

```
!git clone https://github.com/neuralcomputer/ML_School.git
```

```
➔ Cloning into 'ML_School'...
remote: Enumerating objects: 94, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 94 (delta 5), reused 0 (delta 0), pack-reused 79 (from 1)
Receiving objects: 100% (94/94), 33.83 MiB | 17.91 MiB/s, done.
Resolving deltas: 100% (29/29), done.
```

Работа с табличными данными

Большое количество данных, которые используются в машинном обучении, представлены в виде таблиц, где в столбцах представлены различные признаки и целевые переменные. Для работы с такими данными в Python есть замечательная библиотека [pandas](#) тесно связанная с уже известной нам `numpy`. К пандам отношения не имеет, название пошло от "панельные данные".

✓ Pandas

Создание и индексация

В `pandas` два основных вида объектов (типов) `Series` и `Dataframe`, они похожи между собой, но `Dataframe` это фактически таблица, в которой есть строки, столбцы, а `Series` это один столбец со строками. Столбцы и/или строки могут иметь названия, которые выступают их индексами и к определенному столбцу (или строке) можно обращаться именно по названию.

Создадим `Series` (давайте назовем это последовательностью), в которой будут записаны некоторые данные имеющие названия (индексы). И данные (`data`) и индексы (`index`) могут иметь различный тип: числа, строки, объекты.

Но будьте аккуратны, индексы надо делать так, чтобы было понятно. Ниже увидим примеры разрешенных, но совершенно не понятных индексов.

```
import pandas as pd # подключим библиотеку
import numpy as np
```

```
s = pd.Series(data=[10, "11", ['a',12], 'ppp', 14, 42], # данные
              index=[2.1, '2', 'два', 2, 2.1, -2]) # их индексы
s # здесь 5 ячеек
```

```
➔
```

| | 0 |
|-----|---------|
| 2.1 | 10 |
| 2 | 11 |
| два | [a, 12] |
| 2 | ppp |
| 2.1 | 14 |
| -2 | 42 |

в этой последовательности 5 ячеек (data) и к ним можно обратиться по индексам (index), важно понять, что здесь индексы именно названия, а не номера строк.

```
print(s[2]) # здесь вернется подпоследовательность элементов у которых индекс называется 2 (первая и четвер
print('\n')
print(s['2']) # это ячейка с названием '2', это совершенно другой индекс, отличается от названия 2 (вторая
print('\n')
print(s['два']) # это ячейка с названием 'два', третья ячейка, в которой записан массив
print('\n')
print(s[-2]) # это ячейка с названием -2, пятая ячейка
```

➔ ppp

11

['a', 12]

42

s['hello'] # такого нет

➔

```
-----
KeyError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3790         try:
-> 3791             return self._engine.get_loc(casted_key)
    3792         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine._get_loc_duplicates()

index.pyx in pandas._libs.index.IndexEngine._maybe_get_bool_indexer()

index.pyx in pandas._libs.index._unpack_bool_indexer()

KeyError: 'hello'
```

The above exception was the direct cause of the following exception:

```
KeyError                                Traceback (most recent call last)
-----
3 frames -----
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3796         ):
    3797             raise InvalidIndexError(key)
-> 3798         raise KeyError(key) from err
    3799     except TypeError:
    3800         # If we have a listlike key, _check_indexing_error will raise

KeyError: 'hello'
```



s[2.1] # а такой есть, но это не номер, а название индекса



| | 0 |
|-----|----|
| 2.1 | 10 |
| 2.1 | 14 |



Как и в питру можно делать срезы, в этом случае указываются номера, а не названия элементов.

s



| | 0 |
|-----|---------|
| 2.1 | 10 |
| 2 | 11 |
| два | [a, 12] |
| 2 | ppp |
| 2.1 | 14 |
| -2 | 42 |



s[1:3] # срез, вторая и третья ячейки, здесь это номера, а не названия.



| | 0 |
|-----|---------|
| 2 | 11 |
| два | [a, 12] |



s[1:2]



| | 0 |
|---|----|
| 2 | 11 |



s[1] # это ошибка, нет элемента с названием 1



```
-----
KeyError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3790         try:
-> 3791             return self._engine.get_loc(casted_key)
    3792         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine._get_loc_duplicates()

index.pyx in pandas._libs.index.IndexEngine._maybe_get_bool_indexer()

index.pyx in pandas._libs.index._unpack_bool_indexer()

KeyError: 1
```

The above exception was the direct cause of the following exception:

```
-----
KeyError                                Traceback (most recent call last)
----- 3 frames -----
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3796         ):
    3797             raise InvalidIndexError(key)
-> 3798             raise KeyError(key) from err
    3799     except TypeError:
    3800         # If we have a listlike key, _check_indexing_error will raise

KeyError: 1
```



Но срезы могут быть строковыми (удивительно), тогда это именно названия. Pandas сопоставит символьные индексы числовым, найдет их в последовательности, и вернет все, что между ними, включая границы.

```
v = pd.Series(data=[10, "11", ['a',12], 'ppp', 14], # данные
              index=['a', 'f', 'c', 'd', 'e']) # их индексы
v['a':'d']
# v['a':'g'] # так работать не будет, потому что индекса 'g' нет.
```



| | 0 |
|---|---------|
| a | 10 |
| f | 11 |
| c | [a, 12] |
| d | ppp |



Создайте свои последовательности, обратитесь к элементам. Проверьте, что будет если при создании последовательности не указать index? А если при строковом индексе в последовательности индексы будут повторяться?

Аналогично можно создать таблицу DataFrame, в которой несколько столбцов.

```
df = pd.DataFrame([[10, 'aaa'], [s, 21], [30, 31]])
df
```



| | 0 | 1 |
|---|------------------------------|-----|
| 0 | 10 | aaa |
| 1 | 2.1 10 2 11 два [a, 12] 2... | 21 |
| 2 | 30 | 31 |

здесь два столбца, три строки, названия их сделаны по умолчанию, в одну ячейку мы записали последовательность s. Столбцы (columns) и строки (index) можно назвать.

```
df = pd.DataFrame([[10, 'aaa'], [s, 21], [30, 31]],
                  columns=['невторой', 2],
                  index=[1, '1', 'один'])
df
```



| | невторой | 2 |
|------|------------------------------|-----|
| 1 | 10 | aaa |
| 1 | 2.1 10 2 11 два [a, 12] 2... | 21 |
| один | 30 | 31 |

При индексации по названию сначала указываем название столбца - вернется весь столбец в виде последовательности - потом для нее указываем название строки.

```
s0=df['невторой']
```

```
s1=df['невторой']['один']
s1
```

 30

Названия столбцов содержатся в атрибуте columns и их можно изменять.

В некоторых случаях к столбцу можно обратиться как к атрибуту через знак "." Но это работает не всегда. Попробуйте назвать столбец "столбец 1" или 'index' или "1" и посмотрите, что получится.

```
df.columns=['невторой', 'second']
#df.columns=['index', 'second']
df
```



| | невторой | second |
|------|------------------------------|--------|
| 1 | 10 | aaa |
| 1 | 2.1 10 2 11 два [a, 12] 2... | 21 |
| один | 30 | 31 |

```
df.невторой # работает, название столбца как атрибут
```



| невторой | |
|----------|------------------------------|
| 1 | 10 |
| 1 | 2.1 10 2 11 два [a, 12] 2... |
| один | 30 |



```
df.columns=['столбец 1',2] # изменяем названия столбцов
df['столбец 1']
```



| столбец 1 | |
|-----------|------------------------------|
| 1 | 10 |
| 1 | 2.1 10 2 11 два [a, 12] 2... |
| один | 30 |



```
# а вот с такими названиями работать не будет
#df.столбец 1
df.'столбец 1'
```



```
File "<ipython-input-23-f90f029526d1>", line 3
    df.'столбец 1'
      ^
SyntaxError: invalid syntax
```



```
df.columns=['index','second']
df['index']
```



| index | |
|-------|------------------------------|
| 1 | 10 |
| 1 | 2.1 10 2 11 два [a, 12] 2... |
| один | 30 |



```
# так ошибку не выдает, но выдает что-то не то. Это потому, что index уже определен как атрибут
df.index
```



```
Index([1, '1', 'один'], dtype='object')
```

Атрибут `index` возвращает индекс (названия строк)

Атрибут `columns` возвращает названия столбцов (для таблиц)

Атрибут `values` возвращает значения ячеек как массив `numpy`

```
print(df.index)
print(df.columns)
print(df.values)
print(type(df.values))
```

```

Index([1, '1', 'один'], dtype='object')
Index(['index', 'second'], dtype='object')
[[10 'aaa']
 2.1      10
 2        11
 два     [a, 12]
 2        ppp
 2.1      14
 -2       42
 dtype: object 21]
[30 31]]
<class 'numpy.ndarray'>

```

Срез выполняется по строкам

```
df[1:2]
```

```

index second
1 2.1 10 2 11 два [a, 12] 2 21

```

Если нужно обращаться по номеру (числовому индексу), используем атрибут `iloc`, как если бы это был массив `numpy`.

```
df
```

```

index second
1 10 aaa
1 2.1 10 2 11 два [a, 12] 2... 21
один 30 31

```

```
df.iloc[1,0]
```

```

0
2.1 10
2 11
два [a, 12]
2 ppp
2.1 14
-2 42

```

```
df.iloc[1,-1]
```

```
21
```

Индексация может быть логической, истинные элементы (`True`) отбираются, ложные (`False`) отбрасываются.

```

np.random.seed(123)
s = pd.Series(np.random.normal(size=10))
print(s)
ind=s>0
print(ind)
r=s[ind]
print(r)

```

```

0    -1.085631
1     0.997345
2     0.282978
3    -1.506295
4    -0.578600
5     1.651437
6    -2.426679
7    -0.428913
8     1.265936
9    -0.866740
dtype: float64
0     False
1      True
2      True
3     False
4     False
5      True
6     False
7     False
8      True
9     False
dtype: bool
1     0.997345
2     0.282978
5     1.651437
8     1.265936
dtype: float64

```

Другие способы создания и индексации смотри в документации.

✓ Загрузка из файла

Если б таблички нужно было создавать вручную, это было бы слишком утомительно, к счастью, pandas обладает богатыми возможностями по загрузке файлов таблиц разного формата, *.csv, *.xls и других, как с диска так и из Интернет.

Команда [read_csv\(\)](#) позволяет загрузить файлы csv с заданного файла на диске или адреса в Интернет (первый обязательный аргумент `filepath_or_buffer`). Это текстовые файлы, в которых столбцы таблиц разделяются некоторым символом (запятая, точка с запятой или другие), который можно указать команде (аргумент `sep`), возвращается объект типа `Dataframe`. У команды большие возможности по загрузке данных, можно ограничить количество загружаемых строк (аргумент `nrows`), можно указать загружаемые столбцы (аргумент `usecols`), указать, что делать со строками с пропущенными значениями и другие.

Давайте загрузим табличку о 500 лучших компаниях мира с адреса <https://datahub.io/core/s-and-p-500-companies-financials#resource-constituents-financials>, с разделителями столбцов в виде запятой. В нашем примере по умолчанию названия столбцов будут взяты из первой строки файла, в которой они и указаны. *(Если в силу каких-то причин из Интернета не загружается, то скачайте вручную, исправьте url на file, чтобы загрузить уже скаченную копию таблицы или скачайте с другого адреса.)*


```

#url = 'https://datahub.io/core/s-and-p-500-companies-financials/r/constituents-financials.csv'
url = 'https://raw.githubusercontent.com/datasets/s-and-p-500-companies-financials/main/data/constituents-f

```




```
file='constituents-financials_csv.csv'
data = pd.read_csv(url, sep=',')
data
```




| | Symbol | Name | Sector | Price | Price/Earnings | Dividend | Earnings/Share | 52 Week Low | 52 Week High |
|-----|--------|------------------------|------------------------|--------|----------------|----------|----------------|-------------|--------------|
| 0 | MMM | 3M Company | Industrials | 222.89 | 24.31 | 2.332862 | 7.92 | 259.77 | 175.490 |
| 1 | AOS | A.O. Smith Corp | Industrials | 60.24 | 27.76 | 1.147959 | 1.70 | 68.39 | 48.925 |
| 2 | ABT | Abbott Laboratories | Health Care | 56.27 | 22.51 | 1.908982 | 0.26 | 64.60 | 42.280 |
| 3 | ABBV | AbbVie Inc. | Health Care | 108.48 | 19.41 | 2.499560 | 3.29 | 125.86 | 60.050 |
| 4 | ACN | Accenture plc | Information Technology | 150.51 | 25.47 | 1.714470 | 5.44 | 162.60 | 114.820 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 500 | XYL | Xylem Inc. | Industrials | 70.24 | 30.94 | 1.170079 | 1.83 | 76.81 | 46.860 |
| 501 | YUM | Yum! Brands Inc | Consumer Discretionary | 76.30 | 27.25 | 1.797080 | 4.07 | 86.93 | 62.850 |
| 502 | ZBH | Zimmer Biomet Holdings | Health Care | 115.53 | 14.32 | 0.794834 | 9.01 | 133.49 | 108.170 |
| 503 | ZION | Zions Bancorp | Financials | 50.71 | 17.73 | 1.480933 | 2.60 | 55.61 | 38.430 |
| 504 | ZTS | Zoetis | Health Care | 71.51 | 32.80 | 0.682372 | 1.65 | 80.13 | 52.000 |

505 rows × 14 columns



```
data.shape # сколько строк и столбцов?
```



```
(505, 14)
```

```
# выводит первые несколько строк
data.head()
```



| | Symbol | Name | Sector | Price | Price/Earnings | Dividend Yield | Earnings/Share | 52 Week Low | 52 Week High | M |
|---|--------|---------------------|------------------------|--------|----------------|----------------|----------------|-------------|--------------|-----|
| 0 | MMM | 3M Company | Industrials | 222.89 | 24.31 | 2.332862 | 7.92 | 259.77 | 175.490 | 138 |
| 1 | AOS | A.O. Smith Corp | Industrials | 60.24 | 27.76 | 1.147959 | 1.70 | 68.39 | 48.925 | 10 |
| 2 | ABT | Abbott Laboratories | Health Care | 56.27 | 22.51 | 1.908982 | 0.26 | 64.60 | 42.280 | 102 |
| 3 | ABBV | AbbVie Inc. | Health Care | 108.48 | 19.41 | 2.499560 | 3.29 | 125.86 | 60.050 | 181 |
| 4 | ACN | Accenture plc | Information Technology | 150.51 | 25.47 | 1.714470 | 5.44 | 162.60 | 114.820 | 98 |

как называются столбцы?

```
data.columns
```



```
Index(['Symbol', 'Name', 'Sector', 'Price', 'Price/Earnings', 'Dividend Yield',  
      'Earnings/Share', '52 Week Low', '52 Week High', 'Market Cap', 'EBITDA',  
      'Price/Sales', 'Price/Book', 'SEC Filings'],  
      dtype='object')
```

количество строк

```
len(data)
```



```
505
```

из них строк с компаниями из отрасли Industrials (столбец Sector)

```
induk=data['Sector']=='Industrials'
```

```
sum(induk)
```



```
67
```

Примечание: так как false интерпретируется как 0, а true как 1, то sum() и дает число истинных элементов.

✓ Группировка данных

Данные можно сгруппировать по различным критериям, за это отвечает метод [groupby\(\)](#), к результатам которого можно применять разные функции.

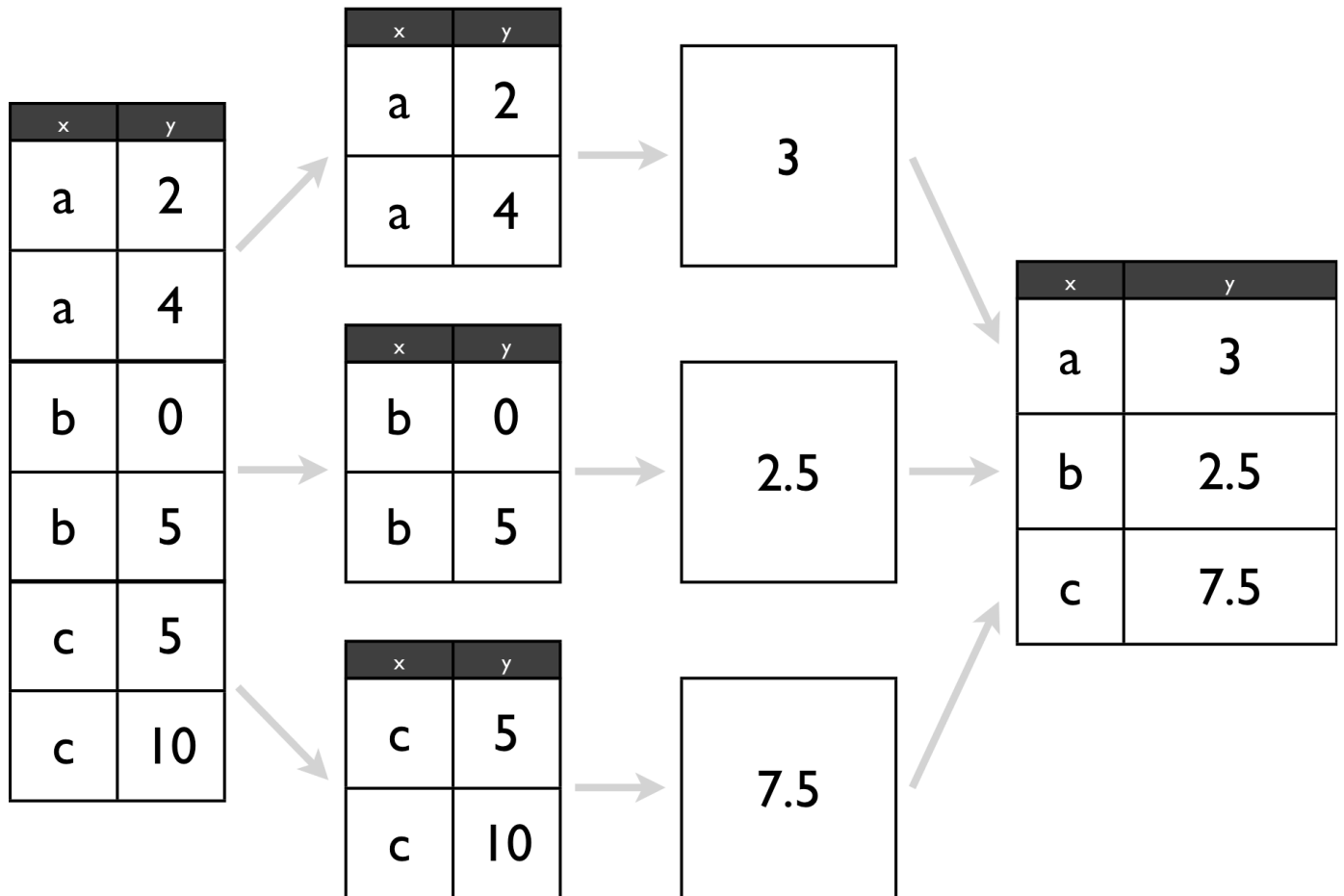
Общий подход показан на рисунке:

данные разбиваются построчно по уникальным значениям (ключ, на рисунке столбец x), строки с одинаковыми ключами объединяются в группы и к группам применяются заданные функции (подсчет среднего на рисунке для столбца y), затем результаты объединяются.

Split

Apply

Combine



```
# группируем по значениям столбцов Sector и
group=data.groupby('Sector')
group.size() # считаем количество строк, оказавшихся в каждой группе
```



0

Sector


| | |
|----------------------------|----|
| Consumer Discretionary | 84 |
| Consumer Staples | 34 |
| Energy | 32 |
| Financials | 68 |
| Health Care | 61 |
| Industrials | 67 |
| Information Technology | 70 |
| Materials | 25 |
| Real Estate | 33 |
| Telecommunication Services | 3 |
| Utilities | 28 |

```
data.count()
```



| | 0 |
|----------------|-----|
| Symbol | 505 |
| Name | 505 |
| Sector | 505 |
| Price | 505 |
| Price/Earnings | 503 |
| Dividend Yield | 505 |
| Earnings/Share | 505 |
| 52 Week Low | 505 |
| 52 Week High | 505 |
| Market Cap | 505 |
| EBITDA | 505 |
| Price/Sales | 505 |
| Price/Book | 497 |
| SEC Filings | 505 |

```
group.count() # число раз, когда значение группы встречалось
# может быть меньше, чем число строк, если значение было пропущено в файле
```



| | Symbol | Name | Price | Price/Earnings | Dividend Yield | Earnings/Share | 52 Week Low | 52 Week High | Market Cap | EB: |
|----------------------------|--------|------|-------|----------------|----------------|----------------|-------------|--------------|------------|-----|
| Sector | | | | | | | | | | |
| Consumer Discretionary | 84 | 84 | 84 | 83 | 84 | 84 | 84 | 84 | 84 | |
| Consumer Staples | 34 | 34 | 34 | 34 | 34 | 34 | 34 | 34 | 34 | |
| Energy | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | |
| Financials | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | 68 | |
| Health Care | 61 | 61 | 61 | 60 | 61 | 61 | 61 | 61 | 61 | |
| Industrials | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | |
| Information Technology | 70 | 70 | 70 | 70 | 70 | 70 | 70 | 70 | 70 | |
| Materials | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | |
| Real Estate | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | |
| Telecommunication Services | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | |
| Utilities | 28 | 28 | 28 | 28 | 28 | 28 | 28 | 28 | 28 | |

```
#group.Price.median() # медиана для групп по числовому столбцу Price
# тут-то нам и пригодилось обращение к столбцу как к атрибуту
```

```
group['Price'].median() # альтернативное написание
```



| Sector | Price |
|----------------------------|---------|
| Consumer Discretionary | 65.140 |
| Consumer Staples | 69.750 |
| Energy | 50.325 |
| Financials | 71.105 |
| Health Care | 96.420 |
| Industrials | 87.600 |
| Information Technology | 84.770 |
| Materials | 105.180 |
| Real Estate | 58.360 |
| Telecommunication Services | 35.570 |
| Utilities | 49.660 |



```
# каждую группу можно посмотреть, вот состав группы Materials
group.get_group('Materials')
```



| | Symbol | Name | Sector | Price | Price/Earnings | Dividend Yield | Earnings/Share | 52 Week Low | 52 W H |
|-----|--------|------------------------------|-----------|--------|----------------|----------------|----------------|-------------|--------|
| 15 | APD | Air Products & Chemicals Inc | Materials | 152.80 | 24.22 | 2.781114 | 13.66 | 175.1700 | 133.6 |
| 18 | ALB | Albemarle Corp | Materials | 105.18 | 26.03 | 1.200413 | 5.66 | 144.9900 | 90.3 |
| 63 | AVY | Avery Dennison Corp | Materials | 110.77 | 22.11 | 1.568218 | 3.11 | 123.6700 | 78.4 |
| 65 | BLL | Ball Corp | Materials | 38.44 | 20.56 | 1.017035 | 0.85 | 43.2400 | 35.6 |
| 101 | CF | CF Industries Holdings Inc | Materials | 37.46 | -59.46 | 3.039514 | -1.20 | 43.9800 | 25.0 |
| 154 | DWDP | DowDuPont | Materials | 68.21 | 49.43 | 2.152975 | 1.59 | 77.0800 | 64.0 |
| 161 | EMN | Eastman Chemical | Materials | 93.57 | 12.28 | 2.263084 | 10.12 | 104.0800 | 76.0 |
| 164 | ECL | Ecolab Inc. | Materials | 127.76 | 28.08 | 1.231971 | 4.14 | 140.5000 | 119.6 |
| 198 | FMC | FMC Corporation | Materials | 80.87 | 32.48 | 0.785995 | 1.56 | 98.7000 | 56.5 |
| 204 | FCX | Freeport-McMoRan Inc. | Materials | 17.16 | 14.67 | 1.119821 | 1.24 | 20.2500 | 11.0 |
| 250 | IP | International Paper | Materials | 56.05 | 15.57 | 3.206751 | 5.14 | 66.9400 | 49.6 |
| 252 | IFF | Intl Flavors & Fragrances | Materials | 138.00 | 24.17 | 1.934128 | 5.05 | 157.4000 | 116.3 |
| 288 | LYB | LyondellBasell | Materials | 105.79 | 10.35 | 3.264714 | 12.25 | 121.9500 | 78.0 |
| 296 | MLM | Martin Marietta Materials | Materials | 208.42 | 30.38 | 0.804204 | 6.63 | 244.3200 | 191.0 |

Уникальные (т.е. без повторов) названия (столбец Name) компаний в каждой группе.

group.Name.unique()



Name

| Sector | |
|----------------------------|---|
| Consumer Discretionary | [Advance Auto Parts, Amazon.com Inc, Aptiv Plc... |
| Consumer Staples | [Altria Group Inc, Archer-Daniels-Midland Co, ... |
| Energy | [Anadarko Petroleum Corp, Andeavor, Apache Cor... |
| Financials | [Affiliated Managers Group Inc, AFLAC Inc, All... |
| Health Care | [Abbott Laboratories, AbbVie Inc., Aetna Inc, ... |
| Industrials | [3M Company, A.O. Smith Corp, Acuity Brands In... |
| Information Technology | [Accenture plc, Activision Blizzard, Adobe Sys... |
| Materials | [Air Products & Chemicals Inc, Albemarle Corp,... |
| Real Estate | [Alexandria Real Estate Equities Inc, American... |
| Telecommunication Services | [AT&T Inc, CenturyLink Inc, Verizon Communicat... |
| Utilities | [AES Corp, Alliant Energy Corp, Ameren Corp, A... |



Каждая группа здесь это кортеж названий и содержания (последовательность) группы.

Посмотрим, есть ли в группе хоть одна фирма с аббревиатурой (столбец Symbol) начинающейся на букву "A".

```
letter="A"
for i in group.Symbol:
    print('{}: {}'.format(i[0], (i[1].str.get(0)==letter).any()))
```



```
Consumer Discretionary: True
Consumer Staples: True
Energy: True
Financials: True
Health Care: True
Industrials: True
Information Technology: True
Materials: True
Real Estate: True
Telecommunication Services: False
Utilities: True
```

Что здесь происходит:

- группы можно итерировать, в цикле переменная `i` будет являться последовательно кортежем из названия и содержание каждой из групп в которой оставлен только столбец Symbol.
- `i[0]` - название группы, `i[1]` - содержимое текущей группы, в нашем случае это объект типа Series.
- у этого объекта Series есть атрибут `str`, который позволяет работать с элементами как со строкой.
- метод `get()` для строки возвращает символ строки на определенном месте, 0 - это первый символ.
- сравниваем этот первый символ с буквой "A", получаем или истину или ложь.
- так как в группе может быть не одна строка, то в результате получится логический вектор (тип Series) из значений истина\ложь.
- метод `any()` для логических векторов возвращает истину если хотя бы один элемент вектора истинный, что нам и требуется.
- наконец печатаем результат.

Агрегировать группы можно разными функциями, даже своими собственными, в этом помогает метод [agg\(\)](#), который позволяет применять одну или несколько функций к группам. Давайте сделаем свою функцию и применим ее к группам.

```
# своя функция, которая вычитает минимальное значение из максимального
def max_min(arr):
    return arr.max() - arr.min()

# считаем по группам для столбца Price свою функцию и среднее.
result=group.Price.agg([max_min, 'mean'])
result
```



| | max_min | mean |
|----------------------------|---------|------------|
| Sector | | |
| Consumer Discretionary | 1795.63 | 124.034524 |
| Consumer Staples | 188.77 | 79.764118 |
| Energy | 166.34 | 57.887500 |
| Financials | 496.00 | 89.056029 |
| Health Care | 575.80 | 132.515738 |
| Industrials | 319.85 | 116.887612 |
| Information Technology | 996.49 | 119.242857 |
| Materials | 370.49 | 102.386800 |
| Real Estate | 395.97 | 88.712727 |
| Telecommunication Services | 32.84 | 33.603333 |
| Utilities | 125.22 | 55.101612 |

✓ Разбивка на интервалы

Значения в столбце можно разбить на интервалы, назвать их и записать эти интервалы вместо значений. В этом поможет команда [cut](#) которой указываем что разбивать, сколько интервалов сделать, их названия и другие аргументы.

```
data['Price']
```




| | Price |
|---|--------|
| 0 | 222.89 |
| 1 | 60.24 |
| 2 | 56.27 |

```
# разбиваем столбец Price на 5 интервалов, называем их.  
cuted=pd.cut(data['Price'], 5,labels=['Low','Medium','High','Very High','Exclusive'], include_lowest=True)  
cuted
```



| | Price |
|-----|-------|
| 0 | Low |
| 1 | Low |
| 2 | Low |
| 3 | Low |
| 4 | Low |
| ... | ... |
| 500 | Low |
| 501 | Low |
| 502 | Low |
| 503 | Low |
| 504 | Low |

505 rows × 1 columns



Заменим столбец Price на cuted

```
data['Price']=cuted  
data
```



| | Symbol | Name | Sector | Price | Price/Earnings | Dividend Yield | Earnings/Share | 52 Week Low | 52 Week High |
|---|--------|------|------------|--------|----------------|----------------|----------------|-------------|--------------|
| 0 | MMM | 3M | Technology | 100.00 | 10.00 | 0.000000 | 10.00 | 100.00 | 100.00 |