



# Библиотеки обработки данных для языка Python

ИУ-5

# Краткий план лекции

- Структуры данных в машинном обучении
- Библиотеки для обработки данных
  - NumPy, SciPy
  - Разреженные матрицы
  - Pandas
  - PandaSQL
  - Dask

# Структуры данных в машинном обучении

# Тензоры

- Тензор – основная структура данных в машинном обучении.
- Существует два определения тензора
  1. «Старое» определение. Объект линейной алгебры, линейно преобразующий элементы одного линейного пространства в элементы другого.
  2. **«Новое» определение, которое почти всегда используется в машинном обучении. Просто многомерная матрица.**
- Существует гипотеза, что большинство алгоритмов машинного обучения можно представить в виде последовательных преобразований тензоров. Гипотеза реализована в библиотеке TensorFlow.

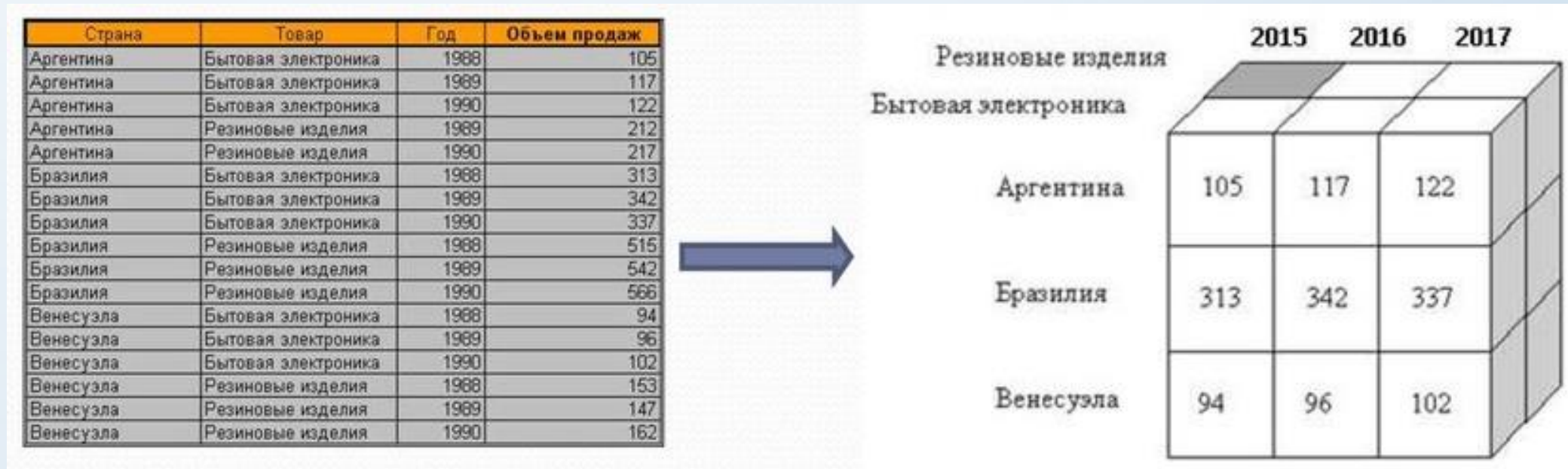
# Факторизация тензоров

- Большинство тензоров очень разрежены (содержат много пустых значений).
- Факторизация тензора – его представление в виде произведения простых объектов (матрицы и тензоров меньшей размерности), которые не являются разреженными:
  - <https://habr.com/ru/company/yandex/blog/313892/> (проф. Иван Оселедец, [премия](#))
  - <https://pdfs.semanticscholar.org/94cc/6daad548a03c6edb0351d686c2d4aa364634.pdf> (проф. Andrzej Cichocki)
- Для реляционных БД аналогом операции факторизации является нормализация схемы БД.
  - В этом случае мы не всегда уменьшаем размер нормализованных таблицы, но всегда превращаем исходную денормализованную таблицу в набор «неразрезанных» нормализованных таблиц.
  - Вместо произведения матриц (для факторизации) используется соединение на основе ключей (join).
- Таким образом, превращение денормализованной структуры в набор нормализованных составных частей (которые удобнее хранить и обрабатывать) является устойчивым «паттерном» анализа данных.
- Но для построения моделей машинного обучения удобнее использовать денормализованную таблицу (которая может быть результатом сборки факторизованного тензора или результатом соединения реляционных таблиц).



# Тензоры и OLAP-кубы

- OLAP-куб является разновидностью тензора.
- Каждый OLAP-куб (справа) может быть представлен в виде денормализованной таблицы (слева). Дополнительный пример.



- Тензорное (кубическое) представление больше соответствует библиотеке NumPy, а денормализованное – библиотеке Pandas.

# Тензоры и OLAP-кубы (2)

- Тензор является прежде всего математической моделью, в то время как OLAP-куб – инженерной.
- В отличие от OLAP-кубов для тензоров не определено естественных операций агрегирования.

# Выводы по разделу:

- Тензор – основная структура данных в машинном обучении, сейчас понимается как многомерная матрица.
- Тензор и OLAP-куб являются «родственными моделями». Для обеих моделей можно использовать срезы и уменьшение размерностей. Но в отличие от OLAP-кубов для тензоров не определено естественных операций агрегирования.
- Тензор и OLAP-куб могут быть представлены в форме денормализованной таблицы.
- Для построения моделей машинного обучения удобнее использовать денормализованную таблицу (которая может быть получена из тензора, OLAP-куба или как результат соединения реляционных таблиц).
- В дальнейшем для построения моделей машинного обучения будут использоваться именно денормализованные таблицы.



# Библиотеки для обработки данных

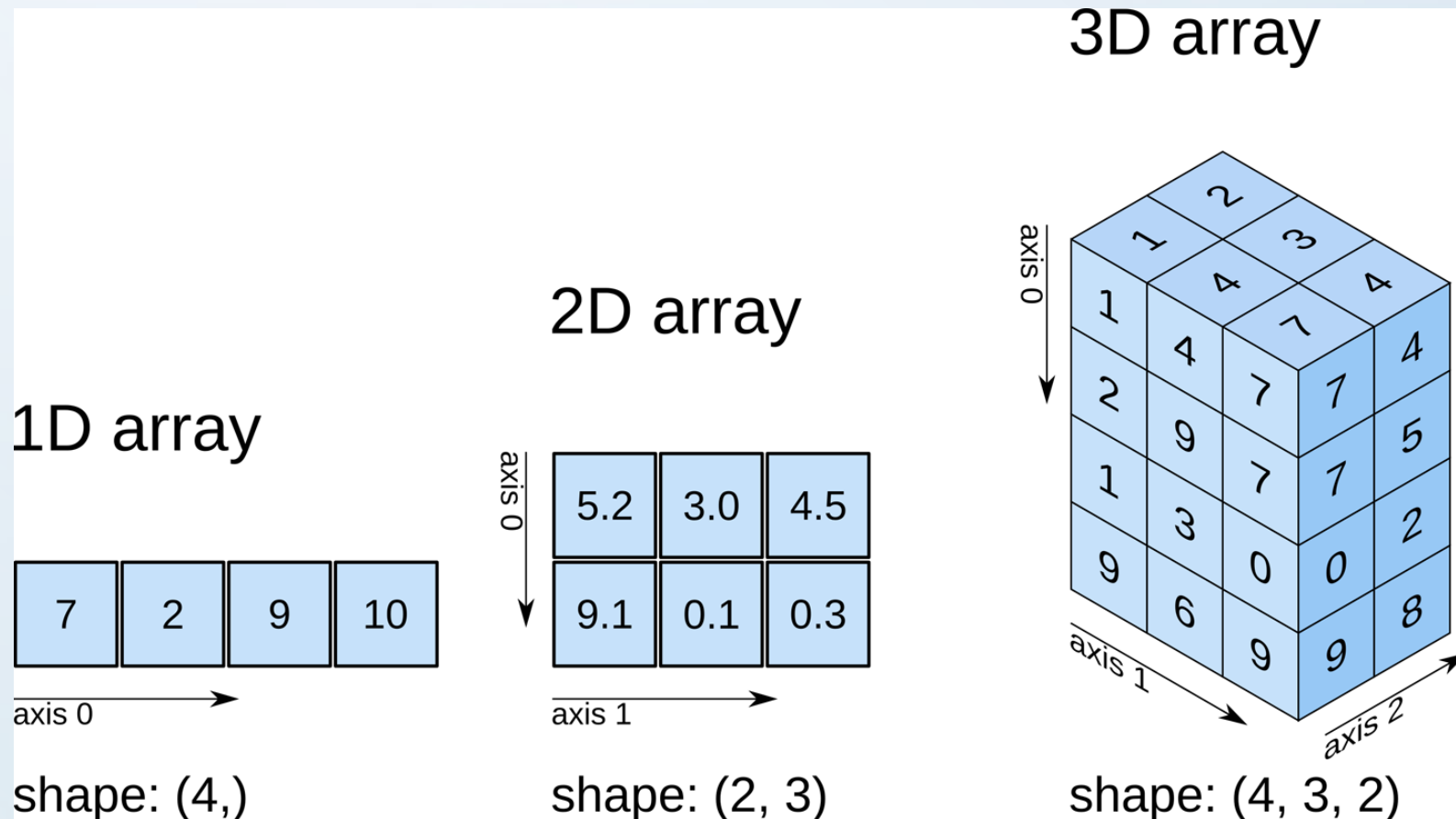
# Рекомендуемая книга

- Библиотеки NumPy и Pandas хорошо описаны в книге Дж. Вандер Пласа.



# Библиотека NumPy (1)

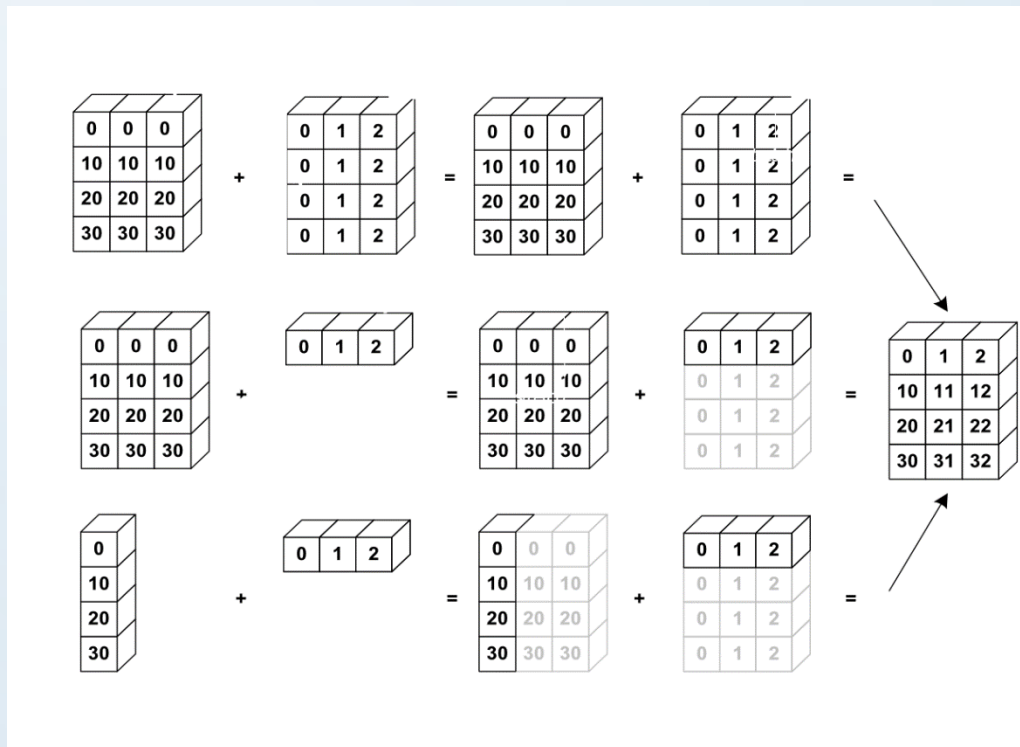
- Библиотека [NumPy](#) предназначена для научных вычислений. Основным типом данных является тензор (многомерная матрица).



[ИСТОЧНИК](#)

# Библиотека NumPy (2)

- Векторизация вычислений - <https://en.wikipedia.org/wiki/Vectorization>
- Векторизация в NumPy - [https://www.pythonlikeyoumeanit.com/Module3\\_IntroducingNumpy/VectorizedOperations.html](https://www.pythonlikeyoumeanit.com/Module3_IntroducingNumpy/VectorizedOperations.html)
- Broadcasting (расширение, транслирование, сопряжение размерностей матриц) - <https://scipy-lectures.org/intro/numpy/operations.html#broadcasting>



# Библиотека NumPy (3)

- Сайт библиотеки - <http://www.numpy.org/>
- Документация - <https://numpy.org/doc/>
- Введение на русском языке - <https://habr.com/ru/post/352678/>
- Введение - <https://docs.scipy.org/doc/numpy/user/quickstart.html>
- Другие тьюториалы по NumPy:
  - Часть 1 - <https://www.machinelearningplus.com/python/numpy-tutorial-part1-array-python-examples/>
  - Часть 2 - <https://www.machinelearningplus.com/python/numpy-tutorial-python-part2/>
  - 101 упражнение по NumPy - <https://www.machinelearningplus.com/python/101-numpy-exercises-python/>
  - 100 упражнение по NumPy - <http://www.labri.fr/perso/nrougier/teaching/numpy.100/>

# Библиотека SciPy

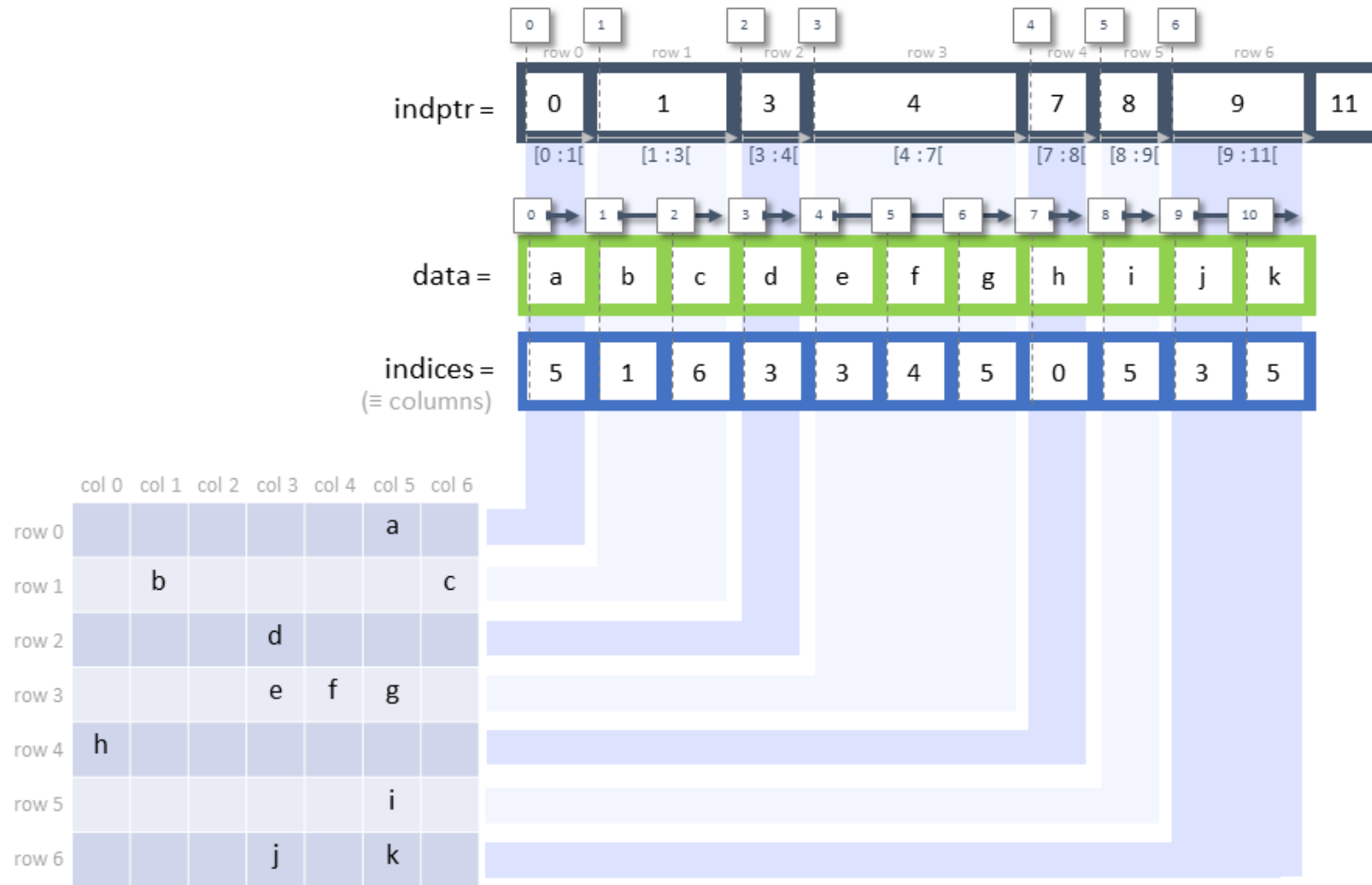
- SciPy — «библиотека для языка программирования Python с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов» (Википедия).
- Основная структура данных – массив NumPy.
- Сайт библиотеки – <https://www.scipy.org/>
- Документация – <https://docs.scipy.org/doc/scipy/>
- Введение на русском языке – <https://coderlessons.com/tutorials/python-technologies/uchitsia-stsipi/scipy-kratkoe-rukovodstvo> (есть опечатки при переводе)
- Пособие по решению оптимизационных задач – <https://habr.com/ru/post/439288/>



# Разреженные матрицы (SciPy)

- Если набор данных слишком велик и содержит много пустых значений, то можно использовать разреженные матрицы:
  - <https://docs.scipy.org/doc/scipy/reference/sparse.html>
  - [http://scipy-lectures.org/advanced/scipy\\_sparse/index.html](http://scipy-lectures.org/advanced/scipy_sparse/index.html)
  - <https://rushter.com/blog/scipy-sparse-matrices/> (простое введение)
  - <https://mattedding.github.io/2019/04/25/sparse-matrices/> (иллюстрированное руководство)

# SciPy, формат csr, кодирование



# Библиотека Pandas (1)

- Pandas - это библиотека, предназначенная для чтения, хранения, записи и обработки наборов данных (датасетов).
- Фактически является in-memory СУБД с колоночной моделью данных.
- Основная структура данных в Pandas это денормализованная таблица данных. Pandas одновременно обладает некоторыми характеристиками электронной таблицы (Excel) и реляционной СУБД.
- В чем Pandas похож на электронную таблицу:
  - В Pandas основной структурой данных является одна таблица (а не схема связанных таблиц как в реляционной СУБД).
  - Таблица должна содержать полный набор данных, предназначенных для дальнейшего построения моделей машинного обучения.
  - Таблица данных, как правило, является денормализованной.
  - В данных могут быть пропущенные значения.
- В чем Pandas похож на реляционную СУБД:
  - Таблица данных (как и реляционная таблица) состоит из типизированных столбцов (атрибутов).
  - Строка таблицы соответствует записи в реляционной БД.
  - Над таблицами возможно выполнение операций реляционной алгебры – соединение (join), группировка и другие операции.

# Библиотека Pandas (2)

- Официальная документация - <http://pandas.pydata.org/pandas-docs/stable/>
- Русскоязычные руководства:
  - <https://habr.com/ru/company/ods/blog/322626/>
  - <https://pythonworld.ru/obrabotka-dannyx/pandas-cookbook-0-ipython.html>
  - <https://python.ivan-shamaev.ru/pandas-series-and-dataframe-objects-build-index/>
  - <https://coderlessons.com/tutorials/python-technologies/vyuchit-python-panda/python-pandas-kratkoe-rukovodstvo>
  - <https://pythonru.com/uroki/osnovy-pandas-1-chtenie-fajlov-dataframe-otbor-dannyh>
  - <https://tproger.ru/articles/pandas-data-wrangling-cheatsheet/>
- Интерактивное руководство - <https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python>
- 101 упражнение по Pandas - <https://www.machinelearningplus.com/python/101-pandas-exercises-python/>
- Pandas не поддерживает параллельную обработку наборов данных. Это замедляет обработку данных при больших объемах данных. Для решения этой проблемы может быть использована библиотека Dask.

# Библиотека Pandas (3)

## Data Wrangling with pandas Cheat Sheet <http://pandas.pydata.org>

### Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(  
    {"a" : [4 ,5, 6],  
     "b" : [7, 8, 9],  
     "c" : [10, 11, 12]},  
    index = [1, 2, 3])  
Specify values for each column.
```

```
df = pd.DataFrame(  
    [[4, 7, 10],  
     [5, 8, 11],  
     [6, 9, 12]],  
    index=[1, 2, 3],  
    columns=['a', 'b', 'c'])  
Specify values for each row.
```

	a	b	c
n	1	4	7
d	2	5	8
e	2	6	9

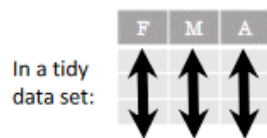
```
df = pd.DataFrame(  
    {"a" : [4 ,5, 6],  
     "b" : [7, 8, 9],  
     "c" : [10, 11, 12]},  
    index = pd.MultiIndex.from_tuples(  
        [('d',1), ('d',2), ('e',2)],  
        names=['n', 'v']))  
Create DataFrame with a MultiIndex
```

### Method Chaining

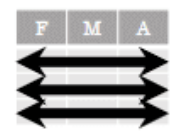
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)  
     .rename(columns={  
         'variable' : 'var',  
         'value' : 'val'})  
     .query('val >= 200'))
```

## Tidy Data – A foundation for wrangling in pandas

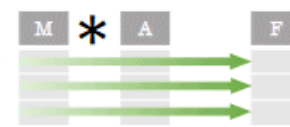


Each **variable** is saved  
in its own **column**



Each **observation** is  
saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

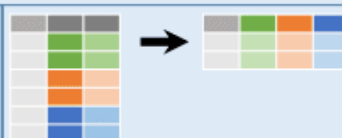


M \* A

### Reshaping Data – Change the layout of a data set



`pd.melt(df)`  
Gather columns into rows.



`df.pivot(columns='var', values='val')`  
Spread rows into columns.



`pd.concat([df1, df2])`  
Append rows of DataFrames



`pd.concat([df1, df2], axis=1)`  
Append columns of DataFrames

`df.sort_values('mpg')`  
Order rows by values of a column (low to high).

`df.sort_values('mpg', ascending=False)`  
Order rows by values of a column (high to low).

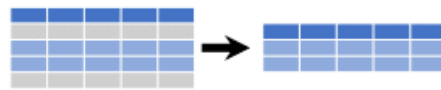
`df.rename(columns = {'y':'year'})`  
Rename the columns of a DataFrame

`df.sort_index()`  
Sort the index of a DataFrame

`df.reset_index()`  
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length', 'Height'])`  
Drop columns from DataFrame

### Subset Observations (Rows)



`df[df.Length > 7]`  
Extract rows that meet logical criteria.

`df.drop_duplicates()`  
Remove duplicate rows (only considers columns).

`df.head(n)`  
Select first n rows.

`df.tail(n)`  
Select last n rows.

`df.sample(frac=0.5)`  
Randomly select fraction of rows.

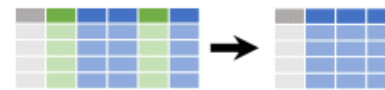
`df.sample(n=10)`  
Randomly select n rows.

`df.iloc[10:20]`  
Select rows by position.

`df.nlargest(n, 'value')`  
Select and order top n entries.

`df.nsmallest(n, 'value')`  
Select and order bottom n entries.

### Subset Variables (Columns)



`df[['width', 'length', 'species']]`

Select multiple columns with specific names.

`df['width']` or `df.width`

Select single column with specific name.

`df.filter(regex='regex')`

Select columns whose name matches regular expression `regex`.

#### regex (Regular Expressions) Examples

regex	Examples
<code>'\.'</code>	Matches strings containing a period '.'
<code>'Length\$'</code>	Matches strings ending with word 'Length'
<code>'^Sepal'</code>	Matches strings beginning with the word 'Sepal'
<code>'^x[1-5]'</code>	Matches strings beginning with 'x' and ending with 1,2,3,4,5
<code>'^(?!Species\$)'</code>	Matches strings except the string 'Species'

`df.loc[:, 'x2': 'x4']`  
Select all columns between x2 and x4 (inclusive).

`df.iloc[:, [1, 2, 5]]`  
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a', 'c']]`  
Select rows meeting logical condition, and only the specific columns.

Logic in Python (and pandas)		
<code>&lt;</code>	Less than	<code>!=</code> Not equal to
<code>&gt;</code>	Greater than	<code>df.column.isin(values)</code> Group membership
<code>==</code>	Equals	<code>pd.isnull(obj)</code> Is NaN
<code>&lt;=</code>	Less than or equals	<code>pd.notnull(obj)</code> Is not NaN
<code>&gt;=</code>	Greater than or equals	<code>&amp;,  , ~, ^, df.any(), df.all()</code> Logical and, or, not, xor, any, all



# Библиотека Pandas (4)

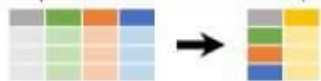
## Summarize Data

**df['w'].value\_counts()**  
Count number of rows with each unique value of variable

**len(df)**  
# of rows in DataFrame.

**df['w'].nunique()**  
# of distinct values in a column.

**df.describe()**  
Basic descriptive statistics for each column (or GroupBy)



pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

<b>sum()</b> Sum values of each object.	<b>min()</b> Minimum value in each object.
<b>count()</b> Count non-NA/null values of each object.	<b>max()</b> Maximum value in each object.
<b>median()</b> Median value of each object.	<b>mean()</b> Mean value of each object.
<b>quantile([0.25,0.75])</b> Quantiles of each object.	<b>var()</b> Variance of each object.
<b>apply(function)</b> Apply function to each object.	<b>std()</b> Standard deviation of each object.

## Group Data



**df.groupby(by="col")**  
Return a GroupBy object, grouped by values in column named "col".

**df.groupby(level="ind")**  
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.

Additional GroupBy functions:

**size()**  
Size of each group.

**agg(function)**  
Aggregate group using function.

## Windows

**df.expanding()**  
Return an Expanding object allowing summary functions to be applied cumulatively.

**df.rolling(n)**  
Return a Rolling object allowing summary functions to be applied to windows of length n.

## Handling Missing Data

**df.dropna()**  
Drop rows with any column having NA/null data.

**df.fillna(value)**  
Replace all NA/null data with value.

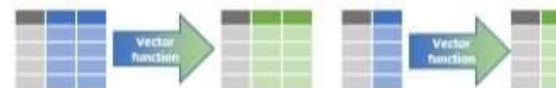
## Make New Columns



**df.assign(Area=lambda df: df.Length\*df.Height)**  
Compute and append one or more new columns.

**df['Volume'] = df.Length\*df.Height\*df.Depth**  
Add single column.

**pd.qcut(df.col, n, labels=False)**  
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

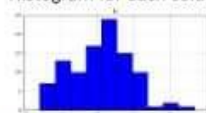
<b>max(axis=1)</b> Element-wise max.	<b>min(axis=1)</b> Element-wise min.
<b>clip(lower=-10,upper=10)</b> Trim values at input thresholds	<b>abs()</b> Absolute value.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

<b>shift(1)</b> Copy with values shifted by 1.	<b>shift(-1)</b> Copy with values lagged by 1.
<b>rank(method='dense')</b> Ranks with no gaps.	<b>cumsum()</b> Cumulative sum.
<b>rank(method='min')</b> Ranks. Ties get min rank.	<b>cummax()</b> Cumulative max.
<b>rank(pct=True)</b> Ranks rescaled to interval [0, 1].	<b>cummin()</b> Cumulative min.
<b>rank(method='first')</b> Ranks. Ties go to first value.	<b>cumprod()</b> Cumulative product.

## Plotting

**df.plot.hist()**  
Histogram for each column



**df.plot.scatter(x='w',y='h')**  
Scatter chart using pairs of points



## Combine Data Sets



Standard Joins

**pd.merge(adf, bdf, how='left', on='x1')**  
Join matching rows from bdf to adf.

**pd.merge(adf, bdf, how='right', on='x1')**  
Join matching rows from adf to bdf.

**pd.merge(adf, bdf, how='inner', on='x1')**  
Join data. Retain only rows in both sets.

**pd.merge(adf, bdf, how='outer', on='x1')**  
Join data. Retain all values, all rows.

Filtering Joins

**adf[adf.x1.isin(bdf.x1)]**  
All rows in adf that have a match in bdf.

**adf[~adf.x1.isin(bdf.x1)]**  
All rows in adf that do not have a match in bdf.



Set-like Operations

**pd.merge(ydf, zdf)**  
Rows that appear in both ydf and zdf (Intersection).

**pd.merge(ydf, zdf, how='outer')**  
Rows that appear in either or both ydf and zdf (Union).

**pd.merge(ydf, zdf, how='outer', indicator=True)**  
**.query('\_merge == "left\_only"')**  
**.drop(columns=['\_merge'])**  
Rows that appear in ydf but not zdf (Setdiff).



# Библиотека PandaSQL

- Предназначена для выполнения SQL-запросов над наборами данных Pandas.
- Официальный сайт - <https://github.com/yhat/pandasql>
- Библиотека использует синтаксис SQLite - <https://www.sqlite.org/lang.html>
- Примеры использования:
  - <https://habr.com/ru/post/279213/>

# Библиотека Dask

- Библиотека Dask предназначена для распараллеливания вычислений при обработке данных.
- Может рассматриваться как «Pandas с распараллеливанием обработки данных».
- Преимущества библиотеки - <http://docs.dask.org/en/latest/why.html>
- Распараллеливания вычислений приводит к более сложной интеграции с библиотеками машинного обучения.
- Сравнения с библиотеками для обработки больших данных:
  - <http://docs.dask.org/en/latest/spark.html>
  - <https://matthewrocklin.com/blog/work/2018/08/28/dataframe-performance-high-level>
- Dask сложнее чем Pandas использовать с библиотеками машинного обучения.
  - <https://examples.dask.org/machine-learning.html>

# Библиотека SymPy

- Реализует «символическую (символьную) математику» - <https://en.wikipedia.org/wiki/SymPy>
- Официальный сайт - <https://www.sympy.org/>
- Документация - <https://docs.sympy.org/latest/index.html>
- Введение на русском языке - <https://habr.com/ru/post/423731/>
- Модуль генерации кода - <https://docs.sympy.org/latest/modules/utilities/codegen.html>
- Этот фреймворк не следует путать с системой моделирования SimPy - <https://ru.wikipedia.org/wiki/SimPy>

# Восстановление моделей на основе «Символьной математики»

- Генетический алгоритм
- Генетическое программирование
- Символьная регрессия
- Библиотека gplearn - <https://gplearn.readthedocs.io/>
- Библиотека SoRa - <https://github.com/LLNL/SoRa>
- Пример использования библиотеки gplearn