Анализ и визуализация данных

Основы работы с датафреймами `pandas`

Загрузка данных

Как мы уже убедились, библиотека `pandas` позволяет создавать датафреймы – удобные структуры для данных в табличном виде. Датафрейм можно получить из списка списков, из списка словарей, из словаря, простого или сложного. Но если наша задача – не преобразовать собранные данные (например, выгруженные с веб-страницы), а обработать уже готовые из имеющегося файла, библиотека `pandas` тоже будет чрезвычайно полезна.

Импортируем ее с сокращенным названием `pd`:

```
import pandas as pd
```

Библиотека `pandas` умеет загружать данные из файлов разных форматов. Среди них файлы Excel (расширения `.xls` и `.xlsx`), Stata (расширение `.dta`), SPSS (расширение `.sav`), текстовые файлы в формате TXT, CSV, JSON и другие. Мы поработаем с форматом CSV, одним из самых распространенных машиночитаемых форматов, название которого расшифровывается как *Comma Separated Values*, то есть «значения, разделенные запятыми». Такое название объясняется просто: внутри такого файла данные в табличном виде записаны так, что значения, относящиеся к разным столбцам, отграничены друг от друга запятыми. Например, таблица со столбцами A, B, C и числовыми значениями в них, в CSV будет выглядеть следующим образом:

Однако на практике можно столкнуться и с менее классическими вариантами этого формата. Например, если открыть файл CSV в Excel, он автоматически изменит разделитель `,` на `;`, чтобы точно не перепутать запятую как разделитель столбцов и запятую как десятичный разделить (между целой и дробной частью в числах). В таких случаях символ для разделителя нужно будет указать внутри аргумента `sep` в соответствующей функции, например, `sep = ";"`.

Перейдем к нашему файлу `Salaries.csv`. Если файл сохранен на компьютере, у нас два пути:

1. Поместить его в рабочую папку (рядом с ipynb-файлом, в котором пишем код для загрузки данных) через кнопку *Upload* на главной странице *Home* в Jupyter.
2. Найти файл на компьютере, кликнуть правой клавишей, зайти в свойства и скопировать полный путь к файлу из его расположения.

В первом случае все просто, вызываем функцию `read_csv()` и помещаем в нее название файла в виде строки:

```
dat = pd.read_csv("Salaries.csv")
```

Во втором случае нужно учесть важный момент: слэши в пути должны быть такие `/`, а не такие `\`. Слэш в обратную сторону внутри строки Python воспринимает как специальный символ, плюс, сочетание `\U`, которое неизбежно возникает при наличии папки `Users` будет намекать Python на кодировку символов в Unicode. На Mac и Linux таких слэшей в пути к файлу не возникает, а на Windows, чтобы не заменять каждый слэш вручную, можно перед кавычками поставить букву `r`:

```
dat = pd.read_csv(r"C:\Users\Boris Kondratenko\!2025\T.2\
Salaries.csv")
```

Примечание для Google Colab. Загрузить файл с данными в облачное хранилище можно через кнопку *Files* (значок папки слева от рабочей области с ячейками), при нажатии на которую появляется возможность выбрать файл с компьютера (значок стрелки). После добавления файла его можно выбрать, кликнуть на три точки справа от названия, скопировать путь через *Copy path* и вставить его в функцию `read_csv()`.

Итак, если никаких сообщений об ошибках не было (если `FileNotFoundError`, файл не найден, проверьте, нет ли опечаток в названии, и лежит ли файл в той папке, к которой мы прописали путь или в которой лежит текущий ipynb-файл), данные благополучно сохранились в виде датафрейма `dat`.

Примечание. Если вы планируете работать с файлами Excel, логика та же, только функция будет `read_excel()`. По умолчанию она считывает только первый лист файла, но в аргументе `sheet_name` можно указать индекс или название листа. Про другие, более продвинутые, опции вроде тех, что позволяют пропускать первые несколько строк в файле или, наоборот, загружать только определенный фрагмент таблицы, можно узнать, запросив `help(pd.read_excel)`.

Знакомство с данными

В этом файле сохранены данные по сотрудникам университета в США, а именно следующие их характеристики:

- `rank`: должность;
- `discipline`: тип преподаваемой дисциплины (`A` – теоретическая, `B` – практическая);
- `yrs.since.phd`: число лет с момента получения степени PhD;
- `yrs.service`: число лет опыта работы;
- `sex`: пол;
- `salary`: заработная плата за 9 месяцев, в долларах.

Приступим к изучению содержимого датафрейма.

Если датафрейм большой, а мы хотим быстро посмотреть на то, какого он вида, можем запросить на экран только первые строчки или последние:

```
# первые 5

dat.head()
```

```
   Unnamed: 0      rank discipline  yrs.since.phd  yrs.service   sex
salary
0           1      Prof          B             19           18  Male
139750
1           2      Prof          B             20           16  Male
173200
2           3  AsstProf          B              4            3  Male
79750
3           4      Prof          B             45           39  Male
115000
4           5      Prof          B             40           41  Male
141500
```

```
# последние 5

dat.tail()
```

```
     Unnamed: 0      rank discipline  yrs.since.phd  yrs.service   sex
salary
392         393      Prof          A             33           30  Male
103106
393         394      Prof          A             31           19  Male
150564
394         395      Prof          A             42           25  Male
101738
395         396      Prof          A             25           15  Male
95329
396         397  AsstProf          A              8            4  Male
81035
```

Если нам нужно другое число строк (не 5, которые показываются по умолчанию, а, скажем, 10), это легко исправить:

```
dat.head(10)
```

```
   Unnamed: 0       rank discipline  yrs.since.phd  yrs.service
sex  \
0           1       Prof          B             19           18
Male
1           2       Prof          B             20           16
Male
2           3   AsstProf          B              4            3
Male
3           4       Prof          B             45           39
Male
4           5       Prof          B             40           41
Male
5           6  AssocProf          B              6            6
Male
6           7       Prof          B             30           23
```

```
Male
7           8       Prof          B              45            45
Male
8           9       Prof          B              21            20
Male
9          10       Prof          B              18            18
Female

   salary
0  139750
1  173200
2   79750
3  115000
4  141500
5   97000
6  175000
7  147765
8  119250
9  129000
```

Выяснить размерность датафрейма можно с помощью атрибута `.shape`, в нем хранится кортеж с числом строк и числом столбцов:

```
dat.shape
```

```
(397, 7)
```

При желании можно извлечь только одно из чисел:

```
dat.shape[0]
```

```
397
```

```
dat.shape[1]
```

```
7
```

Примечание. Атрибут – некоторая фиксированная характеристика объекта (число строк или столбцов в датафрейме, названия строк или столбцов, тип данных), метод – функция, которая выполняет некоторую операцию над объектом. И атрибуты, и методы вызываются через точку, но в конце метода всегда есть круглые скобки (например, `.info()` ниже), они означают, что мы хотим применить метод, а не просто вызвать и посмотреть на него.

Техническое описание датафрейма можно получить, применив метод `.info()`:

```
dat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 7 columns):
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unnamed: 0      397 non-null    int64
 1   rank            397 non-null    object
 2   discipline      397 non-null    object
 3   yrs.since.phd   397 non-null    int64
 4   yrs.service     397 non-null    int64
 5   sex             397 non-null    object
 6   salary          397 non-null    int64
dtypes: int64(4), object(3)
memory usage: 21.8+ KB
```

Какую информацию выдал метод .info()? Во-первых, он сообщил нам, что `dat` является объектом `DataFrame`. Во-вторых, он вывел число строк (`RangeIndex: 397 entries`) и показал их индексы (`0 to 396`). В-третьих, он вывел число столбцов (`total 7 columns`). Наконец, он выдал информацию по каждому столбцу. Остановимся на этом поподробнее.

В выдаче выше представлено, сколько непустых элементов содержится в каждом столбце. Непустые элементы `non-null` – это все, кроме пропущенных значений, которые кодируются особым образом (`NaN` – от *Not A Number*). В нашей таблице все столбцы заполнены полностью. Далее указан тип каждого столбца, целочисленный `int64` и строковый `object`. Числа в конце означают объем памяти, который требуется для хранения, они зависят от битности системы.

Сводную статистическую информацию можно получить с помощью метода `.describe()`.

```
dat.describe()
```

```
       Unnamed: 0  yrs.since.phd  yrs.service         salary
count  397.000000     397.000000   397.000000     397.000000
mean   199.000000      22.314861    17.614610  113706.458438
std    114.748275      12.887003    13.006024   30289.038695
min      1.000000       1.000000     0.000000   57800.000000
25%    100.000000      12.000000     7.000000   91000.000000
50%    199.000000      21.000000    16.000000  107300.000000
75%    298.000000      32.000000    27.000000  134185.000000
max    397.000000      56.000000    60.000000  231545.000000
```

По умолчанию этот метод выбирает только числовые столбцы и выводит для них описательные статистики:

- `count` – число заполненных значений;
- `mean` – среднее арифметическое;
- `std` – стандартное отклонение (показатель разброса данных относительно среднего значения);
- `min` – минимальное значение;
- `max` – максимальное значение;
- `25%` – нижний квартиль (значение, которое 25% значений не превышают);
- `50%` – медиана (значение, которое 50% значений не превышают);

- `75%` – верхний квартиль (значение, которое 75% значений не превышают).

Если мы хотим описать только текстовые столбцы, нужно указать соответствующий тип внутри аргумента `include`:

```
dat.describe(include = "object")
```

```
        rank discipline   sex
count    397        397   397
unique     3          2     2
top     Prof          B  Male
freq     266        216   358
```

В таблице выше добавились новые строки `unique`, `top` и `freq`:

- `unique` – число уникальных значений в столбце (в `sex` их 2, `Male` и `Female`);
- `top` – мода, значение, которое встречается чаще всех (в `sex` больше значений `Male`, в выборке больше сотрудников-мужчин);
- `freq` – частота для значения в `top` (в `sex` 358, в выборке 358 сотрудников-мужчин).

Можно включить в описание все типы сразу, но это будет довольно громоздко (там, где характеристика неприменима, ставится пропуск `NaN`):

```
dat.describe(include = "all")
```

```
        Unnamed: 0  rank discipline  yrs.since.phd  yrs.service   sex
\
count   397.000000   397        397     397.000000   397.000000   397
unique         NaN     3          2            NaN          NaN     2
top            NaN  Prof          B            NaN          NaN  Male
freq           NaN   266        216            NaN          NaN   358
mean    199.000000   NaN        NaN      22.314861    17.614610   NaN
std     114.748275   NaN        NaN      12.887003    13.006024   NaN
min       1.000000   NaN        NaN       1.000000     0.000000   NaN
25%     100.000000   NaN        NaN      12.000000     7.000000   NaN
50%     199.000000   NaN        NaN      21.000000    16.000000   NaN
75%     298.000000   NaN        NaN      32.000000    27.000000   NaN
```

```
max     397.000000   NaN        NaN      56.000000    60.000000   NaN


                salary
count       397.000000
unique             NaN
top                NaN
freq               NaN
mean     113706.458438
std       30289.038695
min       57800.000000
25%       91000.000000
50%      107300.000000
75%      134185.000000
max      231545.000000
```

Если мы хотим запросить отдельно названия строк, нам пригодится атрибут `.index`:

```
dat.index
```

```
RangeIndex(start=0, stop=397, step=1)
```

А если столбцов – то атрибут `.columns`:

```
dat.columns
```

```
Index(['Unnamed: 0', 'rank', 'discipline', 'yrs.since.phd',
'yrs.service',
       'sex', 'salary'],
      dtype='object')
```

Переименование столбцов

Давайте для удобства дальнейшей работы переименуем некоторые столбцы с длинными названиями. Так, вместо `yrs.since.phd` назовем столбец `phd`, а вместо `yrs.service` поставим `service`. Для этого нам понадобится метод `.rename()`. По умолчанию он работает со строками, поэтому, если мы переименовываем столбцы, нужно явно это указать, записав название аргумента `columns`:

```
dat.rename(columns = {"yrs.since.phd" : "phd",
                      "yrs.service" : "service"})
```

```
     Unnamed: 0      rank discipline  phd  service   sex  salary
0             1      Prof          B   19       18  Male  139750
1             2      Prof          B   20       16  Male  173200
2             3  AsstProf          B    4        3  Male   79750
3             4      Prof          B   45       39  Male  115000
4             5      Prof          B   40       41  Male  141500
..          ...       ...        ...  ...      ...   ...     ...
```

```
392         393      Prof          A   33       30  Male  103106
393         394      Prof          A   31       19  Male  150564
394         395      Prof          A   42       25  Male  101738
395         396      Prof          A   25       15  Male   95329
396         397  AsstProf          A    8        4  Male   81035

[397 rows x 7 columns]
```

Соответствия старых и новых значений задаются в виде словаря, где ключами являются старые названия, а значениями – новые. Здесь стоит обратить внимание на один важный момент: метод `.rename()` работает осторожно, он не вносит изменения в исходный датафрейм, а возвращает его измененную копию. Если мы хотим сохранить изменения, нужно перезаписать датафрейм через `=` с тем же названием:

```
dat = dat.rename(columns = {"yrs.since.phd" : "phd",
                            "yrs.service" : "service"})
```

Или сделать то же самое, но более изящно, добавив аргумент `inplace = True` (то же, что и выше, но без `=`, ставим новые названия на место старых):

```
dat.rename(columns = {"yrs.since.phd" : "phd",
                            "yrs.service" : "service"}, inplace = True)
```

Выбор одного столбца

Теперь поработаем с отдельными столбцами. Для выбора столбца достаточно указать его название в квадратных скобках (и обязательно в кавычках, так как название является строкой):

```
dat["salary"]

0      139750
1      173200
2       79750
3      115000
4      141500
        ...
392    103106
393    150564
394    101738
395     95329
396     81035
Name: salary, Length: 397, dtype: int64
```

Еще столбец можно выбрать, не используя квадратные скобки, а просто указав его название через точку:

```
dat.salary
```

```
0      139750
1      173200
2       79750
3      115000
4      141500
        ...
392    103106
393    150564
394    101738
395     95329
396     81035
Name: salary, Length: 397, dtype: int64
```

Однако такой способ не универсален. В случае, если в названии столбца используются недопустимые для переменных символы (пробелы, тире, кириллические буквы), этот метод не подойдет. Для создания нового столбца этот способ тоже не сработает.

Теперь опишем выбранный столбец. Для числового столбца вполне логично использовать метод `.describe()`, он работает и для отдельных столбцов тоже:

```
dat["salary"].describe()
```

```
count       397.000000
mean     113706.458438
std       30289.038695
min       57800.000000
25%       91000.000000
50%      107300.000000
75%      134185.000000
max      231545.000000
Name: salary, dtype: float64
```

Все характеристики, которые выдает этот метод, можно запросить по-отдельности:

```
# число непустых значений

dat["salary"].count()
```

```
397
```

```
# среднее

dat["salary"].mean()
```

```
113706.45843828715
```

```
# медиана — квантиль уровня 0.5

dat["salary"].quantile(0.5)
```

```
107300.0
```

```
# нижний квартиль — квантиль уровня 0.25

dat["salary"].quantile(0.25)
```

```
91000.0
```

Текстовые столбцы тоже можно описывать, только вряд ли будет уместно будет считать для них среднее, медиану или что-то подобное. Поэтому мы посмотрим на уникальные значения и их частоты:

```
dat["sex"].unique()
```

```
array(['Male', 'Female'], dtype=object)
```

```
dat["sex"].value_counts()
```

```
sex
Male      358
Female     39
Name: count, dtype: int64
```

Выбор нескольких столбцов (и строк)

Если нам нужно сразу несколько столбцов (маленький датафрейм на основе старого), то названия столбцов необходимо оформить в виде списка и указать его в квадратных скобках:

```
small = dat[["sex", "rank", "salary"]]
small
```

```
      sex      rank  salary
0    Male      Prof  139750
1    Male      Prof  173200
2    Male  AsstProf   79750
3    Male      Prof  115000
4    Male      Prof  141500
..    ...       ...     ...
392  Male      Prof  103106
393  Male      Prof  150564
394  Male      Prof  101738
395  Male      Prof   95329
396  Male  AsstProf   81035

[397 rows x 3 columns]
```

Если нам нужно несколько столбцов подряд, начиная с одного названия и заканчивая другим, можно воспользоваться методом `.loc`:

```
dat.loc[:, "rank" : "service"]
```

```
          rank discipline  phd  service
0         Prof          B   19       18
1         Prof          B   20       16
2    AsstProf          B    4        3
3         Prof          B   45       39
4         Prof          B   40       41
..         ...        ...  ...      ...
392       Prof          A   33       30
393       Prof          A   31       19
394       Prof          A   42       25
395       Prof          A   25       15
396  AsstProf          A    8        4

[397 rows x 4 columns]
```

Метод `.loc` используется для выбора определенных строк и столбцов, поэтому в квадратных скобках образуется запись через запятую: на первом месте условия для строк, на втором – для столбцов. Здесь нас интересуют все строки (полный срез через `:`) и конкретные столбцы, с `rank` по `service` включительно.

Если бы мы хотели выбрать строки с 0 по 10 и столбцы с `rank` по `service`, тоже бы пригодился метод `.loc`:

```
dat.loc[0:10, "rank" : "service"]
```

```
          rank discipline  phd  service
0         Prof          B   19       18
1         Prof          B   20       16
2    AsstProf          B    4        3
3         Prof          B   45       39
4         Prof          B   40       41
5   AssocProf          B    6        6
6         Prof          B   30       23
7         Prof          B   45       45
8         Prof          B   21       20
9         Prof          B   18       18
10  AssocProf          B   12        8
```

Внимание: хотя в `.loc` мы вроде как задействуем обычные питоновские срезы, внутри этого метода срезы включают как левый, так и правый конец. Так, в примере выше были выбраны строки по 10-ую включительно и столбец `service` также был включен.

Иногда может возникнуть необходимость выбрать столбец по его порядковому номеру. Например, когда названий столбцов нет как таковых или когда названия слишком длинные, а переименовывать их нежелательно. Сделать это можно с помощью метода `.iloc` (`i` – от *index*). Выберем строки с 0 по 9 и столбцы с 0 по 3:

```
dat.iloc[0:10, 0:4]
```

```
   Unnamed: 0       rank discipline  phd
0           1       Prof          B   19
1           2       Prof          B   20
2           3   AsstProf          B    4
3           4       Prof          B   45
4           5       Prof          B   40
5           6  AssocProf          B    6
6           7       Prof          B   30
7           8       Prof          B   45
8           9       Prof          B   21
9          10       Prof          B   18
```

Внимание: в методе `.iloc`, поскольку работа идет с обычными числовыми индексами (как в списках и кортежах), правый конец среза исключается. Поэтому в примере выше 10-я строка и 4-ый столбец показаны не были.

Если в `.iloc` вписать только одно число, по умолчанию будет выдана строка с таким номером:

```
dat.iloc[2]
```

```
Unnamed: 0           3
rank          AsstProf
discipline           B
phd                  4
service              3
sex               Male
salary           79750
Name: 2, dtype: object
```

Это будет объект типа `pandas Series`, своего рода срез датафрейма:

```
type(dat.iloc[2])
```

```
pandas.core.series.Series
```

Фильтрация строк

Часто при работе с датафреймом нас не интересует выбор отдельных строк по названию или номеру, а интересует фильтрация наблюдений – выбор строк датафрейма, которые удовлетворяют определенному условию. Для этого интересующее нас условие необходимо указать в квадратных скобках. Например, выберем только те строки, которые соответствуют сотрудникам с опытом работы более 10 лет:

```
dat[dat["service"] > 10]
```

```
     Unnamed: 0  rank discipline  phd  service   sex  salary
0             1  Prof          B   19       18  Male  139750
1             2  Prof          B   20       16  Male  173200
3             4  Prof          B   45       39  Male  115000
4             5  Prof          B   40       41  Male  141500
6             7  Prof          B   30       23  Male  175000
..          ...   ...        ...  ...      ...   ...     ...
391         392  Prof          A   30       19  Male  151292
392         393  Prof          A   33       30  Male  103106
393         394  Prof          A   31       19  Male  150564
394         395  Prof          A   42       25  Male  101738
395         396  Prof          A   25       15  Male   95329

[242 rows x 7 columns]
```

Почему нельзя было написать проще, то есть `dat["service"] > 10`? Давайте напишем, и посмотрим, что получится:

```
dat["service"] > 10
```

```
0       True
1       True
2      False
3       True
4       True
       ...
392     True
393     True
394     True
395     True
396    False
Name: service, Length: 397, dtype: bool
```

Что мы увидели? Просто результат проверки условия для каждой строки датафрейма, набор из `True` и `False`. Когда мы подставляем это выражение в квадратные скобки, Python выбирает из `dat` те строки, где выражение принимает значение `True`.

Все символьные операторы для объединения условий работают как обычно, только тут важно не забывать про круглые скобки вокруг каждого условия. Например, выберем сотрудников женского пола со стажем более 10 лет:

```
# одновременное выполнение условий

dat[(dat["service"] > 10) & (dat["sex"] == "Female")]
```

```
     Unnamed: 0       rank discipline  phd  service     sex  salary
9            10       Prof          B   18       18  Female  129000
19           20       Prof          A   39       36  Female  137000
47           48       Prof          B   23       19  Female  151768
```

```
48            49       Prof          B   25       25  Female  140096
63            64  AssocProf          B   11       11  Female  103613
68            69       Prof          B   17       17  Female  111512
84            85       Prof          B   17       18  Female  122960
103          104       Prof          B   20       14  Female  127512
123          124  AssocProf          A   25       22  Female   62884
148          149       Prof          B   36       26  Female  144651
230          231       Prof          A   29       27  Female   91000
231          232  AssocProf          A   26       24  Female   73300
233          234       Prof          A   36       19  Female  117555
245          246       Prof          A   17       11  Female   90450
323          324       Prof          B   24       15  Female  161101
341          342       Prof          B   17       17  Female  124312
358          359       Prof          A   28       14  Female  109954
361          362       Prof          A   23       15  Female  109646
```

А теперь выберем профессоров или доцентов:

```
# или одно верно, или другое, или оба

dat[(dat["rank"] == "Prof") | (dat["rank"] == "AssocProf")]
```

```
     Unnamed: 0       rank discipline  phd  service   sex  salary
0             1       Prof          B   19       18  Male  139750
1             2       Prof          B   20       16  Male  173200
3             4       Prof          B   45       39  Male  115000
4             5       Prof          B   40       41  Male  141500
5             6  AssocProf          B    6        6  Male   97000
..          ...        ...        ...  ...      ...   ...     ...
391         392       Prof          A   30       19  Male  151292
392         393       Prof          A   33       30  Male  103106
393         394       Prof          A   31       19  Male  150564
394         395       Prof          A   42       25  Male  101738
395         396       Prof          A   25       15  Male   95329

[330 rows x 7 columns]
```

Если бы разных значений с подстрокой `"Prof"` в `rank` было много, было бы неудобно прописывать через `|` однотипные условия для каждой должности. Тогда логично было бы воспользоваться методом, который позволяет выбрать все строки, где в ячейке с текстом встречается слово "Prof". Такой метод есть – это метод на строках `.contains()`, который возвращает `True`, если некоторая подстрока входит в строку, и `False` – в противном случае.

```
# в нашем случае это все строки

dat[dat["rank"].str.contains("Prof")]
```

```
     Unnamed: 0      rank discipline  phd  service   sex  salary
0             1      Prof          B   19       18  Male  139750
1             2      Prof          B   20       16  Male  173200
2             3  AsstProf          B    4        3  Male   79750
3             4      Prof          B   45       39  Male  115000
4             5      Prof          B   40       41  Male  141500
..          ...       ...        ...  ...      ...   ...     ...
392         393      Prof          A   33       30  Male  103106
393         394      Prof          A   31       19  Male  150564
394         395      Prof          A   42       25  Male  101738
395         396      Prof          A   25       15  Male   95329
396         397  AsstProf          A    8        4  Male   81035

[397 rows x 7 columns]
```

А если наоборот, нам нужно отрицание – все строки, которые относятся к чему угодно, только не к `"Prof"`? Можно воспользоваться оператором `~` для отрицания и поставить его перед всем условием в скобках:

```
# таких нет, у всех Prof встречается в названии должности
# т.е. вернётся пустой датафрейм

dat[~dat['rank'].str.contains('Prof')]
```

```
Empty DataFrame
Columns: [Unnamed: 0, rank, discipline, phd, service, sex, salary]
Index: []
```

Добавление новых столбцов

Так как отдельный столбец датафрейма является объектом типа *pandas Series*, который наследует свойства массива, выполнять операции над столбцами довольно просто. Например, мы хотим добавить в `dat` столбец с заработной платой в тысячах. Для этого достаточно выбрать столбец `salary` и поделить все его значения на 1000:

```
dat['salary'] / 1000
```

```
0      139.750
1      173.200
2       79.750
3      115.000
4      141.500
        ...
392    103.106
393    150.564
394    101.738
395     95.329
396     81.035
Name: salary, Length: 397, dtype: float64
```

Теперь запишем полученный результат в новый столбец `salary_th` датафрейма `dat`:

```
dat['salary_th'] = dat['salary'] / 1000
```

Точно так же можно выполнять поэлементные операции с несколькими столбцами. Например, посчитать разность двух столбцов:

```
dat['service'] - dat['phd']
```

```
0      -1
1      -4
2      -1
3      -6
4       1
       ..
392    -3
393   -12
394   -17
395   -10
396    -4
Length: 397, dtype: int64
```

По умолчанию новые столбцы записываются в конец датафрейма, но при желании столбцы можно упорядочить по своему желанию.

Пример: в некотором датафрейме `df` есть столбцы `a`, `b`, `c`, мы хотим поменять их местами так, чтобы сначала был `c`, потом `a`, а потом `b`:

```
cols = ['c', 'a', 'b']
df = df[cols]
```

Теперь рассмотрим случай посложнее. Допустим, мы хотим добавить новый столбец `female`, который будет содержать значения `True` (респондент женского пола) и `False` (респондент мужского пола). Для этого достаточно написать маленькую функцию и применить ее сразу ко всем ячейкам в столбце `sex` с помощью метода `.apply()`. Раз функция маленькая и явно «разовая», ее можно не определять через `def`, а создать как анонимную lambda-функцию:

```
dat['female'] = dat['sex'].apply(lambda x: x == "Female")
dat.head()
```

```
   Unnamed: 0      rank discipline  phd  service   sex  salary
salary_th  \
0           1      Prof          B   19       18  Male  139750
139.75
1           2      Prof          B   20       16  Male  173200
173.20
2           3  AsstProf          B    4        3  Male   79750
79.75
```

```
3           4      Prof           B   45       39  Male  115000
115.00
4           5      Prof           B   40       41  Male  141500
141.50

   female
0   False
1   False
2   False
3   False
4   False
```

Функция `lambda x: x == "Female"` принимает на вход какой-то `x`, а возвращает `True` или `False`. Метод `.apply()` применяет эту функцию ко всем ячейкам выбранного столбца (аналогично растяжению какой-то ячейки с функцией в Excel или Google Sheets). То есть функцию мы пишем как будто бы для одной ячейки, а затем без всяких циклов применяем ее много раз.

Если мы захотим привести этот столбец к более классическому виду, с 0 и 1, можно преобразовать значения в целочисленный формат:

```
# int() превратит True в 1, а False — в 0

dat['female'] = dat['sex'].apply(lambda x: int(x == "Female"))
dat.head()
```

```
   Unnamed: 0      rank discipline  phd  service   sex  salary
salary_th  \
0           1      Prof           B   19       18  Male  139750
139.75
1           2      Prof           B   20       16  Male  173200
173.20
2           3  AsstProf           B    4        3  Male   79750
79.75
3           4      Prof           B   45       39  Male  115000
115.00
4           5      Prof           B   40       41  Male  141500
141.50

   female
0       0
1       0
2       0
3       0
4       0
```

Если функция, которую мы хотим применить, подразумевает большее количество операций, разумнее будет сделать выбор в пользу обычных неанонимных функций, задаваемых через `def`. Напишем функцию `new_rank()`, которая принимает на вход

значение должности, а возвращает его числовой эквивалент (1 для `Prof`, 2 для `AssocProf`, 3 для `AsstProf`):

```
def new_rank(x):
    if x == "Prof":
        y = 1
    elif x == "AssocProf":
        y = 2
    elif x == "AsstProf":
        y = 3
    else:
        y = None
    return y
```

Почему в функции выше мы добавили ветку с `else`, которая присваивает `y` значение `None`? Для универсальности, на случай, если в столбце встречаются пропущенные значения. В нашем случае это излишне, все ячейки в столбце `rank` заполнены, но в общем случае, если функция столкнется с неучтенным в ней значением, метод `.apply()` не сможет ее применить ко всем ячейкам и вызовет ошибку. А так все неизвестные функции значения будут заменены на пустый ячейки `None` (то же, что `NaN`).

```
dat['rank_num'] = dat['rank'].apply(new_rank)
dat.head()
```

```
   Unnamed: 0      rank discipline  phd  service   sex  salary
salary_th  \
0           1      Prof          B   19       18  Male  139750
139.75
1           2      Prof          B   20       16  Male  173200
173.20
2           3  AsstProf          B    4        3  Male   79750
79.75
3           4      Prof          B   45       39  Male  115000
115.00
4           5      Prof          B   40       41  Male  141500
141.50

   female  rank_num
0       0         1
1       0         1
2       0         3
3       0         1
4       0         1
```

Группировка и агрегирование

Для начала сгруппируем сотрудников по полу (`sex`). Группировка осуществляется с помощью метода `.groupby()`. Далее, на этот метод можно «наслоить» любую функцию

или функции для агрегирования. Вычислим средние значения по всем столбцам, отдельно для женщин, отдельно для мужчин:

```
# обратите внимание на параметр, благодаря которому обрабатываются
# только числовые значения - без него вернётся ошибка
dat.groupby(['sex']).mean(numeric_only=True)
```

```
        Unnamed: 0        phd    service         salary   salary_th
female  \
sex

Female  171.000000  16.512821  11.564103  101002.410256  101.002410
1.0
Male    202.050279  22.946927  18.273743  115090.418994  115.090419
0.0

        rank_num
sex
Female  1.820513
Male    1.463687
```

То, что вернул метод, является обычным датафреймом, поэтому, если мы хотим получить только среднее по определенному столбцу, мы можем выбрать его по названию:

```
# средняя заработная плата женщин и мужчин

dat.groupby(['sex'])['salary'].mean()
```

```
sex
Female    101002.410256
Male      115090.418994
Name: salary, dtype: float64
```

Внутри специальной функции `.agg()` можно указать сразу несколько функций, оформив их в виде списка:

```
# среднее и медиана

dat.groupby('sex')['salary'].agg(['mean', 'median'])
```

```
                mean    median
sex
Female  101002.410256  103750.0
Male    115090.418994  108043.0
```

Внутри `.agg()` можно также использовать свою функцию. Напишем, например, функцию `my_range()` для вычисления размаха – меры разброса данных, которая представляет собой разницу между максимальным значением и минимальным:

```
def delta_range(x):
    return max(x) - min(x)
```

И применим ее внутри `.agg()` (обратите внимание, тут название уже без кавычек, Python по умолчанию его не знает):

```
dat.groupby(['sex'])['salary'].agg(['min', 'max', delta_range])
```

```
          min     max  delta_range
sex
Female  62884  161101        98217
Male    57800  231545       173745
```

В заключение давайте более внимательно посмотрим на объект, который получается в процессе группировки через `.groupby()` и поймем, как его еще можно использовать.

```
dat.groupby('rank')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x000002021FA455B0>
```

Результат группировки от нас скрыт, он хранится в объекте особого типа `DataFrameGroupBy`. Чтобы посмотреть, что внутри, воспользуемся циклом:

```
for g in dat.groupby('rank'):
    print(g)
```

```
('AssocProf',      Unnamed: 0       rank discipline  phd  service
sex  salary  \
5             6  AssocProf          B    6        6    Male   97000
10           11  AssocProf          B   12        8    Male  119800
24           25  AssocProf          A   13        8  Female   74830
39           40  AssocProf          B    9        9    Male  100938
41           42  AssocProf          B   23       23    Male   93418
..          ...        ...        ...  ...      ...     ...     ...
363         364  AssocProf          A   20       17    Male   81285
367         368  AssocProf          A   10        1    Male  108413
370         371  AssocProf          A   13        8    Male   78182
379         380  AssocProf          A   11        8    Male  104121
382         383  AssocProf          A    8        5    Male   86895

     salary_th  female  rank_num
5       97.000       0         2
10     119.800       0         2
24      74.830       1         2
39     100.938       0         2
41      93.418       0         2
..         ...     ...       ...
363     81.285       0         2
```

```
367    108.413       0         2
370     78.182       0         2
379    104.121       0         2
382     86.895       0         2

[64 rows x 10 columns])
('AsstProf',      Unnamed: 0      rank discipline  phd  service   sex
salary  salary_th  \
2             3  AsstProf          B    4        3  Male   79750
79.750
11           12  AsstProf          B    7        2  Male   79800
79.800
12           13  AsstProf          B    1        1  Male   77700
77.700
13           14  AsstProf          B    2        0  Male   78000
78.000
27           28  AsstProf          B    5        3  Male   82379
82.379
..          ...       ...        ...  ...      ...   ...     ...
...
359         360  AsstProf          A   11        4  Male   78785
78.785
376         377  AsstProf          A    4        1  Male   74856
74.856
377         378  AsstProf          A    6        3  Male   77081
77.081
380         381  AsstProf          A    8        3  Male   75996
75.996
396         397  AsstProf          A    8        4  Male   81035
81.035

     female  rank_num
2         0         3
11        0         3
12        0         3
13        0         3
27        0         3
..      ...       ...
359       0         3
376       0         3
377       0         3
380       0         3
396       0         3

[67 rows x 10 columns])
('Prof',      Unnamed: 0  rank discipline  phd  service   sex  salary
salary_th  \
0             1  Prof          B   19       18  Male  139750
139.750
1             2  Prof          B   20       16  Male  173200
```

```
173.200
3             4  Prof          B   45       39  Male  115000
115.000
4             5  Prof          B   40       41  Male  141500
141.500
6             7  Prof          B   30       23  Male  175000
175.000
..          ...   ...        ...  ...      ...   ...     ...         ..
.
391         392  Prof          A   30       19  Male  151292
151.292
392         393  Prof          A   33       30  Male  103106
103.106
393         394  Prof          A   31       19  Male  150564
150.564
394         395  Prof          A   42       25  Male  101738
101.738
395         396  Prof          A   25       15  Male   95329
95.329

     female  rank_num
0         0         1
1         0         1
3         0         1
4         0         1
6         0         1
..      ...       ...
391       0         1
392       0         1
393       0         1
394       0         1
395       0         1

[266 rows x 10 columns])
```

Цикл выше выдает нам кортежи, в которых заключены пары значений: название группы и маленький датафрейм со строками, соответствующими этой группе. Для чего это можно использовать? Например, для сохранения данных по каждой группе в отдельный файл. Сделаем перебор в цикле сразу по элементам внутри пары (вспомните словари и перебор по `.items()`):

```
# на первом месте название, на втором – датафрейм

for group, tab in dat.groupby('rank'):
    fname = group + ".xlsx"
    tab.to_excel(fname)
```

Код выше перебирает названия групп (`group`) и соответствующие им строки датафрейма (`tab`), забирает названия и доклеивает к ним расширение `.xlsx`. А затем через метод

`.to_excel()` выгружает маленькие датафреймы для каждой группы `dat` в файлы с сформированными на предыдущем шаге названиями. Созданные файлы Excel можно найти в рабочей папке, в той папке, где находится текущий ipynb-файл с кодом.