
ФИНАНСОВЫЙ
УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

Применение Python для анализа данных, методы pandas

Знакомство с Python

Напомним:

В Python есть три самых полезных и часто используемых вида структуры данных: кортеж (tuple), список (list) и словарь (dictionary).

Кортеж (Tuple)

Кортеж – это данные, доступные только для чтения

Code

```
a = (1, 2, 3)
print(a)
```

Output

```
(1, 2, 3)
```

Структуры данных
Список

Чтобы задать списки, используются квадратные скобки или форма записи массива. Обратите внимание, что мы используем простой функционал, аналогичный print, чтобы комбинировать строки и переменные при выводе на экран.

Code

```
mylist = [1, 2, 3]
print("Zeroth Value: %d" % mylist[0])
mylist.append(4)
print("List Length: %d" % len(mylist))
for value in mylist:
print(value)
```

Output

```
Zeroth Value: 1
List Length: 4
1 2 3 4
```

Структуры данных

Словари – это сопоставления имен со значениями, например, пары ключ-значение. Обратите внимание, что для записи словарей используются фигурные скобки и двоеточие

Code

```
mydict = {'a': 1, 'b': 2, 'c': 3}
print("A value: %d" % mydict['a'])
mydict['a'] = 11
print("A value: %d" % mydict['a'])
print("Keys: %s" % mydict.keys())
print("Values: %s" % mydict.values())
for key in mydict.keys():
print(mydict[key])
```

Output

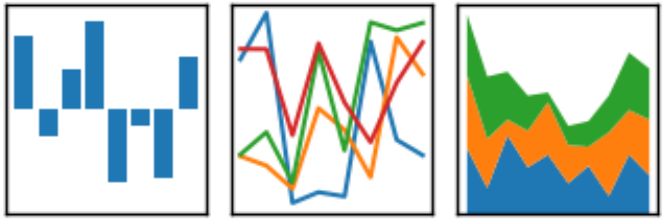
```
A value: 1
A value: 11
Keys: dict_keys(['a', 'b', 'c'])
Values: dict_values([11, 2, 3])
11
2
3
```

Введение в Pandas

- Библиотека для вычислений с табличными данными
- Смешанные типы данных разрешены в одной таблице
- Могут быть поименованы столбцы и строки данных
- Расширенные функции агрегирования данных и статистические функции

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Source: http://pandas.pydata.org/

Введение

Библиотека pandas предоставляет две структуры: **Series** и **DataFrame** для быстрой и удобной работы с данными (на самом деле их три, есть еще одна структура – Panel, но в будущем будет исключена из состава библиотеки pandas).

Series – это маркированная одномерная структура данных, ее можно представить, как таблицу с одной строкой. С Series можно работать как с обычным массивом (обращаться по номеру индекса), и как с ассоциированным массивом, когда можно использовать ключ для доступа к элементам данных.

DataFrame – это двумерная маркированная структура. Идейно она очень похожа на обычную таблицу, что выражается в способе ее создания и работе с ее элементами.
Panel – про который было сказано, что он вскоре будет исключен из pandas, представляет собой трехмерную структуру данных.

Series – одномерный похожий на массив объект, содержащий массив данных (любого типа, поддерживаемого NumPy (“числовой Python”))и ассоциированный с ним массив меток, который называется индексом.

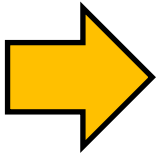
Создать структуру Series можно на базе различных типов данных:

- словари Python
- списки Python
- массивы из numpy: ndarray (многомерный однородный массив элементов фиксированного размера)
- скалярные величины

Введение в Pandas (Introduction to Pandas)

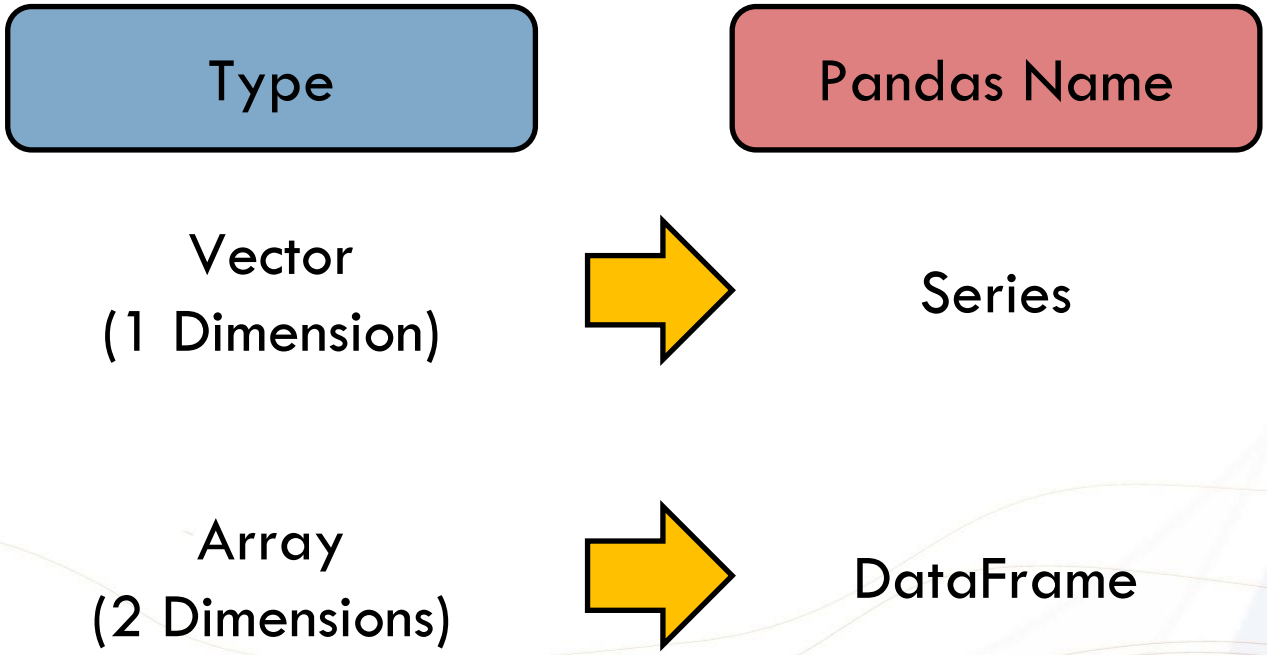
Базовые структуры (Basic data structures)

Type	Pandas Name
Vector (1 Dimension)	Series



Введение в Pandas (Introduction to Pandas)

Базовые структуры (Basic data structures)



Создание и индексирование Pandas Series (Pandas Series Creation and Indexing)

Используйте данные из приложения для отслеживания шагов, чтобы создать Pandas Series (Use data from step tracking application to create a Pandas Series)

Code

```
import pandas as pd

step_data = [3620, 7891, 9761,
             3907, 4338, 5373]

step_counts = pd.Series(step_data,
                        name='steps')
```

Полный список параметров:

pandas.Series

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

class pandas.Series(*data=None, index=None, dtype=None, name=None, copy=False, fastpath=False*)

```
print(step_counts)
```

Конструктор класса Series выглядит следующим образом:

pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)

data – массив, словарь или скалярное значение, на базе которого будет построен Series;

index – список меток, который будет использоваться для доступа к элементам Series. Длина списка должна быть равна длине data;

dtype – объект numpy.dtype, определяющий тип данных;

copy – создает копию массива данных, если параметр равен True в ином случае ничего не делает.

В большинстве случаев, при создании Series, используют только первые два параметра.

Создание и индексирование Pandas Series (Pandas Series Creation and Indexing)

Используйте данные из приложения для отслеживания шагов, чтобы создать Pandas Series (Use data from step tracking application to create a Pandas Series)

Code

```
import pandas as pd

step_data = [3620, 7891, 9761,
             3907, 4338, 5373]

step_counts = pd.Series(step_data,
                        name='steps')

print(step_counts)
```

Output

```
>>> 0 3620
    1 7891
    2 9761
    3 3907
    4 4338
    5 5373
    Name: steps, dtype: int64
```

Pandas Series Creation and Indexing

Добавить диапазон дат в Series (Add a date range to the Series)

Code

```
step_counts.index = pd.date_range('20150329',
                                  periods=6)
```

Полный список параметров:

pandas.date_range

pandas.date_range(*start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, **kwargs*) [source]

https://pandas.pydata.org/docs/reference/api/pandas.date_range.html

```
print(step_counts)
```

Pandas Series Creation and Indexing

Добавить диапазон дат в Series (Add a date range to the Series)

Code

```
step_counts.index = pd.date_range('20150329',
                                  periods=6)

print(step_counts)
```

Output

```
>>> 2015-03-29 3620
    2015-03-30 7891
    2015-03-31 9761
    2015-04-01 3907
    2015-04-02 4338
    2015-04-03 5373
    Freq: D, Name: steps,
    dtype: int64
```

Pandas Series Creation and Indexing

Выбрать данные по значениям индекса (Select data by the index values)

Code

```
# Just like a dictionary
print(step_counts['2015-04-01'])
```

Output

Pandas Series Creation and Indexing

Выбрать данные по значениям индекса (Select data by the index values)

Code

```
# Just like a dictionary (так же, как словарь)
print(step_counts['2015-04-01'])
```

Output

```
>>> 3907
```

Pandas Series Creation and Indexing

Выбрать данные по значениям индекса (Select data by the index values)

Code

```
# Just like a dictionary
print(step_counts['2015-04-01'])

# Or by index position--like an array
print(step_counts[3])
```

Output

```
>>> 3907
```

Pandas Series Creation and Indexing

Выбрать данные по значениям индекса (Select data by the index values)

Code

```
# Just like a dictionary
print(step_counts['2015-04-01'])

# Or by index position--like an array
print(step_counts[3])
```

Output

```
>>> 3907

>>> 3907
```

Pandas Series Creation and Indexing

Выбрать данные по значениям индекса (Select data by the index values)

Code

```
# Just like a dictionary
print(step_counts['2015-04-01'])

# Or by index position--like an array
print(step_counts[3])

# Select all of April
print(step_counts['2015-04'])
```

Output

```
>>> 3907

>>> 3907
```

Pandas Series Creation and Indexing

Выбрать данные по значениям индекса (Select data by the index values)

Code

```
# Just like a dictionary
print(step_counts['2015-04-01'])

# Or by index position--like an array
print(step_counts[3])

# Select all of April
print(step_counts['2015-04'])
```

Output

```
>>> 3907

>>> 3907

>>> 2015-04-01 3907
    2015-04-02 4338
    2015-04-03 5373
    Freq: D, Name: steps,
    dtype: int64
```

Типы данных Pandas и заполнение (Pandas Data Types and Imputation

Типы данных можно просматривать и преобразовывать (Data types can be viewed and converted)

Code

```
# View the data type
print(step_counts.dtypes)
```

Output

Pandas Data Types and Imputation

Типы данных можно просматривать и преобразовывать (Data types can be viewed and converted)

Code

```
# View the data type
print(step_counts.dtypes)
```

Output

```
>>> int64
```

Pandas Data Types and Imputation

Типы данных можно просматривать и преобразовывать (Data types can be viewed and converted)

Code

```
# View the data type
print(step_counts.dtypes)

# Convert to a float
step_counts = step_counts.astype(np.float)

# View the data type
print(step_counts.dtypes)
```

Output

```
>>> int64
```

Pandas Data Types and Imputation

Типы данных можно просматривать и преобразовывать (Data types can be viewed and converted)

Code

```
# View the data type
print(step_counts.dtypes)

# Convert to a float
step_counts = step_counts.astype(np.float)

# View the data type
print(step_counts.dtypes)
```

Output

```
>>> int64

>>> float64
```

Pandas Data Types and Imputation

Недействительные данные можно легко заполнить значениями (Invalid data points can be easily filled with values)

Code

```
# Create invalid data
step_counts[1:3] = np.NaN # NaN - Not a Value

# Now fill it in with zeros
step_counts = step_counts.fillna(0.)
# equivalently,
# step_counts.fillna(0., inplace=True)

print(step_counts[1:3])
```

Output

Pandas Data Types and Imputation

Недействительные данные можно легко заполнить значениями (Invalid data points can be easily filled with values)

Code

```
# Create invalid data
step_counts[1:3] = np.NaN

# Now fill it in with zeros
step_counts = step_counts.fillna(0.)
# equivalently,
# step_counts.fillna(0., inplace=True)

print(step_counts[1:3])
```

Output

```
>>> 2015-03-30 0.0
    2015-03-31 0.0
    Freq: D, Name: steps,
    dtype: float64
```

Pandas DataFrame Creation and Methods

Недействительные данные можно легко заполнить значениями (Invalid data points can be easily filled with values)

Code

Output

```
# Cycling distance
cycling_data = [10.7, 0, None, 2.4, 15.3,
                10.9, 0, None]

# Create a tuple of data
joined_data = list(zip(step_data,
                       cycling_data))

# The dataframe
activity_df = pd.DataFrame(joined_data)

print(activity_df)
```

Объект **DataFrame** представляет табличную структуру данных, состоящую из упорядоченной коллекции столбцов, причем типы значений (числовой, строковый, булев и т.д.) в разных столбцах могут различаться.

В объекте DataFrame хранятся два индекса: по строкам и по столбцам. Можно считать, что это словарь объектов **Series**.

Внутри объекта данные хранятся в виде одного или нескольких двумерных блоков, а не в виде списка, словаря или еще какой-нибудь коллекции одномерных массивов.

Pandas DataFrame Creation and Methods

DataFrames можно создавать из списков, словарей и Pandas Series (DataFrames can be created from lists, dictionaries, and Pandas Series)

Code

```
# Cycling distance
cycling_data = [10.7, 0, None, 2.4, 15.3,
                10.9, 0, None]

# Create a tuple of data
joined_data = list(zip(step_data,
                       cycling_data))

# The dataframe
activity_df = pd.DataFrame(joined_data)

print(activity_df)
```

Output

```
>>>
```

	0	1
0	3620	10.7
1	7891	0.0
2	9761	NaN
3	3907	2.4
4	4338	15.3
5	5373	10.9

Pandas DataFrame Creation and Methods

Можно добавить поименованные столбцы и индексы (Labeled columns and an index can be added)

Code

```
# Add column names to dataframe
activity_df = pd.DataFrame(
    joined_data,
    index=pd.date_range('20150329', periods=6),
    columns=['Walking','Cycling'])

print(activity_df)
```

Output

Полный список параметров:

pandas.DataFrame

class pandas.DataFrame(*data=None, index=None, columns=None, dtype=None, copy=False*)

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

Конструктор класса DataFrame выглядит так:

class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)

data – массив ndarray, словарь (dict) или другой DataFrame

index – список меток для записей (имена строк таблицы)

columns – список меток для полей (имена столбцов таблицы)

dtype – объект numpy.dtype, определяющий тип данных

copy – создает копию массива данных, если параметр равен True в ином случае ничего не делает

Pandas DataFrame Creation and Methods

Можно добавить поименованные столбцы и индексы (Labeled columns and an index can be added)

Code

```
# Add column names to dataframe
activity_df = pd.DataFrame(joined_data,
              index=pd.date_range('20150329',
              periods=6),
              columns=['Walking','Cycling'])

print(activity_df)
```

Output

```
>>>
```

	Walking	Cycling
2015-03-29	3620	10.7
2015-03-30	7891	0.0
2015-03-31	9761	NaN
2015-04-01	3907	2.4
2015-04-02	4338	15.3
2015-04-03	5373	10.9

Indexing DataFrame Rows

Строки DataFrame можно индексировать с помощью методов loc и iloc (DataFrame rows can be indexed by row using the 'loc' and 'iloc' methods)

Code

```
# Select row of data by index name
print(activity_df.loc['2015-04-01'])
```

Output

Indexing DataFrame Rows

Строки DataFrame можно индексировать с помощью методов loc и iloc (DataFrame rows can be indexed by row using the 'loc' and 'iloc' methods)

Code

```
# Select row of data by index name
print(activity_df.loc['2015-04-01'])
```

Output

```
>>> Walking 3907.0
    Cycling 2.4
    Name: 2015-04-01,
    dtype: float64
```

Indexing DataFrame Rows

Строки DataFrame можно индексировать с помощью методов loc и iloc (DataFrame rows can be indexed by row using the 'loc' and 'iloc' methods)

Code

```
# Select row of data by integer position
print(activity_df.iloc[-3])
```

Output

Indexing DataFrame Rows

Строки DataFrame можно индексировать с помощью методов loc и iloc (DataFrame rows can be indexed by row using the 'loc' and 'iloc' methods)

Code

```
# Select row of data by integer position
print(activity_df.iloc[-3])
```

Output

```
>>> Walking 3907.0
    Cycling 2.4
    Name: 2015-04-01,
    dtype: float64
```

Indexing DataFrame Columns

Столбцы DataFrame можно индексировать по имени (DataFrame columns can be indexed by name)

Code

```
# Name of column
print(activity_df['Walking'])
```

Output

```
>>> 2015-03-29 3620
    2015-03-30 7891
    2015-03-31 9761
    2015-04-01 3907
    2015-04-02 4338
    2015-04-03 5373
    Freq: D, Name: Walking,
    dtype: int64
```

Indexing DataFrame Columns

Столбцы DataFrame также можно индексировать как свойства (DataFrame columns can also be indexed as properties)

Code

```
# Object-oriented approach
print(activity_df.Walking)
```

Output

Indexing DataFrame Columns

Столбцы DataFrame также можно индексировать как свойства (DataFrame columns can also be indexed as properties)

Code

```
# Object-oriented approach
print(activity_df.Walking)
```

Output

```
>>> 2015-03-29 3620
    2015-03-30 7891
    2015-03-31 9761
    2015-04-01 3907
    2015-04-02 4338
    2015-04-03 5373
    Freq: D, Name: Walking,
    dtype: int64
```

Indexing DataFrame Columns

DataFrame columns can be indexed by integer (столбцы DataFrame можно индексировать целым числом)

Code

```
# First column
print(activity_df.iloc[:,0])
```

Output

Indexing DataFrame Columns

DataFrame columns can be indexed by integer (столбцы DataFrame можно индексировать целым числом)

Code

```
# First column
print(activity_df.iloc[:,0])
```

Output

```
>>> 2015-03-29 3620
    2015-03-30 7891
    2015-03-31 9761
    2015-04-01 3907
    2015-04-02 4338
    2015-04-03 5373
    Freq: D, Name: Walking,
    dtype: int64
```

Reading Data with Pandas

CSV и другие распространенные типы файлов можно прочитать с помощью одной команды (CSV and other common filetypes can be read with a single command)

Code

```
# The location of the data file
filepath = 'data/Iris_Data/Iris_Data.csv'

# Import the data
data = pd.read_csv(filepath)

# Print a few rows
print(data.iloc[:5])
```

Output

Reading Data with Pandas

CSV и другие распространенные типы файлов можно прочитать с помощью одной команды (CSV and other common filetypes can be read with a single command)

Code

```
# The location of the data file
filepath = 'data/Iris_Data/Iris_Data.csv'

# Import the data
data = pd.read_csv(filepath)

# Print a few rows
print(data.iloc[:5])
```

Output

>>>

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Assigning New Data to a DataFrame

Данные могут быть (повторно) присвоены столбцу DataFrame (Data can be (re-)assigned to a DataFrame column)

Code

```
# Create a new column that is a product
# of both measurements
data['sepal_area'] = data.sepal_length *
                     data.sepal_width

# Print a few rows and columns
print(data.iloc[:5, -3:])
```

Output

Assigning New Data to a DataFrame

Данные могут быть (повторно) присвоены столбцу DataFrame (Data can be (re-)assigned to a DataFrame column)

Code

```
# Create a new column that is a product
# of both measurements
data['sepal_area'] = data.sepal_length *
                     data.sepal_width

# Print a few rows and columns
print(data.iloc[:5, -3:])
```

Output

>>>

	petal_width	species	sepal_area
0	0.2	Iris-setosa	17.85
1	0.2	Iris-setosa	14.70
2	0.2	Iris-setosa	15.04
3	0.2	Iris-setosa	14.26
4	0.2	Iris-setosa	18.00

Applying a Function to a DataFrame Column

Функции могут применяться к столбцам или строкам DataFrame или Series (Functions can be applied to columns or rows of a DataFrame or Series)

Code

```
# The lambda function applies what
# follows it to each row of data
data['abbrev'] = (data
                  .species
                  .apply(lambda x:
                   x.replace('Iris-','')))

# Note that there are other ways to
# accomplish the above

print(data.iloc[:5, -3:])
```

Output

Applying a Function to a DataFrame Column

Функции могут применяться к столбцам или строкам DataFrame или Series (Functions can be applied to columns or rows of a DataFrame or Series)

Code

```
# The lambda function applies what
# follows it to each row of data
data['abbrev'] = (data
                  .species
                  .apply(lambda x:
                   x.replace('Iris-','')))

# Note that there are other ways to
# accomplish the above

print(data.iloc[:5, -3:])
```

Output

>>>

	petal_width	species	abbrev
0	0.2	Iris-setosa	setosa
1	0.2	Iris-setosa	setosa
2	0.2	Iris-setosa	setosa
3	0.2	Iris-setosa	setosa
4	0.2	Iris-setosa	setosa

Concatenating Two DataFrames

Два DataFrames могут быть объединены по любому измерению (Two DataFrames can be concatenated along either dimension)

Code

```
# Concatenate the first two and
# last two rows
small_data = pd.concat([data.iloc[:2],
                        data.iloc[-2:]])


print(small_data.iloc[:,-3:])

# See the 'join' method for
# SQL style joining of dataframes
```

Output

Concatenating Two DataFrames

Два DataFrames могут быть объединены по любому измерению (Two DataFrames can be concatenated along either dimension)

Code

```
# Concatenate the first two and
# last two rows
small_data = pd.concat([data.iloc[:2],
                        data.iloc[-2:]])


print(small_data.iloc[:,-3:])

# See the 'join' method for
# SQL style joining of dataframes
```

Output

```
>>>
```

	petal_length	petal_width	species
0	1.4	0.2	Iris-setosa
1	1.4	0.2	Iris-setosa
148	5.4	2.3	Iris-virginica
149	5.1	1.8	Iris-virginica

Aggregated Statistics with GroupBy

С помощью метода groupby вычисляется агрегированная статистика DataFrame (Using the groupby method calculated aggregated DataFrame statistics)

Code

Output

```
# Use the size method with a
# DataFrame to get count
# For a Series, use the .value_counts
# method
group_sizes = (data
               .groupby('species')
               .size())

print(group_sizes)
```

Aggregated Statistics with GroupBy

С помощью метода groupby вычисляется агрегированная статистика DataFrame (Using the groupby method calculated aggregated DataFrame statistics)

Code

```
# Use the size method with a
# DataFrame to get count
# For a Series, use the .value_counts
# method
group_sizes = (data
               .groupby('species')
               .size())

print(group_sizes)
```

Output

```
>>> species
    Iris-setosa        50
    Iris-versicolor    50
    Iris-virginica     50
    dtype: int64
```

Выполнение статистических расчетов (Performing Statistical Calculations)

Pandas содержит множество статистических методов - среднее значение, медиана и мода и др. (Pandas contains a variety of statistical methods - mean, median, and mode)

Code

```
# Mean calculated on a DataFrame
print(data.mean())
```

Output

Performing Statistical Calculations

Pandas содержит множество статистических методов - среднее значение, медиана и мода и др. (Pandas contains a variety of statistical methods - mean, median, and mode)

Code

```
# Mean calculated on a DataFrame
print(data.mean())
```

Output

```
>>> sepal_length 5.843333
    sepal_width 3.054000
    petal_length 3.758667
    petal_width 1.198667
    dtype: float64
```

Performing Statistical Calculations

Pandas содержит множество статистических методов - среднее значение, медиана и мода и др. (Pandas contains a variety of statistical methods - mean, median, and mode)

Code

```
# Mean calculated on a DataFrame
print(data.mean())



# Median calculated on a Series
print(data.petal_length.median())
```

Output

```
>>> sepal_length 5.843333
    sepal_width 3.054000
    petal_length 3.758667
    petal_width 1.198667
    dtype: float64

>>> 4.35
```

Performing Statistical Calculations

Pandas содержит множество статистических методов - среднее значение, медиана и мода и др. (Pandas contains a variety of statistical methods - mean, median, and mode)

Code

```
# Mean calculated on a DataFrame
print(data.mean())
```

```
# Median calculated on a Series
print(data.petal_length.median())
```

```
# Mode calculated on a Series
print(data.petal_length.mode())
```

Output

```
>>> sepal_length 5.843333
    sepal_width 3.054000
    petal_length 3.758667
    petal_width 1.198667
    dtype: float64
```

```
>>> 4.35
```

```
>>> 0 1.5
    dtype: float64
```

Performing Statistical Calculations

Также можно рассчитать стандартное отклонение, дисперсию, квантили и др. (Standard deviation, variance, SEM and quantiles can also be calculated)

Code

```
# Standard dev, variance, and SEM
print(data.petal_length.std(),
      data.petal_length.var(),
      data.petal_length.sem())
```

Output

Performing Statistical Calculations

Также можно рассчитать стандартное отклонение, дисперсию, квантили и др. (Standard deviation, variance, SEM and quantiles can also be calculated)

Code

```
# Standard dev, variance, and SEM
print(data.petal_length.std(),
      data.petal_length.var(),
      data.petal_length.sem())
```

Output

```
>>> 1.76442041995
    3.11317941834
    0.144064324021
```

Performing Statistical Calculations

Также можно рассчитать стандартное отклонение, дисперсию, квантили и др. (Standard deviation, variance, SEM and quantiles can also be calculated)

Code

```
# Standard dev, variance, and SEM
print(data.petal_length.std(),
      data.petal_length.var(),
      data.petal_length.sem())

# As well as quantiles
print(data.quantile(0))
```

Output

```
>>> 1.76442041995
    3.11317941834
    0.144064324021

>>> sepal_length 4.3
    sepal_width 2.0
    petal_length 1.0
    petal_width 0.1
    Name: 0, dtype: float64
```

Performing Statistical Calculations

Множественные вычисления могут быть представлены в DataFrame (Multiple calculations can be presented in a DataFrame)

Code

Output

```
print(data.describe())
```

Performing Statistical Calculations

Множественные вычисления могут быть представлены в DataFrame (Multiple calculations can be presented in a DataFrame)

Code

```
print(data.describe())
```

Output

```
>>>
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Выборка из DataFrames (Sampling from DataFrames)

Из DataFrames можно выбирать произвольно (DataFrames can be randomly sampled from)

Code

```
# Sample 5 rows without replacement
sample = (data
          .sample(n=5,
                  replace=False,
                  random_state=42))


print(sample.iloc[:,-3:])
```

Output

Sampling from DataFrames

Из DataFrames можно выбирать произвольно (DataFrames can be randomly sampled from)

Code

```
# Sample 5 rows without replacement
sample = (data
          .sample(n=5,
                  replace=False,
                  random_state=42))


print(sample.iloc[:,-3:])
```

Output

>>>

	petal_length	petal_width	species
73	4.7	1.2	Iris-versicolor
18	1.7	0.3	Iris-setosa
118	6.9	2.3	Iris-virginica
78	4.5	1.5	Iris-versicolor
76	4.8	1.4	Iris-versicolor

Sampling from DataFrames

Из DataFrames можно выбирать произвольно (DataFrames can be randomly sampled from)

Code

```
# Sample 5 rows without replacement
sample = (data
          .sample(n=5,
                  replace=False,
                  random_state=42))


print(sample.iloc[:,-3:])
```

Output

```
>>>
```

	petal_length	petal_width	species
73	4.7	1.2	Iris-versicolor
18	1.7	0.3	Iris-setosa
118	6.9	2.3	Iris-virginica
78	4.5	1.5	Iris-versicolor
76	4.8	1.4	Iris-versicolor

SciPy и NumPy также содержат множество статистических функций.

> Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

```
>>> x+2 #Sum of two variables
 7
>>> x-2 #Subtraction of two variables
 3
>>> x*2 #Multiplication of two variables
 10
>>> x**2 #Exponentiation of a variable
 25
>>> x%2 #Remainder of a variable
 1
>>> x/float(2) #Division of a variable
 2.5
```

Types and Type Conversion

str()

```
'5', '3.45', 'True' #Variables to strings
```

int()

```
5, 3, 1 #Variables to integers
```

float()

```
5.0, 1.0 #Variables to floats
```

bool()

```
True, True, True #Variables to booleans
```

> Libraries

Import Libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

> Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
 'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Indexing

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper() #String to uppercase
>>> my_string.lower() #String to lowercase
>>> my_string.count('w') #Count String elements
>>> my_string.replace('e', 'i') #Replace String elements
>>> my_string.strip() #Strip whitespaces
```

> NumPy Arrays

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Subset

```
>>> my_array[1] #Select item at index 1
 2
```

Slice

```
>>> my_array[0:2] #Select items at index 0 and 1
 array([1, 2])
```

Subset 2D Numpy arrays

```
>>> my_2darray[:,0] #my_2darray[rows, columns]
 array([1, 4])
```

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False, True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape #Get the dimensions of the array
>>> np.append(other_array) #Append items to an array
>>> np.insert(my_array, 1, 5) #Insert items in an array
>>> np.delete(my_array,[1]) #Delete items in an array
>>> np.mean(my_array) #Mean of the array
>>> np.median(my_array) #Median of the array
>>> my_array.corrcoef() #Correlation coefficient
>>> np.std(my_array) #Standard deviation
```

> Lists

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Subset

```
>>> my_list[1] #Select item at index 1
>>> my_list[-3] #Select 3rd last item
```

Slice

```
>>> my_list[1:3] #Select items at index 1 and 2
>>> my_list[1:] #Select items after index 0
>>> my_list[:3] #Select items before index 3
>>> my_list[:] #Copy my_list
```

Subset Lists of Lists

```
>>> my_list2[1][0] #my_list[list][itemOfList]
>>> my_list2[1][:2]
```

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a) #Get the index of an item
>>> my_list.count(a) #Count an item
>>> my_list.append('!') #Append an item at a time
>>> my_list.remove('!') #Remove an item
>>> del(my_list[0:1]) #Remove an item
>>> my_list.reverse() #Reverse the list
>>> my_list.extend('!') #Append an item
>>> my_list.pop(-1) #Remove an item
>>> my_list.insert(0,'!') #Insert an item
>>> my_list.sort() #Sort the list
```

> Pandas Data Structures

Series

A **one-dimensional** labeled array
capable of holding any data type

Index →

a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Dataframe

A **two-dimensional** labeled data structure
with columns of potentially different types

Columns →

Index →

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

> Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

> Asking For Help

```
>>> help(pd.Series.loc)
```

> Sort & Rank

```
>>> df.sort_index() #Sort by labels along an axis
>>> df.sort_values(by='Country') #Sort by the values along an axis
>>> df.rank() #Assign ranks to entries
```

> I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` **is a convenience wrapper around** `read_sql_table()` **and** `read_sql_query()`

```
>>> df.to_sql('myDf', engine)
```

> Selection

Getting

```
>>> s['b'] #Get one element
 -5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
 1 India New Delhi 1303171035
 2 Brazil Brasilia 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
 Country Brazil
 Capital Brasilia
 Population 207847528
>>> df.ix[:,'Capital'] #Select a single column of subset of columns
 0 Brussels
 1 New Delhi
 2 Brasilia
>>> df.ix[1,'Capital'] #Select rows and columns
 'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

> Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape #(rows,columns)
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> df.count() #Number of non-NA values
```

Summary

```
>>> df.sum() #Sum of values
>>> df.cumsum() #Cummulative sum of values
>>> df.min()/df.max() #Minimum/maximum values
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value
>>> df.describe() #Summary statistics
>>> df.mean() #Mean of values
>>> df.median() #Median of values
```

> Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f) #Apply function
>>> df.applymap(f) #Apply function element-wise
```

> Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
 a 10.0
 b NaN
 c 5.0
 d 7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_values=0)
 a 10.0
 b -5.0
 c 5.0
 d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```


ФИНАНСОВЫЙ
УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ