение набора данных горизонтально на индексы одна третья и две третиных*

```
hor_splits = np.hsplit (dataset, (3))
```

```
In [34]: hor_splits = np.hsplit(dataset, [3])
```

**Рисунок 1.20:** Разделение набора данных горизонтально на индексы одна третья и две третиных

- Теперь нужно разделить первую треть на две равных части вертикально. Существует также **vsplit** метод, который делает именно это.

*# разделение набора данных вертикально на индексе 2*

```
ver_splits = np.vsplit (hor_splits [0], (2))
```

```
In [33]: ver_splits = np.vsplit (hor_splits [0], (2))
```

**Рисунок 1.21:** Разделение набора данных вертикально на индексы одна третья и две третиных

- При сравнении форм видно, что подмножество имеет необходимую половину строк и третью половину столбцов:

*# требуемый подраздел набора данных, который имеет только половину строк и только одной трети столбцов*

```
print ("Dataset", dataset.shape)
```

```
print ("Subset", ver_splits [0] .shape)
```

Вывод предыдущего кода следующий:

```
In [32]: print ('Dataset', dataset.shape)
          print ('Subset', ver_splits [0] .shape)

Dataset (24, 8)
Subset (12, 3)
```

**Рисунок 1.22: Сравнение форм оригинала набора данных и его подмножества**

## Итерация

4. Смотря на данную часть кода, видно, что индекс просто увеличен с каждым элементом.

Это работает только с одномерными данными. Если потребуется индексировать многомерные данные, это не будет работать:

```
# итерация по целому набору данных (каждое значение в каждой строке) curr_index = 0
for x in np.nditer (dataset):
    print (x, curr_index)
    curr_index += 1
```

Вывод предыдущего кода следующий:

```
In [36]: curr_index = 0

for x in np.nditer (dataset):
    print (x, curr_index)
    curr_index += 1
```

99.14931546 0  
104.03852715 1  
107.43534677 2  
97.85230675 3  
98.74986914 4  
98.80833412 5  
96.81964892 6  
98.56783189 7  
92.02628776 8  
97.10439252 9  
99.32066924 10  
97.24584816 11  
92.9267508 12  
92.65657752 13  
105.7197853 14  
101.23162942 15  
95.66253664 16  
95.17750125 17

Рисунок 1.23: Итерация всего набора данных

5. `ndenumerate` является правильным методом для использования в этой задаче. В дополнение к значению это возвращает индекс. Это работает с многомерными данными, также:

*# итерация по целому набору данных с индексами, соответствующими положению в наборе данных*

```
for index, value in np.ndenumerate (dataset):
    print (index, value)
```

Вывод предыдущего кода следующий:

```
In [37]: for index, value in np.ndenumerate (dataset):
    print (index, value)

(0, 0) 99.14931546
(0, 1) 104.03852715
(0, 2) 107.43534677
(0, 3) 97.85230675
(0, 4) 98.74986914
(0, 5) 98.80833412
(0, 6) 96.81964892
(0, 7) 98.56783189
(1, 0) 92.02628776
(1, 1) 97.10439252
(1, 2) 99.32066924
(1, 3) 97.24584816
(1, 4) 92.9267508
(1, 5) 92.65657752
(1, 6) 105.7197853
(1, 7) 101.23162942
(2, 0) 95.66253664
(2, 1) 95.17750125
(2, 2) 90.93318132
```

**Рисунок 1.24:** Перечисление набора данных с многомерными данными

### Действие 3: фильтрация, сортировка, объединение, и

#### Изменение

#### Решение:

Требуется использовать функции фильтрации NumPy для сортировки, укладки, объединения и изменения данных.

1. Импортировать необходимые библиотеки:

```
# импорт необходимых зависимостей
import numpy as np
```

2. Загрузить **normal\_distribution.csv** набор данных NumPy. Удостовериться, что все работает, взглянув на ndarray, как в предыдущем действии:

```
# загрузка набора данных
dataset = np.genfromtxt ('./data/normal_distribution.csv', delimiter = ',')
```

#### Фильтрация

3. Получение значений, больше, чем 105, может быть выполнено путем предоставления условия в скобках:

```
# значения, которые больше 105
vals_greater_five = dataset [dataset > 105]
```

```
In [38]: vals_greater_five = dataset [dataset > 105]
```

**Рисунок 1.25: Значения, которые больше 105**

4. Для использования более сложных условий можно использовать метод **извлечения (extract)** NumPy.

```
# значения, которые находятся между 90 и 95
vals_between_90_95 = np.extract ((dataset> 90) & (dataset <95), dataset)
```

```
In [41]: vals_between_90_95 = np.extract ((dataset> 90) & (dataset <95), dataset)
```

**Рисунок 1.26: Значения, которые находятся между 90 и 95**

5. **Where** метод **numpy** позволяет только получать **индексы (строки, столбцы)**

для каждого из совпадающих значений. В этой задаче нужно распечатать их. Можно объединить **строки** с соответствующими **столбцами с помощью List comprehension**. В этом примере просто добавлен столбец к соответствующей строке:

```
# индексы значений, которые имеют дельту меньше чем 1 - 100  
rows, cols = np.where (abs (dataset - 100) <1)  
one_away_indices = [[rows [index], cols [index]] for (index, _) in  
np.ndenumerate (rows)]
```

```
In [42]: rows, cols = np.where (abs (dataset - 100) <1)  
one_away_indices = [[rows [index], cols [index]] for (index, _) in np.ndenumerate (rows)]
```

**Рисунок 1.27: Индексы значений, которые имеют дельту меньше чем 1-100**

### Примечание

*List comprehension* является способом Python для отображения по данным. Это удобная нотация для создания нового списка с некоторой операцией, которая относится к каждому элементу старого списка.

Например, если нужно удвоить значение каждого элемента в этом списке, *list = [1, 2, 3, 4, 5]*, следует использовать *list comprehension* как *this:doubled\_list = [x\*x for x in list]*. Это дало бы следующий список: *[1, 4, 9, 16, 25]*.

### Сортировка

6. Сортировка каждой строки в наборе данных работает при помощи **метода сортировки**.

```
# значения отсортированы для каждой строки  
row_sorted = np.sort (dataset)
```

```
In [43]: row_sorted = np.sort (dataset)
```

**Рисунок 1.28: Значения отсортированы для каждой строки**

7. С многомерными данными можно использовать параметр **оси** для определения, какой набор данных должен быть отсортирован. Если **axis = 0**, то в этом случае будет сортировка по столбцам:

```
# значения отсортированы для каждого столбца
```

```
col_sorted = np.sort (dataset, axis=0)
```

```
In [44]: col_sorted = np.sort (dataset, axis=0)
```

**Рисунок 1.29:** Значения отсортированы для каждого столбца

8. Если необходимо сохранить порядок набора данных и только знать, какие индексы значений в отсортированном наборе данных имели бы, можно использовать **argsort**. В сочетании с индексацией можно легко получить доступ к отсортированным элементам:

```
# индексы положений для каждой строки
```

```
index_sorted = np.argsort (dataset)
```

```
In [45]: index_sorted = np.argsort (dataset)
```

**Рисунок 1.30:** Индексы положений для каждой строки

## Объединение

9. Следует использовать объединяющие функции, чтобы добавить вторую половину первого столбца, добавить второй столбец к объединенному набору данных и добавить третий столбец к объединенному набору данных.

```
# разделение набора данных от activity03
```

```
thirds = np.hsplit (dataset, (3))
```

```
halfed_first = np.vsplit (thirds [0], (2))
```

```
In [47]: thirds = np.hsplit (dataset, [3])
```

```
halfed_first = np.vsplit (thirds [0], (2))
```

**Рисунок 1.31:** Разделение набора данных от activity03

10. В зависимости от способа, которым требуется объединить данные, следует использовать **vstack** или **hstack**. Оба берут список наборов данных для укладки вместе:

```
# добавление второй половины первого столбца к данным
```

```
first_col = np.vstack ([halfed_first [0], halfed_first[1]])
```

```
In [48]: first_col = np.vstack ([halfed_first [0], halfed_first[1]])
```

**Рисунок 1.32: Добавление второй половины первого столбца к данным**

11. После vstacking вторая половина набора данных разделения есть одна треть начального набора данных, снова сложенного вместе. Требуется добавить другие два оставшихся набора данных к **first\_col** набору данных. Это реализуемо при помощи **hstack** метода, который комбинирует уже объединенный **first\_col** со вторым из трех наборов данных разделения:

```
# добавление второго столбца к объединенному набору данных
```

```
first_second_col = np.hstack ([first_col, thirds [1]])
```

```
In [49]: first_second_col = np.hstack ([first_col, thirds [1]])
```

**Рисунок 1.33: Добавление второго столбца к набору данных**

12. Для повторной сборки начального набора данных одна треть все еще отсутствует. Можно воспользоваться **hstack** для добавления последнего столбца одной трети в набор данных.

```
# добавление третьего столбца к объединенному набору данных
```

```
full_data = np.hstack ([first_second_col, thirds [2]])
```

```
In [52]: full_data = np.hstack ([first_second_col, thirds [2]])
```

**Рисунок 1.34: Добавление третьего столбца к объединенному набору  
данных**

## Изменение

13. Первая подзадача должна изменить набор данных в единственный список. Это сделано с помощью `reshape` метода:

```
# изменение к списку значений  
single_list = np.reshape (dataset, (1,-1))
```

```
In [53]: single_list = np.reshape (dataset, (1,-1))
```

**Рисунок 1.35: Изменение к списку значений**

14. Используя `-1` можно не задумываться о размерах данных.

```
# изменение к матрице с двумя столбцами  
two_col_dataset = dataset.reshape (-1, 2)
```

```
In [54]: two_col_dataset = dataset.reshape (-1, 2)
```

**Рисунок 1.36: Изменение к матрице с двумя столбцами**

#### Действие 4: Использование pandas для вычисления среднего, медианы, и различие для данных чисел

Решение:

Использовать функции панд такие, как: mean, median, и variance для выполнения некоторых вычислений на данных:

1. Импортировать необходимые библиотеки:

```
# import необходимых зависимостей  
import pandas as pd
```

```
In [55]: import pandas as pd
```

Рисунок 1.37: Импорт pandas

2. После импорта панд использовать **read\_csv** метод для загрузки вышеупомянутого набора данных. Нужно использовать первый столбец, содержащий названия страны, как индекс. Используется **index\_col** параметр для этого.

```
# загрузка набора данных  
dataset = pd.read_csv ('./data/world_population.csv', index_col=0)  
In [56]: dataset = pd.read_csv ('world_population.csv', index_col=0)
```

Рисунок 1.38: Загрузка набора данных

3. Во-первых, требуется распечатать подмножество набора данных с первыми двумя строками. Можно снова использовать синтаксис списка Python для создания подмножества DataFrame первых двух строк:

```
# рассмотрение первых двух строк набора данных  
dataset [0:2]
```

Вывод предыдущего кода следующий:

In [57]: dataset[0:2]													
Out[57]:													
Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008
Aruba	ABW	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	307.972222	312.366667	314.983333	316.827778	318.666667	320.622222	...	562.322222	563.011111
Andorra	AND	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	30.587234	32.714894	34.914894	37.170213	39.470213	41.800000	...	180.591489	182.161702

2 rows × 60 columns

**Рисунок 1.39: Первые две распечатанные строки**

4. Необходимо убедиться, что набор данных был успешно загружен, после чего можно начать решать задачу.

К третьей строке можно получить доступ путем индексации `dataset.iloc[[2]]`.

Для получения среднего из стран, а не ежегодного столбца нужно передать параметр **оси**:

```
# вычисление среднего из третьей строки
dataset.iloc [[2]] .mean (axis=1)
```

Вывод предыдущего кода следующий:

In [58]:	dataset.iloc [[2]] .mean (axis=1)
Out[58]:	Country Name Afghanistan 25.373379 dtype: float64

**Рисунок 1.40: Вычисление среднего из третьей строки**

5. Последний элемент DataFrame, точно так же, как с NumPy ndarrays и списками Python, может быть индексирован с помощью **-1** как индекс. Так, `dataset.iloc [[-1]]` даст последнюю строку:

```
# вычисление среднего из последней строки
dataset.iloc [[-1]] .mean (axis=1)
```

Вывод предыдущего кода следующий:

```
In [59]: dataset.iloc [[-1]] .mean (axis=1)
```

```
Out[59]: Country Name
Zimbabwe    24.520532
dtype: float64
```

**Рисунок 1.41: Вычисление среднего из последней строки**

6. Помимо использования `iloc` к строкам для доступа на основе их индекса, можно также использовать `loc`, которое работает на основе индекса столбца. Это может быть определено при помощи `index_col=0` в вызове `read_csv`:

```
# вычисление среднего из страны Германия
dataset.loc [[“Germany”]] .mean (axis=1)
```

Вывод предыдущего кода следующий:

```
In [61]: dataset.loc [ [ ‘Germany’ ] ] .mean (axis=1)
```

```
Out[61]: Country Name
Germany    227.773688
dtype: float64
```

**Рисунок 1.42: Индексация страны и вычисление среднего для Германии**

7. Перемещаясь в следующий набор задач, которые покрывают использование медианы, будет видно, что API последователен между различными методами. Это означает, что следует продолжать предоставлять `axis=1` вызовам метода, чтобы удостовериться, что методы применяются для каждой страны:

```
# вычисление медианы последней строки
dataset.iloc [[-1]] .median (axis=1)
```

Вывод предыдущего кода следующий:

```
In [62]: dataset.iloc [[-1]] .median (axis=1)
```

```
Out[62]: Country Name
Zimbabwe    25.505431
dtype: float64
```

**Рисунок 1.43: Поиск медианы в последней строке**

8. Разрезание строк у панд подобно разрезанию в NumPy. Можно использовать реверсивную индексацию для получения последних трех столбцов **dataset [-3:]**:

```
# вычисление медианы последних 3 строк  
dataset[-3:].median(axis=1)
```

Вывод предыдущего кода следующий:

```
In [63]: dataset[-3:].median(axis=1)  
  
Out[63]: Country Name  
Congo, Dem. Rep.      14.419050  
Zambia                10.352668  
Zimbabwe              25.505431  
dtype: float64
```

**Рисунок 1.44: Медиана последних трех столбцов**

9. При обработке больших наборов данных порядок, в котором выполняются методы, имеет значение. **head (10)** просто берет набор данных и возвращает первые 10 строк в нем, сокращая ввод до метода **mean ()** кардинально. Это будет определенно оказывать влияние при использовании более интенсивно использующих память вычислений, поэтому необходимо следить за порядком:

```
# вычисление медианы первых 10 стран  
dataset.head(10).median(axis=1)
```

Вывод предыдущего кода следующий:

```
In [64]: dataset.head(10).median(axis=1)
```

```
Out[64]: Country Name
Aruba           348.022222
Andorra         107.300000
Afghanistan    19.998926
Angola          8.458253
Albania         106.001058
Arab World      15.307283
United Arab Emirates 19.305072
Argentina       11.618238
Armenia          105.898033
American Samoa 220.245000
dtype: float64
```

**Рисунок 1.45:** Использование оси для вычисления медианы первых 10 строк

10. Последний метод, который будет освещен здесь - различие. Снова, панды предоставляют последовательный API, который делает его использование легким. Так как необходимо отобразить последние пять столбцов, следует использовать метод **tail**:

```
# вычисление различия последних 5 столбцов
dataset.var().tail()
```

Вывод предыдущего кода следующий:

```
In [65]: dataset.var().tail()
```

```
Out[65]: 2012    3.063475e+06
          2013    3.094597e+06
          2014    3.157111e+06
          2015    3.220634e+06
          2016        NaN
dtype: float64
```

**Рисунок 1.46:** Различие между последними пятью столбцами

11. Как упомянуто во введении в pandas, pandas совместимы с несколькими

функциями NumPy.

Вот пример того, как использовать **mean** метод NumPy с pandas DataFrame. В некоторых случаях NumPy имеет лучшую функциональность, но у панд, с ее DataFrames формат лучше:

```
# совместимость NumPy и pandas
import numpy as np
print ("pandas", dataset ["2015"] .mean ())
print ("numpy", np.mean (dataset ["2015"]))
```

Вывод предыдущего кода следующий:

```
In [66]: import numpy as np
          print ('pandas', dataset ['2015'] .mean ())
          print ('numpy', np.mean (dataset ['2015']))
```

```
pandas 368.7066010400187
numpy 368.7066010400187
```

**Рисунок 1.47: Использование метода поиска среднего NumPy с pandas DataFrame**

## Действие 5: Индексация, Разрезание и Итерация Используя панд

### Решение:

Использовать индексацию, разрезание и итерацию операций для отображения плотности населения Германии, Сингапура, Соединенных Штатов и Индии в течение 1970, 1990, и 2010 годов.

### Индексация

1. Импортировать необходимые библиотеки:

```
# импорт необходимых зависимостей
```

```
import pandas as pd
```

2. После импорта панд использовать **read\_csv** метод для загрузки упомянутого набора данных. Следует использовать первый столбец, содержащий названия страны, как индекс. Использовать **index\_col** параметр для этого.

```
# загрузка набора данных
```

```
dataset = pd.read_csv ('./data/world_population.csv', index_col=0)
```

3. Для индексации строки с **index\_col** “Соединенными Штатами” использовать метод **loc**:

```
# индексация строки США
```

```
dataset.loc [[“United States”]] .head ()
```

Вывод предыдущего кода следующий:

In [67]:	dataset.loc [[‘United States’]] .head ()																																																
Out[67]:	<table><thead><tr><th></th><th>Country Code</th><th>Indicator Name</th><th>Indicator Code</th><th>1960</th><th>1961</th><th>1962</th><th>1963</th><th>1964</th><th>1965</th><th>1966</th><th>...</th><th>2007</th><th>2008</th><th>2009</th><th>:</th></tr><tr><th>Country Name</th><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></thead><tbody><tr><td>United States</td><td>USA</td><td>Population density (people per sq. km of land ...)</td><td>EN.POP.DNST</td><td>NaN</td><td>20.05588</td><td>20.366723</td><td>20.661953</td><td>20.950959</td><td>21.214527</td><td>21.460952</td><td>...</td><td>32.878611</td><td>33.243687</td><td>33.536399</td><td>33.81</td></tr></tbody></table> <p>1 rows × 60 columns</p>		Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009	:	Country Name																United States	USA	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	20.05588	20.366723	20.661953	20.950959	21.214527	21.460952	...	32.878611	33.243687	33.536399	33.81
	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009	:																																		
Country Name																																																	
United States	USA	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	20.05588	20.366723	20.661953	20.950959	21.214527	21.460952	...	32.878611	33.243687	33.536399	33.81																																		

Рисунок 1.48: Индексация Соединенных Штатов с методом loc

4. Обратная индексация может также использоваться с pandas. Для индексации предпоследней строки необходимо использовать **iloc** метод.

```
# индексация последней предпоследней строки индексом
```

```
dataset.iloc [-2]]
```

Вывод предыдущего кода следующий:

```
In [68]: dataset.iloc [-2]
Out[68]:
   Country      Indicator      Indicator    1960    1961    1962    1963    1964    1965    1966 ... 2007    2008    2009    2010
   Code        Name       Code
Country
Name
   Zambia      Population
   ZMB      density
   (people
   per sq. km
   of land ...
   EN.POP.DNST  NaN  4.227724  4.359305  4.496824  4.639914  4.788452  4.942343 ... 17.135926 17.641587 18.170609 18.721585
1 rows x 60 columns
```

**Рисунок 1.49: Индексация предпоследней строки**

- Столбцы индексируются с помощью их заголовка. Это первая строка файла CSV. Для получения столбца с заголовком **2000** нужно использовать нормальную индексацию. **head ()**, метод просто возвращает первые пять строк:

```
# индексация столбца 2000 как ряд
dataset["2000"].head()
```

Вывод предыдущего кода следующий:

```
In [69]: dataset['2000'].head()
Out[69]: Country Name
Aruba      504.766667
Andorra    139.146809
Afghanistan 30.177894
Angola     12.078798
Albania    112.738212
Name: 2000, dtype: float64
```

**Рисунок 1.50: Индексация всех 2,000 столбцов**

- Так как двойная нотация скобок снова возвращает DataFrame, можно объединить вызовы метода в цепочку для получения отличных элементов. Для получения плотности населения Индии в 2000 сначала необходимо получить данные на 2000 год и только затем выбрать Индию с помощью **loc ()** метода:

```
# индексация плотности населения Индии в 2000 (DataFrame)
dataset[["2000"]].loc[["India"]]
```

Вывод предыдущего кода следующий:

```
In [70]: dataset[['2000']].loc[['India']]
```

Out[70]:

Country Name
India 354.326858

**Рисунок 1.51: Плотность населения индии в 2000**

- Если требуется получить объект ряда, следует заменить двойные скобки единственными. Это даст отличное значение вместо нового DataFrame:

```
# индексация плотности населения Индии в 2000 (Ряд)
dataset["2000"].loc["India"]
```

Вывод предыдущего кода следующий:

```
In [71]: dataset['2000'].loc['India']
```

Out[71]: 354.326858357522

**Рисунок 1.52: плотность населения Индии в 2000 году**

## Разрезание

- Для создания части со строками 2 - 5 следует использовать `iloc()` метод снова. Можно просто обеспечить тот же синтаксис разрезания как NumPy:

```
# разрезание стран строк 2 - 5
dataset.iloc[1:5]
```

Вывод предыдущего кода следующий:

In [72]: dataset.iloc [1:5]															
Out[72]:		Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009
Country Name															
Andorra	AND	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	30.587234	32.714894	34.914894	37.170213	39.470213	41.800000	...	180.591489	182.161702	181.859574	
Afghanistan	AFG	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	14.038148	14.312061	14.599692	14.901579	15.218206	15.545203	...	39.637202	40.634655	41.674005	
Angola	AGO	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	4.305195	4.384299	4.464433	4.544558	4.624228	4.703271	...	15.387749	15.915819	16.459536	
Albania	ALB	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	60.576642	62.456898	64.329234	66.209307	68.058066	69.874927	...	108.394781	107.566204	106.843759	

4 rows × 60 columns

**Рисунок 1.53: Страны в строках 2 - 5**

9. Используя `loc()` метод, можно также получить доступ к выделенным строкам их `index_col` (который был определен в вызове `read_csv`). Для получения нескольких строк в новом DataFrame использовать вложенные скобки для получения списка элементов:

```
# нарезка строк Германия, Сингапур, Соединенные Штаты и Индия
dataset.loc [[“Germany”, “Singapore”, “United States”, “India”]]
```

Вывод предыдущего кода следующий:

In [73]: dataset.loc[[‘Germany’, ‘Singapore’, ‘United States’, ‘India’]]															
Out[73]:		Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009
Country Name															
Germany	DEU	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	210.172807	212.029284	214.001527	215.731495	217.579970	219.403406	...	235.943362	235.5221		
Singapore	SGP	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	2540.895522	2612.238806	2679.104478	2748.656716	2816.268657	2887.164179	...	6602.300719	6913.4228		
United States	USA	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	20.055880	20.366723	20.661953	20.950959	21.214527	21.460952	...	32.878611	33.2436		
India	IND	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	154.275864	157.424902	160.679256	164.029246	167.470047	170.995768	...	396.774384	402.6214		

4 rows × 60 columns

**Рисунок 1.54: Нарезка стран Германия, Сингапур, Соединенные Штаты и Индия**

10. Так как двойные запросы через скобки возвращают новый DataFrames, можно объединить методы и доступ к отдельным подкадрам данных:

```
# нарезка подмножества Германии, Сингапура, Соединенных Штатов  
и Индии  
  
# в течение 1970, 1990, 2010 <  
  
country_list = ["Germany", "Singapore", "United States", "India"]  
dataset.loc [country_list] [[“1970”, “1990”, “2010”]]
```

Вывод предыдущего кода следующий:

```
In [74]: country_list = ['Germany', 'Singapore', 'United States', 'India']  
dataset.loc [country_list] [ ['1970', '1990', '2010']]  
  
Out[74]:
```

	1970	1990	2010
Country Name			
Germany	223.897371	227.517054	234.606908
Singapore	3096.268657	4547.958209	7231.811966
United States	22.388131	27.254514	33.817936
India	186.312757	292.817404	414.028200

**Рисунок 1.55: Нарезка Германии, Сингапура, Соединенных Штатов и Индии с их плотностью населения за 1970, 1990, и 2010**

## Итерация

11. Чтобы выполнить итерации набора данных и распечатать страны вплоть до **Анголы**, рекомендуется использовать **iterrows ()** метод. Индекс будет названием строки, и строка будет содержать все столбцы:

```
# итерация по первым трем странам (строка строкой)  
for index, row in dataset.iterrows ():  
    # только печать строк до Анголы  
    if index == 'Angola':  
        break  
    print (index, '\n', row [ [“Country Code”, “1970”, “1990”, “2010”]], '\n')
```

Вывод предыдущего кода следующий:

```
In [75]: for index, row in dataset.iterrows ():
    if index == 'Angola':
        break
    print (index, '\n', row[['Country Code', '1970', '1990', '2010']], '\n')

Aruba
  Country Code      ABW
1970            328.139
1990            345.267
2010            564.428
Name: Aruba, dtype: object

Andorra
  Country Code      AND
1970            51.6574
1990            115.981
2010            179.615
Name: Andorra, dtype: object

Afghanistan
  Country Code      AFG
1970            17.0344
1990            18.4842
2010            42.8303
Name: Afghanistan, dtype: object
```

**Рисунок 1.56: Итерация всех стран до Анголы**

## Действие 6: фильтрация, сортировка и изменение

### Решение:

Использовать pandas, чтобы отфильтровать, отсортировать, и изменить данные.

### Фильтрация

1. Импортировать необходимые библиотеки:

```
# импорт необходимых зависимостей  
import pandas as pd
```

2. После импорта панд использовать **read\_csv** метод для загрузки вышеупомянутого набора данных. Следует использовать первый столбец, содержащий названия страны, в качестве индекса. Использовать **index\_col** параметр для этого.

```
# загрузка набора данных  
dataset = pd.read_csv ('./data/world_population.csv', index_col=0)
```

3. Вместо того, чтобы использовать синтаксис скобки, можно также использовать метод **фильтра** для фильтрации определенных объектов. Здесь, предоставляется список элементов, которые должны быть сохранены:

```
# фильтрация столбцов 1961, 2000 и 2015  
dataset.filter (items = ["1961", "2000", "2015"]) .head ()
```

Вывод предыдущего кода следующий:

In [77]:	<code>dataset [(dataset ['2000'] &gt; 500)] [[ '2000']]</code>
Out[77]:	
<b>2000</b>	
Country Name	
Aruba	504.766667
Bangladesh	1008.532988
Bahrain	939.232394
Bermuda	1236.660000
Barbados	627.530233
Channel Islands	766.623711
Gibraltar	2735.100000
Hong Kong SAR, China	6347.619048
Macao SAR, China	21595.350000
St. Martin (French part)	521.764706
Monaco	16040.500000
Maldives	953.333333
Malta	1191.759375
Mauritius	584.666502
Nauru	502.100000
Singapore	6011.771642
Sint Maarten (Dutch part)	897.617647

**Рисунок 1.57: Фильтрация для 1961, 2000, и 2015 годов**

4. Если требуется отфильтровать для определенных значений в определенном столбце, можно использовать условия. Для получения всех стран, которые имели более высокую плотность населения, чем 500 в 2000 необходимо просто передать это условие в скобках:

*# фильтрация стран, которые имели большую плотность населения, чем 500 в 2000*

`dataset [(dataset ["2000"] > 500)] [[ "2000"]]`

Вывод предыдущего кода следующий:

In [77]:	dataset [(dataset ['2000'] > 500)] [['2000']]
Out[77]:	2000
<b>Country Name</b>	
<hr/>	
Aruba	504.766667
Bangladesh	1008.532988
Bahrain	939.232394
Bermuda	1236.660000
Barbados	627.530233
Channel Islands	766.623711
Gibraltar	2735.100000
Hong Kong SAR, China	6347.619048
Macao SAR, China	21595.350000
St. Martin (French part)	521.764706
Monaco	16040.500000
Maldives	953.333333
Malta	1191.759375
Mauritius	584.666502
Nauru	502.100000
Singapore	6011.771642
Sint Maarten (Dutch part)	897.617647

**Рисунок 1.58: Фильтрация стран, которые имели плотность, больше 500 в 2000**

5. Один мощный параметр фильтрации - метод **regex**. Это позволяет искать произвольные столбцы или строки (в зависимости от данного индекса), которые соответствуют определенному **regex**. Для получения всех столбцов, которые начинаются с 2, можно просто передать **^2**, что означает, что он начинается с 2:

```
# фильтрация в течение 2000 и позже
dataset.filter(regex = " ^2 ", axis=1) .head ()
```

Вывод предыдущего кода следующий:

	In [78]:	dataset.filter (regex ='^2', axis=1).head ()												
Out[78]:		2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011												
Country Name		Aruba 504.766667 516.077778 527.750000 538.972222 548.566667 555.727778 560.166667 562.322222 563.011111 563.422222 564.427778 566.311111 Andorra 139.146809 144.191489 151.161702 159.112766 166.674468 172.814894 177.389362 180.591489 182.161702 181.859574 179.614894 175.161702 Afghanistan 30.177894 31.448029 32.912231 34.475030 35.995236 37.373936 38.574296 39.637202 40.634655 41.674005 42.830327 44.127634 Angola 12.078798 12.483188 12.921871 13.388462 13.873025 14.368286 14.872437 15.387749 15.915819 16.459536 17.020898 17.600302 Albania 112.738212 111.685146 111.350730 110.934891 110.472226 109.908285 109.217044 108.394781 107.566204 106.843759 106.314635 106.013869												
		Aruba 504.766667 516.077778 527.750000 538.972222 548.566667 555.727778 560.166667 562.322222 563.011111 563.422222 564.427778 566.311111 Andorra 139.146809 144.191489 151.161702 159.112766 166.674468 172.814894 177.389362 180.591489 182.161702 181.859574 179.614894 175.161702 Afghanistan 30.177894 31.448029 32.912231 34.475030 35.995236 37.373936 38.574296 39.637202 40.634655 41.674005 42.830327 44.127634 Angola 12.078798 12.483188 12.921871 13.388462 13.873025 14.368286 14.872437 15.387749 15.915819 16.459536 17.020898 17.600302 Albania 112.738212 111.685146 111.350730 110.934891 110.472226 109.908285 109.217044 108.394781 107.566204 106.843759 106.314635 106.013869												

**Рисунок 1.59: Получение всех столбцов, начинающихся с 2**

6. Используя параметр **оси**, можно выбрать, в котором направлении должна выполняться фильтрация. Для фильтрации строк вместо столбцов следует передать **axis=0**. Это будет полезно для ситуаций, когда необходимо отфильтровать все строки, которые начинаются с **A**:

# фильтрация стран, которые начинаются с *A*

*dataset.filter (regex = " ^A ", axis=0) .head ()*

Вывод предыдущего кода следующий:

	In [79]:	dataset.filter (regex = '^A', axis=0).head ()												
Out[79]:		Country Code Indicator Name Indicator Code 1960 1961 1962 1963 1964 1965 1966 ... 2007 2008												
Country Name		Aruba ABW Population density (people per sq. km of land ...) EN.POP.DNST NaN 307.972222 312.366667 314.983333 316.827778 318.666667 320.622222 ... 562.322222 563.011111 563 Andorra AND Population density (people per sq. km of land ...) EN.POP.DNST NaN 30.587234 32.714894 34.914894 37.170213 39.470213 41.800000 ... 180.591489 182.161702 181 Afghanistan AFG Population density (people per sq. km of land ...) EN.POP.DNST NaN 14.038148 14.312061 14.599692 14.901579 15.218206 15.545203 ... 39.637202 40.634655 41 Angola AGO Population density (people per sq. km of land ...) EN.POP.DNST NaN 4.305195 4.384299 4.464433 4.544558 4.624228 4.703271 ... 15.387749 15.915819 16 Albania ALB Population density (people per sq. km of land ...) EN.POP.DNST NaN 60.576642 62.456898 64.329234 66.209307 68.058066 69.874927 ... 108.394781 107.566204 106												
		Aruba ABW Population density (people per sq. km of land ...) EN.POP.DNST NaN 307.972222 312.366667 314.983333 316.827778 318.666667 320.622222 ... 562.322222 563.011111 563 Andorra AND Population density (people per sq. km of land ...) EN.POP.DNST NaN 30.587234 32.714894 34.914894 37.170213 39.470213 41.800000 ... 180.591489 182.161702 181 Afghanistan AFG Population density (people per sq. km of land ...) EN.POP.DNST NaN 14.038148 14.312061 14.599692 14.901579 15.218206 15.545203 ... 39.637202 40.634655 41 Angola AGO Population density (people per sq. km of land ...) EN.POP.DNST NaN 4.305195 4.384299 4.464433 4.544558 4.624228 4.703271 ... 15.387749 15.915819 16 Albania ALB Population density (people per sq. km of land ...) EN.POP.DNST NaN 60.576642 62.456898 64.329234 66.209307 68.058066 69.874927 ... 108.394781 107.566204 106												

**Рисунок 1.60: Получение строк, которые начинаются с А**

7. Если требуется получить все строки или столбцы, которые содержат некоторое определенное значение или символ, можно использовать **like** запрос. Например, если требуется иметь только страны, которые содержат слово “land”, такие как Швейцария:

```
# фильтрация стран, которые содержат слово «земля»
```

```
dataset.filter (like = "land", axis=0) .head ()
```

Вывод предыдущего кода следующий:

In [80]: dataset.filter (like = 'land', axis=0) .head ()													
Out[80]:													
	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008
Country Name													
Switzerland	CHE	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	137.479609	141.009285	144.056036	146.458915	148.160089	149.716707	...	191.090115	193.533632
Channel Islands	CHI	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	569.067010	574.551546	580.386598	586.484536	592.742268	599.103093	...	806.783505	812.304124
Cayman Islands	CYM	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	33.441667	33.925000	34.283333	34.579167	34.879167	35.175000	...	214.500000	220.520833
Finland	FIN	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	14.645934	14.745865	14.850484	14.933330	14.983197	15.039460	...	17.391956	17.484038
Faroe Islands	FRO	Population density (people per sq. km of land ...)	EN.POP.DNST	NaN	24.878223	25.181232	25.465616	25.749284	26.047994	26.363897	...	34.813037	34.834527

**Рисунок 1.61: Получение всех стран, содержащих слово “land”**

## Сортировка

8. Сортировка у pandas может быть сделана при помощи **sort\_values** или **sort\_index** методов. Если требуется получить страны с самой низкой плотностью населения в течение определенного года, следует отсортировать по этому определенному столбцу:

```
# значения отсортированы по столбцу 1961
```

```
dataset.sort_values (=[“1961”]) [[“1961”]] .head (10)
```

Вывод предыдущего кода следующий:

```
In [81]: dataset.sort_values ([ '1961' ]) [[ '1961' ]] .head (10)
```

Out[81]:

1961	
Country Name	
Greenland	0.098625
Mongolia	0.632212
Namibia	0.749775
Libya	0.843320
Mauritania	0.856916
Botswana	0.946793
United Arab Emirates	1.207955
Australia	1.364565
Iceland	1.785825
Oman	1.825186

**Рисунок 1.62:** Сортировка по значениям на 1961 год

9. Для сравнения, требуется сделать ту же сортировку **на 2015**. Это даст некоторое лучшее понимание данных. Видно, что порядок стран с самой низкой плотностью населения изменился немного, но первые три записи неизменны:

```
# значения отсортированы по столбцу 2015  
dataset.sort_values (= ["2015"]) [[ "2015" ]] .head (10)
```

Вывод предыдущего кода следующий:

```
In [82]: dataset.sort_values(['2015'])[['2015']].head(10)
```

Out[82]:

2015	
Country Name	
Greenland	0.136713
Mongolia	1.904744
Namibia	2.986590
Australia	3.095579
Iceland	3.299980
Suriname	3.480609
Libya	3.568227
Guyana	3.896800
Canada	3.942567
Mauritania	3.946409

Рисунок 1.63: Сортировка по столбцу 2015

10. Порядок сортировки по умолчанию **возрастает**. Это означает, что необходимо обеспечить отдельный параметр, если нужно сортировать **в порядке убывания**.

```
# значения, отсортированные по столбцу 2015 в порядке убывания  
dataset.sort_values(['2015'], ascending=False)  
[['2015']].head(10)
```

Вывод предыдущего кода следующий:

```
In [86]: dataset.sort_values(['2015'], ascending=False)  
dataset[['2015']].head(10)
```

Out[86]:

2015	
Country Name	
Aruba	577.161111
Andorra	149.942553
Afghanistan	49.821649
Angola	20.070565
Albania	105.444051
Arab World	28.779858
United Arab Emirates	109.533050
Argentina	15.864696
Armenia	105.996207
American Samoa	277.690000

Рисунок 1.64: Сортировка в порядке убывания

## Изменение

11. Как было упомянуто прежде, изменение данных с пандами может быть очень сложной задачей. Требуется получить DataFrames, где столбцы являются **кодами страны**, и единственная строка является 2015 годом. Так как есть только одна маркировка **2015 года**, следует ее копировать столько же раз, сколько ее в наборе данных. Это приведет к каждому значению, получающему индекс строки 2015 года:

```
# преобразование к 2015 году как строки и коду страны как столбцу  
dataset_2015 = dataset[['Country Code', '2015']]  
dataset_2015.pivot(index = ['2015'] * len(dataset_2015), columns =  
    "Country Code", values = "2015"),
```

Вывод предыдущего кода следующий:

In [87]:	dataset_2015 = dataset [['Country Code', '2015']] dataset_2015.pivot (index = ['2015'] * len (dataset_2015), columns = 'Country Code', values = '2015')																														
Out[87]:	<table border="1"> <thead> <tr> <th>Country Code</th><th>ABW</th><th>AFG</th><th>AGO</th><th>ALB</th><th>AND</th><th>ARB</th><th>ARE</th><th>ARG</th><th>ARM</th><th>ASM</th><th>...</th><th>VGB</th><th>VIR</th><th>VNM</th></tr> </thead> <tbody> <tr> <td>2015</td><td>577.161111</td><td>49.821649</td><td>20.070565</td><td>105.444051</td><td>149.942553</td><td>28.779858</td><td>109.53305</td><td>15.864696</td><td>105.996207</td><td>277.69</td><td>...</td><td>200.78</td><td>295.925714</td><td>295.751927</td></tr> </tbody> </table> <p>1 rows x 264 columns</p>	Country Code	ABW	AFG	AGO	ALB	AND	ARB	ARE	ARG	ARM	ASM	...	VGB	VIR	VNM	2015	577.161111	49.821649	20.070565	105.444051	149.942553	28.779858	109.53305	15.864696	105.996207	277.69	...	200.78	295.925714	295.751927
Country Code	ABW	AFG	AGO	ALB	AND	ARB	ARE	ARG	ARM	ASM	...	VGB	VIR	VNM																	
2015	577.161111	49.821649	20.070565	105.444051	149.942553	28.779858	109.53305	15.864696	105.996207	277.69	...	200.78	295.925714	295.751927																	

**Рисунок 1.65: Преобразование набора данных в одну строку для значений 2015 года**

## Задание

Считать набор данных о корабле Титаник, вывести информацию про первых 5 пассажиров, всех, кроме последних 2x и с 3 по 15 включительно, проанализировать на предмет наличия особенностей утонувших, таких как

1. Средний возраст, максимальный и минимальный
2. Узнать количество женщин и мужчин
3. Количество женщин, которым больше 18 лет
4. Количество мужчин, которые были в первом классе.

Отсортировать всех пассажиров по возрасту по возрастанию и по убыванию и вывести

## Заключение

NumPy и pandas являются существенными инструментами для пререкания данных. Их удобные для пользователя интерфейсы и производительная реализация делают обработку данных легкой. Даже при том, что они обеспечивают лишь немногим более лучшее понимание наборов данных, они абсолютно ценные для пререкания, увеличения и очистки наборов данных.

Были изучены основы NumPy и панд и понятий статистики. Даже при том, что изученные статистические понятия являются очень простыми, они необходимы для обогащения визуализации информацией, которая, в большинстве случаев, непосредственно не предоставляется в наших наборах данных. Этот практический опыт поможет реализовать упражнения и операции в лабораторных.

## **Список литературы**

1. Data Visualization with Python Copyright Mario Döbler and Tim Großmann

© 2019 Packt Publishing.