Анализ данных с использованием языка Python

А. В. Агафонов

2020

1. Введение

Одна из причин популярности реляционных баз данных и языка SQL (структурированного языка запросов) - простота соединения, фильтрации, преобразования и агрегирования данных.

Однако в том, что касается групповых операций, языки запросов типа SQL несколько ограничены.

Как мы увидим, выразительность и мощь языка Python и библиотеки Pandas позволяют выполнять гораздо более сложные групповые операции с помощью функций, принимающих произвольный объект Pandas или массив NumPy.

Своим успехом в качестве платформы для научных расчетов Python отчасти обязан простоте интеграции с кодом на C, C++ и FORTRAN. Во многих современных вычислительных средах применяется общий набор унаследованных библиотек, написанных на FORTRAN и C, содержащих реализации алгоритмов линейной алгебры, оптимизации, интегрирования, быстрого преобразования Фурье и других.

Поэтому многочисленные компании и национальные лаборатории используют Python как «клей» для объединения написанных за 30 лет программ.

Многие программы содержат небольшие участки кода, на выполнение которых уходит большая часть времени, и большие куски «склеивающего кода», который выполняется нечасто.

Во многих случаях время выполнения склеивающего кода несущественно, реальную отдачу дает оптимизация узких мест, которые иногда имеет смысл переписать на низкоуровневом языке типа C.

Данное пособие предполагает у обучающихся базовых знаний языка Python в объеме раздела The Python Tutorial его документации. Также хорошими книгами, с которых можно начать изучение языка, являются «Программирование на Python 3. Подробное руководство» Марка Саммерфилда и «Программирование на Python» Марка Лутца.

2. Требования к рабочей среде

Пособие предполагает использование следующего набора библиотек языка Python версии 3, предназначенных для обработки научных данных:

- Pandas;
- NumPy;
- Matplotlib;
- SciPy;
- IPython;

Основные компоненты Pandas базируются на следующих библиотеках: NumPy, реализующей быстродействующие операции с многомерными массивами, а также Matplotlib, обеспечивающей отображение различной графической информации.

Библиотека SciPy также базируется на NumPy и позволяет реализовывать с многомерными массивами ряд операций матричной арифметики, математической статистики и теории обработки сигналов. Применение данной библиотеки позволяет расширить базовый функционал, имеющийся в Pandas.

3. Создание и запуск рабочей среды

В Debian GNU/Linux необходимые пакеты устанавливаются командой:

```
apt install python3-pandas python3-scipy python3-qtconsole \
ipython3
```

Для запуска рабочей среды необходимо запустить в терминале:

```
ipython3 qtconsole --pylab=inline
```

Параметр `qtconsole` запускает рабочую среду в графическом режиме с использованием библиотеки Qt. Ввод параметра `--pylab` приводит к автоматической загрузке среды Pylab после запуска среды и позволяет, в частности, отображать графики непосредственно в интерпретаторе.

Затем необходимо загрузить библиотеку Pandas:

```
import pandas as pd
```

В данном листинге производится импорт двух основных классов библиотеки.

Класс `Series` является базовым для Pandas и предназначен для описания одномерной последовательности.

Класс `DataFrame` является аналогом таблицы базы данных. При его можно рассматривать как одномерную последовательность, элементами которой являются объекты класса `Series`.

4. Основные типы данных

4.1. Series

Series — основной класс библиотеки Pandas. Экземпляры данного класса схожи со списком (`list`) языка Python и представляют собой совокупность одномерного массива типа `array`, описанного в библиотеке NumPy и массива меток, называемого индексом.

Простейший объект Series может быть создан на основе списка:

```
In [1]: s = pd.Series([3, 14, 88, 12])

In [2]: s
Out[2]:
0     3
1    14
2    88
3    12
dtype: int64
```

При строковом представлении `Series`, отображаемом в интерактивном режиме, индекс находится слева, а соответствующие значения справа. Так как индекс не был задан явно, он был создан по-умолчанию и состоит из последовательных целых чисел.

Получить представление самого массива и его индекса можно с помощью атрибутов values и index соответственно:

```
In [3]: s.values
Out[3]: array([3, 14, 88, 12, 5])

In [4]: s.index
Out[4]: Int64Index([0, 1, 2, 3])
```

Часто желательно создать объект `Series` с индексом, идентифицирующим каждый элемент данных:

```
In [5]: s2 = pd.Series([3, 14, 88, 12],
   ...:          index=['a', 'b', 'c', 'd'])

In [6]: s2
Out[6]:
0     3
1    14
2    88
3    12
4     5
dtype: int64
```

```
In [7]: s2.index
Out[7]: Index(['a', 'b', 'c', 'd'], dtype = object)
```

Для выборки одного или нескольких элементов из объекта `Series` можно использовать значения индекса, что позволяет уподобить его упорядоченному словарю:

```
In [8]: s2['b']
Out[8]: 14
```

Возможно также передать объекту список ключей:

```
In [9]: s2[['b', 'd']]
Out[9]:
b    14
d    12
dtype: int64
```

К объекту `Series` применимы также операции над множествами. В частности, с помощью оператора `in` можно определить наличие заданного элемента в массиве:

```
In [10]: 14 in s2
Out[10]: True

In [11]: 42 in s2
Out[11]: False
```

Объект `Series` может быть создан на основе словаря (`dict`):

```
In [12]: d =  {'a': 35, 'b': 71, 'c': 16}

In [13]: s3 = pd.Series(d)

In [14]: s3
Out[14]:
a    35
b    71
c    16
dtype: int64
```

Для задания порядка элементов словаря можно явно задать последовательность значений индекса:

```
In [15]: i = ['c', 'a', 'd', 'b']

In [16]: s4 = pd.Series(d, index = i)
```

```
In [17]: s4
Out[17]:
c    16
a    35
d   NaN
b    71
dtype: float64
```

В данном случае три значения, найденные в `d`, помещены в соответствующие им позиции. Отсутствующее значение для индекса `'d'`, было заполнено числом с плавающей точкой `NaN`, (not a number), обозначающим отсутствующие данные, поэтому тип данных всего массива был автоматически изменен на `float64`.

Для выделения отсутствующих данных используются методы объекта `Series` `isnull()` и `notnull()`:

```
In [18]: s4.isnull()
Out[18]:
c    False
a    False
d     True
b    False
dtype: bool

In [19]: s4.notnull()
Out[19]:
c     True
a     True
d    False
b     True
dtype: bool
```

Массив без пустых записей формируется путем вызова метода `dropna()`:

```
In [20]: s4.dropna()
Out[20]:
c    16
a    35
b    71
dtype: float64
```

Объект `Series` сам по себе, а также его индекс имеют атрибут `name`, широко используемый другими компонентами библиотеки Pandas:

```
In [21]: s4.index.name = 'letter'
```

```
In [22]: s4.name = 'value'

In [23]: s4
Out[23]:
letter
c         16
a         35
d        NaN
b         71
Name: value, dtype: float64
```

Индекс может быть изменен с помощью операции присваивания:

```
In [24]: s4.index = ['x', 'y', 'z', 'w']
Out[24]:
x    16
y    35
z   NaN
w    71
Name: value, dtype: float64
```

Чрезвычайно важной операцией с объектом типа `Series`, унаследованной от типа `array` библиотеки NumPy, является извлечение значений с помощью булева фильтра.

Ниже приведен пример простейшего булева фильтра:

```
In [25]: s2 > 10
Out[25]:
a    False
b     True
c     True
d     True
dtype: bool
```

Для объекта `Series` большинство математических операторов переопределены таким образом, что возвращают массив, значения которого соответствуют результату применения соответствующего оператора к каждому из элементов исходного объекта.

Булевы фильтры могут быть скомбинировать для получения сложного критерия отбора элементов массива:

```
In [26]: s2[(s2 > 10) & (s2 < 20)]
Out[26]:
b    14
d    12
dtype: int64
```

Результатом выполнения данного листинга являются элементы исходного массива со значениями больше 10 и меньше 20.

При этом операции с массивом, например фильтрация с помощью булева массива, скалярное умножение или применение математических функций, сохраняют связь между индексом и значением.

4.2. DataFrame

Объект `DataFrame` представляет собой табличную структуру данных, содержащую упорядоченное множество столбцов, причем типы значений (числовой, строковый, булев и т. д.) в разных столбцах могут различаться.

Объект `DataFrame` содержит два индекса: по строкам и по столбцам.

Данный объект можно рассматривать как словарь объектов Series, однако внутри объекта данные хранятся в виде одного или нескольких двумерных блоков, а не в виде списка, словаря или какой-либо еще последовательности одномерных массивов.

Простейший способ создании объекта `DataFrame` — на основе словаря списков одинаковой длины или массивов NumPy:

```
In [1]: data = {
    ...:     'f1': [1, 3, 2],
    ...:     'f2': ['a', 'b', 'c'],
    ...:     'f3': [1.1, 1.5, 1.7]
    ...: }

In [2]: df = pd.DataFrame(data)
Out[2]: df
   f1 f2  f3
0   1  a 1.1
1   3  b 1.5
2   2  c 1.7
```

Выведенный в левом столбце индекс для получившегося `DataFrame` был построен автоматически.

Список столбцов может быть отображен с помощью атрибута `columns`:

```
In [3]: df.columns
Out[3]: Index(['f1', 'f2', 'f3'], dtype='object')
```

Типы данных, используемые для столбцов, могут быть отображены с помощью атрибута `dtypes`:

```
In [4]: df.dtypes
Out[4]:
f1      int64
f2     object
f3    float64
dtype: object
```

С использованием операции индексирования можно извлечь содержимое одного или нескольких столбцов. При этом в случае извлечения одного столбца будет возвращен объект типа `Series`, иначе — `DataFrame`:

```
In [5]: df['f1']
Out[5]:
0    1
1    3
2    2
Name: f1, dtype: int64

In [6]: type(df['f1'])
Out[6]: pandas.core.series.Series

In [7]: df[['f2', 'f3']]
Out[7]:
  f2   f3
0  a  1.1
1  b  1.5
2  c  1.7

In [8]: type(df[['f2', 'f3']])
Out[8]: pandas.core.frame.DataFrame
```

Объект `DataFrame` как и `Series` поддерживает фильтрацию с использованием булевых фильтров:

```
In [9]: df[(df['f1'] >= 2) & (df['f2'] >= 'b')]
Out[9]:
   f1 f2   f3
1   3  b  1.5
2   2  c  1.7
```

5. Визуализация данных

Примеры, рассмотренные ранее, хорошо иллюстрируют удобство средств отображения объектов в текстовом виде, как встроенных в язык Python, так и определенных в библиотеке Pandas.

С другой стороны, информация, представленная в виде диаграмм или графиков, оказывается удобнее для восприятия и анализа человеком. При использовании Pandas существует два основных способа графического вывода:

- с помощью встроенных методов объектов Pandas;
- с использованием явного вызова функций библиотеки Matplotlib.

5.1. Визуализация встроенными средствами объектов Pandas

Объекты Pandas обеспечивают простой и удобный интерфейс для представления их данных в виде различных графиков.

Обычный график выводится с помощью метода `plot` соответствующего объекта.

```
import numpy as np
s = pd.Series(np.random.randn(1000, 1))
s.plot()
```

Для изменения типа графика можно указать виде строки именованный параметр `kind`. Определены следующие типы:

- `bar` и `barh` — столбчатые диаграммы;
- `hist` — гистограммы;
- `box` — вероятностные распределения множества столбцов объекта `DataFrame`;
- `kde` и `density` — отображения функции плотности вероятности;
- `area` — закрашенных областей под графиком;
- `scatter` — точечной диаграммы;
- `hexbin` — плотности распределения точек по шестиугольным участкам;
- `pie` — круговых диаграмм.

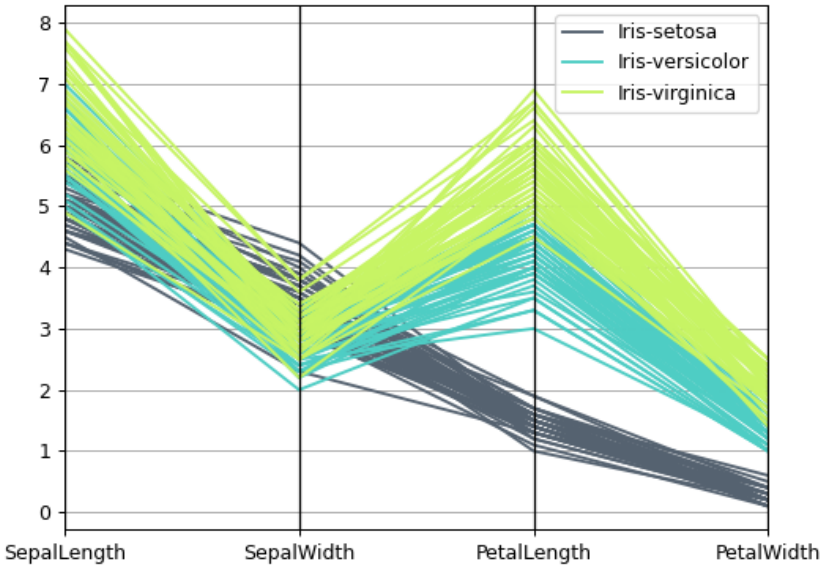
Однако в большинстве случаев удобнее оказывается обратиться к одноименным атрибутам метода `plot`:

```
In [3]: s.plot.pie()
```

Помимо базовых определены также дополнительные функции отображения, которые вынесены в отдельный дочерний модуль Pandas — `plotting`. Одной из наиболее полезных функций является график в параллельных координатах, позволяющий отобразить каждую точку пространства произвольной размерности в виде гистограммы и, таким образом, визуально оценивать их сходство.

```
In [89]: from pandas.plotting import parallel_coordinates
In [90]: data = pd.read_csv('iris.csv')
In [91]: plt.figure()
Out[91]: <matplotlib.figure.Figure at 0x13212da90>
In [92]: parallel_coordinates(data, 'Name')
Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x12da44da0>
```

Исходные коды данных функций в Debian GNU/Linux находятся файле `/usr/lib/python3/dist-packages/pandas/tools/plotting` и при необходимости могут быть скопированы и модифицированы.



5.2. Визуализация с помощью Matplotlib

Matplotlib расширяет возможности отображения информации, в сравнении с встроенными методами объектов Pandas, требуя, однако, значительно больше усилий для подготовки графиков.

6. Загрузка, переформатирование и очистка данных

Pandas поддерживает множество различных методов ввода и вывода данных, включая файлы различных типов, а также подключения к СУБД. Полное описание поддерживаемых типов данных приведено в документации библиотеки.

Для чтения используются функции семейства `read`, например `read_csv`, `read_json`, `read_hdf` и `read_sql`.

Запись осуществляется функциями семейства `to`, например `to_csv`, `to_json`, `to_hdf` и `to_sql`.

6.1. Загрузка и форматирование файлов в формате CSV

Рассмотрим, например, процедуру загрузки данных из файла `sample1.csv` в формате CSV с нестандартными разделителями (`;`) и кодировкой (`cp1251`).

Файл имеет следующее содержимое:

```
Объект;Широта;Долгота;Время
A;14.00000;65.00000;8:30 01.01.2020
Б;12.00000;60.00000;8:25 02.01.2020
Б;12.00100;60.05000;12:10 3.1.2020
Б;12.00100;60.05000;12:10 3.1.2020
A;;;12:20 03.01.2020
A;12;61;11:10 02.01.2020
A;12;61;11:10 02.01.2020
A;12;61;11:10 02.01.2020
;;;
```

Загрузим его в Pandas:

```
In [37]: df = pd.read_csv('sample1.csv')
```

Таким образом файл загрузился некорректно. Необходимо явно указать его кодировку и символ-разделитель.

```
In [37]: df = pd.read_csv(\
    ...:     'sample1.csv',
    ...:     encoding = 'cp1251',
    ...:     sep      = ';'
    ...: )
```

Файл загрузился, однако временные отметки не были распознаны, так как тип соответствующего поля `df` — `object`, то есть текстовая строка.

Если объем данных велик и загрузка из файла занимает большое время, то для решения данной проблемы можно переформатировать некорректно распознанные столбцы:

```
In [37]: df.Время = pd.to_datetime(df.Время, '%H:%M:%S %d.%m.%Y')
```

Еще одним вариантом решения является повторная загрузка с явным указанием списка столбцов и формата временных отметок:

```
In [37]: df = pd.read_csv(\
    ...:     'sample1.csv',
    ...:     encoding = 'cp1251',
    ...:     sep      = ';',
    ...:     parse_dates = ['Время'],
    ...:     date_parser = lambda s: pd.to_datetime(\
    ...:         s,
    ...:         '%H:%M:%S %d.%m.%Y'
    ...:     )
    ...: )
```

Аналогично следует решать проблемы с форматированием и других типов данных, прежде всего — текстовых строк, распознаваемых как числа. Для их явного форматирования при загрузке служит именованный параметр `dtype` функции `read_csv`.

6.2. Загрузка и форматирование данных из СУБД

СУБД могут оказаться чрезвычайно полезными при решении аналитических задач, так как в отличие от большинства специализированных информационно-аналитических систем позволяют хранить и обрабатывать данные объемом гораздо больше емкости оперативной памяти компьютера.

При этом функционал СУБД по анализу данных как правило оказывается более узок, поэтому типичный сценарий обработки большого объема данных предполагает два шага:

1) Извлечение ограниченного объема записей, необходимых для анализа (например, находящихся внутри заданного временного интервала).

2) Применение сложных аналитических методик к извлеченным данным с помощью информационно-аналитической системы.

Рассмотрим процесс извлечения данных на примере СУБД PostgreSQL, развернутой на данном компьютере.

Подключимся к ней:

```
psql -u postgres
```

Создадим таблицу `sample2` и наполним ее данными:

```
postgres> CREATE TABLE sample2 (obj varchar, lat float, lon float, t datetime);
postgres> INSERT INTO sample2 VALUES ('A', 14, 65, 2020-01-01 8:30);
postgres> INSERT INTO sample2 VALUES ('B', 12, 60, 2020-01-02 8:25);
postgres> CREATE USER user WITH password='password';
postgres> GRANT read TO user ON sample2;
postgres> \q
```

С помощью Pandas подключимся к СУБД и извлечем данные:

```
from sqlalchemy import create_engine
In [504]: engine = create_engine('postgresql://user:password@localhost:5432/postgres')
con = engine.connect()
data = pd.read_sql_table('sample2', con)
```

Библиотека `sqlalchemy` не является обязательной зависимостью для Pandas, и не устанавливается автоматически, поэтому в случае ошибки импорта следует развернуть ее вручную из репозитория дистрибутива.

6.3. Очистка данных

Очистка данных позволяет оставить в загруженной выборке лишь записи, представляющие ценность, и удалить ненужные.

Данные, извлеченные из каких-либо источников, могут содержать неполные записи, непригодные для дальнейшей обработки и дубликаты. Для их удаления используются ранее рассмотренные методы объектов `DataFrame`.

Также в выборке могут присутствовать различного рода выбросы. Способ их обработки зависит от цели, которая ставится перед аналитиком. Если ставится задача исследования усредненных значений, то выбросы обычно удаляются. В случае же решения задач криминалистики выбросы могут являться индикаторами искомых аномалий, и поэтому, наоборот, должны детально исследоваться.

Выделить в выборке крайние или средние значения позволяет совместное применение методов, возвращающих статистические характеристики выборки, и рассмотренных ранее критериев отбора объектов `Series` и `DataFrame`, например:

Также полезными могут оказаться регулярные выражения.

7. Агрегирование данных

После успешного выполнения загрузки, переформатирования и очистки данных может быть выполнен их анализ, важнейшим из инструментов которого являются операции агрегирования.

Специалистами в области обработки данных предлагается их реализация по схеме «разделение — применение — объединение».

В соответствии с данной схемой, на первом этапе данные разделяются на группы по одному или нескольким признакам, для чего используется механизм `groupby`.

Затем к каждой группе применяется заданная функция, которая порождает последовательность соответствующих им величин.

Заключительным этапом является объединение данных величин в результирующий объект, на основе которого может быть сформированы сводные таблицы для построения отчета или визуализации.

Ключи группировки могут задаваться по-разному и необязательно должны быть одного типа:

- список или массив значений той же длины, что ось, по которой производится группировка;
- значение, определяющее имя столбца объекта `DataFrame`;
- словарь или объект `Series`, определяющий соответствие между значениями на оси группировки и именами групп;
- функция, которой передается индекс оси или отдельные метки из этого индекса.

```
In [1]: arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
   ...:           ['Captive', 'Wild', 'Captive', 'Wild']]

In [2]: index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))

In [3]: df = pd.DataFrame({'Max Speed': [390., 350., 30., 20.]}, index=index)

In [4]: df
Out[4]:
                Max Speed
Animal Type
Falcon Captive      390.0
       Wild         350.0
Parrot Captive       30.0
       Wild          20.0

In [5]: df.groupby(level=0).mean()
Out[5]:
        Max Speed
Animal
Falcon      370.0
Parrot       25.0

In [6]: df.groupby(level='Type').mean()
Out[6]:
         Max Speed
Type
Captive      210.0
Wild         185.0
```

В данном примере показано использование сложного индекса и группировка по его полям. Также показано вычисление средних значений по сформированным группам. Для этого используется метод `mean` сгруппированных объектов, которые имеют тип `Series`. Аналогично могут использоваться такие методы, как например `min`, `max`, `sum`, `count` и т. п.

Простейшие операции агрегирования могут быть реализованы единственным вызовом метода объектов классов `Series` или `DataFrame`. Например, очень полезен метод `value_counts`:

```
In [1]: df = pd.DataFrame({'num_legs': [2, 4, 4, 6],
   ...:     'num_wings': [2, 0, 0, 0]},
   ...:     index=['falcon', 'dog', 'cat', 'ant']
   ...: )

In [2]:df
Out[2]:
```

```
        num_legs  num_wings
falcon         2          2
dog            4          0
cat            4          0
ant            6          0

In [3]:df.value_counts()
Out[3]:
num_legs  num_wings
4         0            2
6         0            1
2         2            1
dtype: int64
```

8. Корреляционный анализ

Корреляционный анализ занимается степенью связи между процессами изменения значений двух переменных, x и y.

Каждая из переменных представляется выборкой из n отсчетов.

Мы можем отметить точку, соответствующую паре соответствующих отсчетов каждой переменной, на двумерном графике рассеяния точек.

Обычно на графике переменную x располагают на горизонтальной оси, а — на вертикальной. Размещая точки для всех n отсчетов, получают график рассеяния точек, который говорит о соотношении между этими двумя переменными.

Соотношение и линейное, если прямая линия, проведенная через центральную часть скопления точек, дает наиболее подходящую аппроксимацию наблюдаемого соотношения.

Можно измерить, как близко находятся наблюдения к прямой линии, которая лучше всего описывает их линейное соотношение путем вычисления коэффициента корреляции Пирсона, обычно называемого просто коэффициентом корреляции.

Его истинная величина (генеральный коэффициент корреляции) ρ оценивается в выборке как r (выборочный коэффициент корреляции), которую обычно получают в результатах компьютерного расчета.

Пусть $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ - исследуемая выборка.

Выборочный коэффициент корреляции r определяется как

$$\frac{r = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{1}$$

где $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ — выборочные средние значения.

Свойства коэффициента корреляции r:

1) r изменяется в интервале от -1 до $+1$.

2) Знак r означает, увеличивается ли одна переменная по мере того, как увеличивается другая (положительный r), или уменьшается ли одна переменная по мере того, как увеличивается другая (отрицательный r).

3) Величина r указывает, как близко расположены точки к прямой линии. В частности, если $r = +1$ или $r = -1$, то имеется абсолютная (функциональная) корреляция по всем точкам, лежащим на линии (практически это маловероятно); если , то линейной корреляции нет (хотя может быть нелинейное соотношение). Чем ближе r к крайним точкам (± 1), тем больше степень линейной связи.

4) Коэффициент корреляции r безразмерен, то есть не имеет единиц измерения.

5) Величина r обоснована только в диапазоне значений x и y в выборке. Нельзя заключить, что он будет иметь ту же величину при рассмотрении значений x или y, которые значительно больше, чем их значения в выборке.

6) x и y могут взаимозаменяться, не влияя на величину r.

7) Корреляция между x и не обязательно означает соотношение причины и следствия.

Расчет r может ввести в заблуждение, если:

- соотношение между двумя переменными нелинейное, например квадратичное;
- данные включают более одного наблюдения по каждому случаю;
- есть аномальные значения (выбросы);
- данные содержат ярко выраженные подгруппы наблюдений.

При рассмотрении корреляционной связи подробно обсуждается понятие «сильной» (почти единичной) и «слабой» (почти нулевой) корреляции, но на практике ни та, ни другая никогда не встречаются. В результате остается неясным вопрос о разумной трактовке обычных для практики «промежуточных» значений коэффициента корреляции.

Для определения значимости используется статистический аппарат проверки гипотез, рассмотрение которого выходит за рамки пособия.

На практике обычно используются следующие эмпирические принципы определения значимости — это наличие выборки большого объема (порядка сотни отсчетов) и значения коэффициента корреляции $r >= 0.7$

В класс `DataFrame` встроен метод `corr`, попарно вычисляющий коэффициенты корреляции для всех имеющихся столбцов и возвращающий таблицу с соответствующими значениями:

```
In [1]: df = pd.DataFrame([(.2, .3), (.0, .6), (.6, .0), (.2, .1)],
   ...: columns=['a', 'b'])
In [2]: df.corr()
Out[2]:
      a     b
a   1.0   0.3
b   0.3   1.0
```

Аналогичный метод имеется у объектов класса `Series`. В качестве параметра он принимает другой объект, с которым производится сравнение.

9. Кластерный анализ

Кластерный анализ (кластеризация) — совокупность методов многомерного статистического анализа, включающая в себя набор различных алгоритмов классификации объектов.

Классификация состоит в разбиении заданной выборки объектов (пациентов, признаков и др.) на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.

Методы кластерного анализа можно разделить на две группы:

- иерархические;
- неиерархические.

Каждая из групп включает множество подходов и алгоритмов.

Используя различные методы кластерного анализа, аналитик может получить различные решения для одних и тех же данных. Это считается нормальным явлением.

Суть иерархической кластеризации состоит в последовательном объединении меньших кластеров в большие или разделении больших кластеров на меньшие.

Иерархические агломеративные методы (Agglomerative Nesting, AGNES) характеризуются последовательным объединением исходных элементов и соответствующим уменьшением числа кластеров.

В начале работы алгоритма все объекты являются отдельными кластерами. На первом шаге наиболее похожие объекты объединяются в кластер. На последующих шагах объединение продолжается до тех пор, пока все объекты не будут составлять один кластер.

Иерархические дивизимные методы (DIvisive ANAlysis, DIANA) являются логической противоположностью агломеративным методам. В начале работы алгоритма все объекты принадлежат одному кластеру, который на последующих шагах делится на меньшие кластеры, в результате образуется последовательность расщепляющих групп.

Иерархические методы кластеризации различаются правилами построения кластеров. В качестве правил выступают критерии, которые используются при решении вопроса о «схожести» объектов при их объединении в группу (агломеративные методы) либо разделения на группы (дивизимные методы).

Преимуществом иерархических методов кластеризации является их наглядность и удобство выбора оптимального количества кластеров при анализе выборки в интерактивном режиме.

Иерархические алгоритмы связаны с построением дендрограмм (от греческого dendron — «дерево»), которые являются результатом иерархического кластерного анализа. Дендрограмма описывает близость отдельных точек и кластеров друг к другу, представляет в графическом виде последовательность объединения (разделения) кластеров.

Дендрограмма (dendrogram) — древовидная диаграмма, содержащая n уровней, каждый из которых соответствует одному из шагов процесса последовательного укрупнения кластеров. Дендрограмму также называют древовидной схемой, деревом объединения кластеров, деревом иерархической структуры. Дендрограмма представляет собой вложенную группировку объектов, которая изменяется на различных уровнях иерархии. Существует много способов построения дендрограмм. В дендрограмме объекты могут располагаться вертикально или горизонтально.

Неиерархические методы представляют собой итеративные методы дробления исходной совокупности. В процессе деления новые кластеры формируются до тех пор, пока не будет выполнено правило остановки.

Неиерархическая кластеризация состоит в разделении набора данных на определенное количество отдельных кластеров. Существует два подхода. Первый заключается в определении границ кластеров как наиболее плотных участков в многомерном пространстве исходных данных, то есть определение кластера там, где имеется большое «сгущение точек». Второй подход заключается в минимизации меры различия объектов.

Наиболее распространен среди неиерархических методов алгоритм k-средних, также называемый быстрым кластерным анализом. Полное описание алгоритма можно найти в работе Хартигана и Вонга (Hartigan and Wong, 1978). В отличие от иерархических методов, которые не требуют предварительных предположений относительно числа кластеров, для возможности использования этого метода необходимо иметь гипотезу о наиболее вероятном количестве кластеров.

Алгоритм k-средних строит k кластеров, расположенных на возможно больших расстояниях друг от друга. Основной тип задач, которые решает алгоритм k-средних, - наличие предположений (гипотез) относительно числа кластеров, при этом они должны быть различны настолько, насколько это возможно. Выбор числа k может базироваться на результатах предшествующих исследований, теоретических соображениях или интуиции.

Существуют также неиерархические методы кластеризации, автоматически определяющие оптимальное количество кластеров, например DBSCAN и MeanShift, реализованные в библиотеке машинного обучения Scikit-learn.

Однако при их использовании необходимо учитывать ограничения, накладываемые на исследуемую выборку, так как ее особенности могут оказывать очень сильное влияние на получаемый результат.

В библиотеке Pandas отсутствуют встроенные средства кластерного анализа, однако ее объекту успешно могут использоваться в функциях, реализованных в Scipy и упомянутой Scikit-learn.

Подключим необходимые библиотеки:

```
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from scipy.cluster.vq import whiten
import numpy as np
```

Сгенерируем тестовую выборку из 2 кластеров: в одном 100 точек, в другом - 50:

```
np.random.seed(0)
a = np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]], size=[100,])
b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size=[50,])
df = pd.DataFrame({'a': a, 'b': b})
df.plot
```

Допустим, элементы столбцов имеют различную природу, например один из них описывает момент времени, а другой — значение какой-то переменной в метрах. При кластеризации принципиально важен механизм определения близости между точками, в качестве которого чаще всего выступает евклидово расстояние. В любом случае, пространство параметров, в котором производится кластеризация, считается изотропным, то есть координатные оси равнозначны и единицы измерения у них должны быть равны.

Каким образом сравнить секунды с метрами? Строго говоря — это невозможно. Но на практике прибегают к переходу от конкретных единиц измерения к отклонению точек от математического ожидания (то есть среднего значения), измеряемого в долях дисперсии (то есть разброса выборки) по соответствующим осям. Данная операция также называется иногда *отбеливанием* (whitening) данных.

Для этого в Scipy имеется функция `whiten`:

```
w = whiten(df)
```

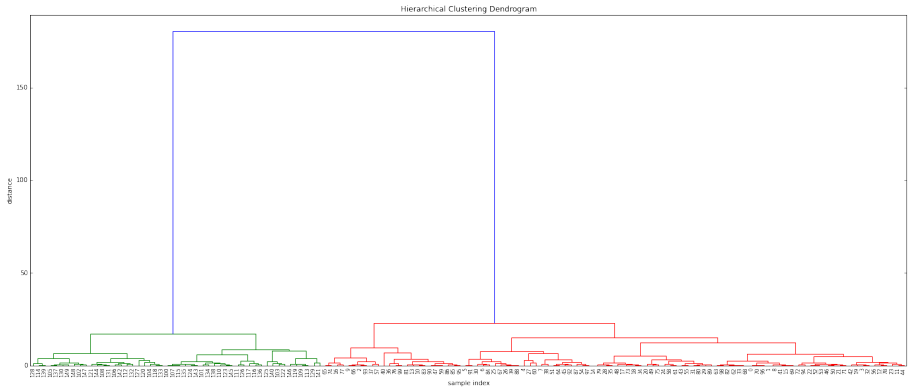
После выполнения отбеливания можно произвести кластеризацию.

```
Z = linkage(w, 'ward')
```

В данном случае используется мера расстояния `ward`, являющаяся оптимальной в большинстве случаев. Подробное описание ее, а также других метрик приведено в документации библиотеки. Полученный в результате объект `Z` — матрица связности точек.

Для определения оптимального количества кластеров построим дендрограмму:

```
plt.figure(figsize=(25, 10))
plt.title('Дендрограмма')
plt.xlabel('Индекс точки')
plt.ylabel('Расстояние')
dendrogram(
    Z,
    leaf_rotation=90,
    leaf_font_size=8,
)
plt.show()
```



По оси y снизу вверх в данном случае отображается расстояние между центрами кластеров, формируемых на каждом из шагов алгоритма. Соответственно, чем больше расстояние, тем дальше кластеры.

В интерактивном режиме оптимальное количество кластеров может быть определено визуально. В этом случае функции `fcluster`, отвечающей за отнесение точек к кластерам, можно явно указать их количество (`n`):

```
n = 2
c = fcluster(Z, n, criterion='maxclust')
```

По умолчанию используется критерий `inconsistent`, автоматически выделяющий кластер в случае, если расстояние от его центра до центра потенциального родительского кластера h_p сильно отличается от средних расстояний до центров его дочерних кластеров $\bar{h_c}$. Мерой отличия является коэффициент несоответствия i:

$$i = \frac{h_p - \bar{h_c}}{\sigma(h_c)}, \tag{2}$$

где $\sigma(h_c)$ — среднеквадратическое отклонение расстояний до дочерних кластеров, пропорциональное разбросу их значений.

В этом случае указывается коэффициент несоответствия:

```
i = 3
c = fcluster(Z, i)
```

Получаемый в результате массив по соответствующим индексам хранит номера кластеров, к которым были отнесены точки.

```
In[30]: c
Out[30]:
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

Далее можно добавить данный массив как столбец в изначальный объект `DataFrame`:

```
df['cluster'] = c
```

Данный столбец затем можно использовать при выводе графика на экран для определения цвета выводимой точки.

10. Сохранение и запуск программы

Для того, чтобы сохранить ранее введенные в интерпретаторе команды для дальнейшего использования, можно выполнить следующие шаги:

1) Вывести на терминал историю введенных команд с помощью команды `%history`.
2) Сохранить вывод предыдущей команды в файл `script.py`
3) Удалить из файла строки, содержащие ошибки.
4) Добавить в заголовок файла описание интерпретатора:

   ```
   #!/usr/bin/env python3
   ```

5) Ниже подключить необходимые библиотеки:

   ```
   import pandas as pd,...
   ```

6) Разрешить запуск скрипта:

   ```
   chmod +x script.py
   ```

7) Запустить программу:

   ```
   ./script.py
   ```

11. Лабораторные работы

11.1. Загрузка и запросы к данным в Pandas

Цель работы: изучить методы загрузки данных и запросов к ним с помощью Pandas.
Задания:

1) Загрузить данные из файла `billing.csv`.

2) Выделить все сеансы связи, которые осуществлял абонент 79022810325 в период с 10.09.13 до 10.10.13 (включительно).

3) Определить распределение сеансов связи по дням недели для абонента 70887390016.

4) Определить идентификаторы базовых станций, где были зафиксированы сеансы связи абонента 79217570540.

5) Определить распределение сеансов связи по часам в течении суток для абонента 79216666377.

11.2. Корреляционный анализ

Цель работы: изучить основы корреляционного анализа данных.
Задание:

1) Загрузить данные из файла `SacramentocrimeJanuary2006.csv`.

2) определить коэффициенты корреляции между количеством правонарушений, совершаемых за день, на протяжении месяца в северной, южной и центральной частями Сакраменто.

Замечание: необходимо определить крайние северную и южную точки выборки — их считать границами города. Город необходимо разбить с севера на юг на три равные зоны.

11.3. Кластерный анализ

Цель работы: изучить основы кластерного анализа данных.
Задания:

1) Выполнить кластерный анализ по полям широты, долготы и временной отметки преступлений в Сакраменто.

2) По каждому из кластеров произвести анализ распределений количества преступлений различных типов (построить гистограмму).

Замечание: для решения задания необходимо ознакомиться с классификацией преступлений полиции Сакраменто (файл `SPD Crime Class Codes.html`), синтаксисом регулярных выражений документации языка Python и с их помощью произвести замену описаний, введенных патрулями в свободной форме на единообразную форму обозначения, что далее позволит отфильтровать интересующие группы данных.