# Exploratory Data Analysis

## 1. Знакомство с библиотекой numpy

**NumPy** — библиотека языка Python, позволяющая [удобно] работать с многомерными массивами и матрицами, содержащая математические функции. Кроме того, NumPy позволяет векторизовать многие вычисления, имеющие место в машинном обучении.

- numpy
- numpy tutorial
- 100 numpy exercises
- numpy data types

In [1]:
```
import numpy as np
```

Основным типом данных NumPy является многомерный массив элементов одного типа — numpy.ndarray. Каждый подобный массив имеет несколько *измерений* или *осей* — в частности, вектор (в классическом понимании) является одномерным массивом и имеет 1 ось, матрица является двумерным массивом и имеет 2 оси и т.д.

Создать массив из списков или кортежей можно с помощью функции `np.array()` .

`np.array` - функция, создающая объект типа `np.ndarray` .

In [2]:
```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
matrix
```

Out[2]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [3]:
```
type(matrix)
```

Out[3]:
```
numpy.ndarray
```

In [4]:
```
matrix.ndim
```

Out[4]:
```
2
```

Чтобы узнать длину массива по каждой из осей, можно воспользоваться атрибутом `shape` . Общее количество элементов выводим с помощью `size` .

In [5]:
```
matrix.shape, matrix.size
```

Out[5]:
```
((2, 3), 6)
```

Альтернативные способы задать матрицу:

In [6]:
```
np.matrix('1 2; 3 4') # аргумент - строка или список
```

```
matrix([[1, 2],
```

Out[6]:
```
       [3, 4]])
```

In [7]:
```
np.ndarray(shape=(3,3), dtype='float') # из случайных чисел
```

Out[7]:
```
array([[ 6.17779239e-31, -1.23555848e-30,  3.08889620e-31],
       [-1.23555848e-30,  2.68733969e-30, -8.34001973e-31],
       [ 3.08889620e-31, -8.34001973e-31,  4.78778910e-31]])
```

In [8]:
```
np.ndarray(shape=(3,3), dtype='int', buffer=np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

Out[8]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Массивы специального вида можно создать при помощи функций zeros, ones, empty, identity:

In [9]:
```
np.zeros((3,))
```

Out[9]:
```
array([0., 0., 0.])
```

In [10]:
```
np.ones((3, 4))
```

Out[10]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [11]:
```
np.identity(3)
```

Out[11]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [12]:
```
np.empty((2, 5))
```

Out[12]:
```
array([[-1.28822975e-231,  2.32035559e+077,  2.47032823e-323,
         0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         2.12199579e-314,  6.95337703e-309]])
```

Еще можно создавать массивы при помощи функции arange:

In [5]:
```
np.arange(2, 20, 3)
```

Out[5]:
```
array([ 2,  5,  8, 11, 14, 17])
```

In [6]:
```
np.arange(9)
```

Out[6]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [15]:
```
np.arange(9).reshape(3, 3)
```

Out[15]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Вместо значения длины массива по одному из измерений можно указать -1 — в этом случае значение будет рассчитано автоматически:

In [16]:
```
np.arange(8).reshape(2, -1)
```

Out[16]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [7]:
```
[1, 2, 3] + [4, 5, 6]
```

Out[7]:
```
[1, 2, 3, 4, 5, 6]
```

In [8]:
```
np.array([1, 2, 3]) + np.array([4, 5, 6])
```

Out[8]:
```
array([5, 7, 9])
```

In [ ]:
```

```

Теперь порешаем задачки!

**Задание 1.** Реализуйте функцию, возвращающую максимальный элемент в векторе x среди элементов, перед которыми стоит нулевой. Для x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0]) ответом является 5. Если нулевых элементов нет, функция должна возвращать None.

In [ ]:
```
x = np.array([1, 5, 8, 6, 6, 1, 7, 4, 4, 0])
y = np.random.randint(9, size=10)

def max_element_before_zero(x):
    # your code here
```

[1, 0, 2, 0, 5, 9] – 5

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [10]:
```
shift_by_one = np.hstack((1, x))
```

In [12]:
```
(shift_by_one == 0)[:np.size(shift_by_one) - 1]
```

Out[12]:
```
array([False, False, False, False, False, False, False, False, False,
       False])
```

In [18]:
```
x = np.array([1, 5, 8, 6, 6, 1, 7, 4, 4, 0])
y = np.random.randint(9, size=10)

def max_element_before_zero(x):
    shift_by_one = np.hstack((1, x))
```

```
    bool_array = (shift_by_one == 0)[:np.size(shift_by_one) - 1]
    if np.size(x[bool_array]) == 0:
        return None
    else:
        return np.max(x[bool_array])

print ('Пример:', x, 'Максимальный элемент:', max_element_before_zero(x), sep='\n')
print ('Случайный вектор:', y, 'Максимальный элемент:', max_element_before_zero(y),
```

```
Пример:
[1 5 8 6 6 1 7 4 4 0]
Максимальный элемент:
None
Случайный вектор:
[7 8 3 0 2 0 3 3 3 6]
Максимальный элемент:
3
```

**Задание 2.** Реализуйте функцию, принимающую на вход матрицу и некоторое число и возвращающую ближайший к числу элемент матрицы. Например: для X = np.arange(0,10).reshape((2, 5)) и v = 3.6 ответом будет 4.

In [19]:
```
def nearest_value(X, v):
    Y = np.abs(X - v)
    m = np.min(Y)
    inds = np.where(Y == m)
    return X[inds[0][0], inds[1][0]]

X = np.arange(0,10).reshape((2, 5))
v = 3.6
print ('Матрица:', X, 'Число:', v, 'Ближайший к числу элемент матрицы:', nearest_val
```

```
Матрица:
[[0 1 2 3 4]
 [5 6 7 8 9]]
Число:
3.6
Ближайший к числу элемент матрицы:
4
```

In [28]:
```
np.where(Y == np.min(np.abs(X - v)))[0][0]
```

Out[28]:
```
0
```

In [24]:
```
Y = X - v
```

In [29]:
```
np.argmin(np.abs(X - v))
```

Out[29]:
```
4
```

In [31]:
```
X.reshape(1, -1)[0]
```

Out[31]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [32]:
```
# Решение в одну строку
def get_nearest_value(X, v):
```

```
    return X.reshape(1, -1)[0][np.argmin(np.abs(X - v))]

print ('Матрица:', X, 'Число:', v,
       'Ближайший к числу элемент матрицы:', get_nearest_value(X, v), sep='\n')
```

```
Матрица:
[[0 1 2 3 4]
 [5 6 7 8 9]]
Число:
3.6
Ближайший к числу элемент матрицы:
4
```

**Задание 3.** Реализуйте функцию scale(X), которая принимает на вход матрицу и масштабирует каждый ее столбец (вычитает выборочное среднее и делит на стандартное отклонение). Убедитесь, что в функции не будет происходить деления на ноль. Протестируйте на случайной матрице (для её генерации можно использовать, например, функцию numpy.random.randint).

In [38]:
```
def scale(X):
    std_deviation = X.std(axis=0)
    std_deviation[std_deviation == 0] = 1
    X = X - X.mean(axis=0) # или X -= X.mean(axis=0) с преобразованием X = X.astype(
    X /= std_deviation
    return(X)

X = np.random.randint(-7, 8, size=(2,5))
print ('Матрица:', X, 'Масштабированная матрица:', scale(X), sep='\n')
```

```
Матрица:
[[ 0  4  7  3 -7]
 [-5 -7 -1  5 -3]]
Масштабированная матрица:
[[ 1.  1.  1. -1. -1.]
 [-1. -1. -1.  1.  1.]]
```

In [33]:
```
X
```

Out[33]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

In [34]:
```
np.mean(X)
```

Out[34]:
```
4.5
```

In [35]:
```
np.mean(X, axis=0)
```

Out[35]:
```
array([2.5, 3.5, 4.5, 5.5, 6.5])
```

In [36]:
```
np.mean(X, axis=1)
```

Out[36]:
```
array([2., 7.])
```

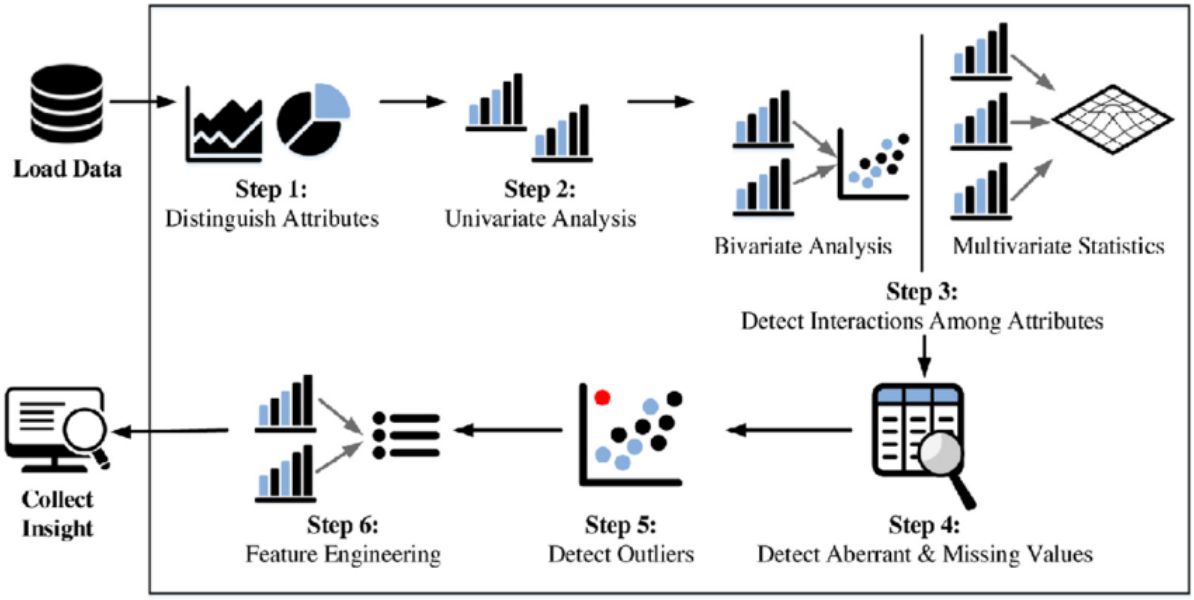
In [ ]:
```

```

# 2. Pandas и первичный анализ данных

**Pandas** (Python Data Analysis Library) — библиотека языка Python для удобных обработки и анализа данных.

Обычно первые шаги для EDA (Exploratory Data Analysis) выглядят примерно так:




Небольшой список литературы:

1. Шпаргалка по pandas
2. Введение в pandas
3. Статься в блоге сообщества OpenDataScience на Habr
4. Туториалы в официальной документации (на английском)

In [39]:
```
import pandas as pd
```

Качаем набор данных о футболистах из гитхаба курса, сохраняем в формате csv. Далее загружаем в Jupyter Notebook и внимательно указываем путь к файлу (в какие папки нужно перейти, чтобы увидеть файл).

In [40]:
```
df = pd.read_csv('data/data_football_profile.csv', sep='\t')
```

In [41]:
```
df.columns
```

Out[41]:
```
Index(['Name', 'Age', 'Nationality', 'Club', 'Value', 'Wage'], dtype='object')
```

Функция возвращает DataFrame (то есть таблицу), однако затем приобретает ещё много важных параметров, среди которых:

- sep — разделитель данных, по умолчанию ',';
- decimal — разделитель числа на целую и дробную часть, по умолчанию'.';
- names — список с названиями колонок, не обязательный параметр;

- skiprows — если файл содержит системную информацию, можно просто её пропустить. Необязательный параметр.

Дополнительные параметры можно посмотреть в официальной документации.

С помощью функции `head` можем посмотреть на первые несколько строк нашего датасета:

In [44]:
```
df.head(10)
```

Out[44]:

| | Name | Age | Nationality | Club | Wage |
|---|---|---|---|---|---|
| 0 | L. Messi | 31.0 | Argentina | FC Barcelona | 565000,0$ |
| 1 | NaN | 33.0 | Portugal | Juventus | 405000,0$ |
| 2 | Neymar Jr | 26.0 | Brazil | Paris Saint-Germain | 290000,0$ |
| 3 | De Gea | 27.0 | Spain | Manchester United | 260000,0$ |
| 4 | K. De Bruyne | 27.0 | Belgium | Manchester City | 355000,0$ |
| 5 | E. Hazard | 27.0 | Belgium | Chelsea | 340000,0$ |
| 6 | L. Modrić | 32.0 | Croatia | Real Madrid | 420000,0$ |
| 7 | L. Suárez | 31.0 | Uruguay | FC Barcelona | 455000,0$ |
| 8 | Sergio Ramos | 32.0 | Spain | Real Madrid | 380000,0$ |
| 9 | J. Oblak | 25.0 | Slovenia | Atlético Madrid | 94000,0$ |

Удалим колонку Value, к которой мы не знаем точную интерпретацию:

In [43]:
```
df.drop(['Value'], axis=1, inplace=True)
```

In [45]:
```
# последние несколько строк:
df.tail(3)
```

Out[45]:

| | Name | Age | Nationality | Club | Wage |
|---|---|---|---|---|---|
| 12894 | NaN | 16.0 | England | Cambridge United | 1000,0$ |
| 12895 | D. Walker-Rice | 17.0 | England | Tranmere Rovers | 1000,0$ |
| 12896 | G. Nugent | 16.0 | England | Tranmere Rovers | 1000,0$ |

Посмотрим на размер нашего датасета. Первое число – количество строк (наблюдений), второе – количество столбцов (признаков):

In [46]:
```
df.shape
```

Out[46]:
```
(12897, 5)
```

Если вы хотите переименовать какую-то переменную, воспользуйтесь `rename` :

In [47]:
```
df.rename({'Wage' : 'Salary'}, axis='columns', inplace=True)
```

In [48]:
```
df.columns
```

Out[48]:
```
Index(['Name', 'Age', 'Nationality', 'Club', 'Salary'], dtype='object')
```

Давайте посмотрим на информацию о датасете. В .info() можно передать дополнительные параметры, среди которых:

- verbose: печатать ли информацию о DataFrame полностью (если таблица очень большая, то некоторая информация может потеряться);
- memory_usage: печатать ли потребление памяти (по умолчанию используется True, но можно поставить либо False, что уберёт потребление памяти, либо 'deep' , что подсчитает потребление памяти более точно);
- null_counts: подсчитывать ли количество пустых элементов (по умолчанию True).

In [49]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12897 entries, 0 to 12896
Data columns (total 5 columns):
Name           12266 non-null object
Age            12242 non-null float64
Nationality    12213 non-null object
Club           12226 non-null object
Salary         12897 non-null object
dtypes: float64(1), object(4)
memory usage: 503.9+ KB
```

Можно вывести только тип данных в каждой колонке:

In [50]:
```
df.dtypes
```

Out[50]:
```
Name            object
Age            float64
Nationality     object
Club            object
Salary          object
dtype: object
```

Заметим, что зарплата у нас записана строкой.

In [51]:
```
def fix_wage(x):
    # отрезаем $
    x = x[:-1]
    # меняем запятую на точку
    x = x.replace(',', '.')
    return float(x)
```

In [52]:
```
df['Salary'] = df['Salary'].apply(fix_wage)
```

Метод describe показывает основные статистические характеристики данных по каждому числовому признаку (типы int64 и float64): число непропущенных значений, среднее, стандартное отклонение, диапазон, медиану, 0.25 и 0.75 квартили.

In [53]:
```
df.describe()
```

Out[53]:

| | Age | Salary |
|---|---|---|
| count | 12242.000000 | 12263.000000 |
| mean | 24.814900 | 7530.946750 |
| std | 4.885492 | 23338.219396 |
| min | 16.000000 | 1000.000000 |
| 25% | 21.000000 | 1000.000000 |
| 50% | 24.000000 | 2000.000000 |
| 75% | 28.000000 | 4000.000000 |
| max | 45.000000 | 565000.000000 |

Чтобы посмотреть статистику по нечисловым признакам (например, по строчным (object) или булевым (bool) данным), нужно явно указать интересующие нас типы в параметре метода describe include:

In [35]:
```
df.describe(include = ['object'])
```

Out[35]:

| | Name | Nationality | Club |
|---|---|---|---|
| **count** | 12266 | 12213 | 12226 |
| **unique** | 11742 | 155 | 650 |
| **top** | J. Rodríguez | England | Shonan Bellmare |
| **freq** | 7 | 1297 | 30 |

Было бы полезно узнать, много ли у нас пропусков в датасете.

In [57]:
```
df.isna().sum()
```

Out[57]:
```
Name           631
Age            655
Nationality    684
Club           671
Salary         634
dtype: int64
```

Чтобы удалить пропуски из данных, нужно вопспользоваться `df.dropna()`, либо заполнить их значениями (например, средним) - `df.fillna(df['column_name'].mean())`. Если в датасете содержатся дубликаты строк - воспользуйтесь методом `df.drop_duplicates()`.

In [58]:
```
# заполним количественные переменные средними значениями (медианой)
for column in df.columns:
    if df[column].dtype != 'object':
        df[column] = df[column].fillna(df[column].median())

# у оставшихся переменных удалим строки с пропусками
df.dropna(inplace=True)

df.isna().sum()
```

```
Name           0
```

Out[58]:
```
Age            0
Nationality    0
Club           0
Salary         0
dtype: int64
```

Сохранить полученную таблицу с помощью `to_csv`. Параметр `index` по дефолту стоит `True`, из-за чего первой колонкой в сохраненном файле будут индексы, отключим это.

*А можно и не в csv. Смотрите тут.*

In [59]:
```
df.head()
```

Out[59]:

| | Name | Age | Nationality | Club | Salary |
|---|---|---|---|---|---|
| 0 | L. Messi | 31.0 | Argentina | FC Barcelona | 565000.0 |
| 2 | Neymar Jr | 26.0 | Brazil | Paris Saint-Germain | 290000.0 |
| 3 | De Gea | 27.0 | Spain | Manchester United | 260000.0 |
| 4 | K. De Bruyne | 27.0 | Belgium | Manchester City | 355000.0 |
| 5 | E. Hazard | 27.0 | Belgium | Chelsea | 340000.0 |

In [38]:
```
# df.to_csv('Путь к папке/Название.csv', index=False)
```

Выведем уникальные значения по возрасту и сколько раз каждое из них встречается в датасете (по убыванию).

In [64]:
```
df['Age'].value_counts()
```

Out[64]:
```
24.0    1288
21.0     876
22.0     808
20.0     786
19.0     724
23.0     715
25.0     693
26.0     688
27.0     616
28.0     580
18.0     536
29.0     536
30.0     436
31.0     372
32.0     312
33.0     221
34.0     220
17.0     209
35.0     124
36.0     100
37.0      58
16.0      36
38.0      32
39.0      22
40.0      11
41.0       4
44.0       2
42.0       1
```

```
45.0       1
Name: Age, dtype: int64
```

Чтобы вывести уникальные значения в столбце или их количество, нужно использовать `unique` и `nunique` соответственно. Посмотрим, сколько у нас уникальных футбольных клубов.

In [40]:
```
print('Всего {} футбольных клубов'.format(df['Club'].nunique()))
```

```
Всего 650 футбольных клубов
```

In [41]:
```
df['Club'].unique()[:10]
```

Out[41]:
```
array(['FC Barcelona', 'Paris Saint-Germain', 'Manchester United',
       'Manchester City', 'Chelsea', 'Real Madrid', 'Atlético Madrid',
       'FC Bayern München', 'Juventus', 'Liverpool'], dtype=object)
```

Посмотрим на среднюю зарплату по клубу:

In [42]:
```
grouped = df.groupby('Club', as_index=False)['Salary'].mean()
# добавим сортировку по убыванию
grouped.sort_values(by='Salary', ascending=False)
```

Out[42]:

| | Club | Salary |
|---|---|---|
| **469** | Real Madrid | 187500.000000 |
| **212** | FC Barcelona | 184722.222222 |
| **325** | Juventus | 148461.538462 |
| **373** | Manchester City | 137761.904762 |
| **134** | Chelsea | 103894.736842 |
| **...** | ... | ... |
| **617** | Viktoria Plzeň | 1000.000000 |
| **90** | Boyacá Chicó FC | 1000.000000 |
| **317** | Itagüí Leones FC | 1000.000000 |
| **195** | Dynamo Kyiv | 1000.000000 |
| **416** | Olympiacos CFP | 1000.000000 |

650 rows × 2 columns

In [67]:
```
df.groupby('Club', as_index=False)['Salary'].mean()
```

Out[67]:

| | Club | Salary |
|---|---|---|
| 0 | SSV Jahn Regensburg | 3222.222222 |
| 1 | 1. FC Heidenheim 1846 | 4000.000000 |
| 2 | 1. FC Kaiserslautern | 1454.545455 |
| 3 | 1. FC Köln | 9200.000000 |
| 4 | 1. FC Magdeburg | 3842.105263 |

| | Club | Salary |
|---|---|---|
| ... | ... | ... |
| 645 | Zagłębie Sosnowiec | 1047.619048 |
| 646 | Çaykur Rizespor | 5095.238095 |
| 647 | Örebro SK | 1454.545455 |
| 648 | Östersunds FK | 2055.555556 |
| 649 | Śląsk Wrocław | 2045.454545 |

650 rows × 2 columns

Добавим еще подсчет минимума, максимума и медианы по каждой группе:

In [43]:
```
df.groupby('Club')['Salary'].agg(['mean', 'min', 'max', 'median'])
```

Out[43]:

| Club | mean | min | max | median |
|---|---|---|---|---|
| **SSV Jahn Regensburg** | 3222.222222 | 1000.0 | 6000.0 | 3000.0 |
| **1. FC Heidenheim 1846** | 4000.000000 | 1000.0 | 14000.0 | 3000.0 |
| **1. FC Kaiserslautern** | 1454.545455 | 1000.0 | 3000.0 | 1000.0 |
| **1. FC Köln** | 9200.000000 | 1000.0 | 26000.0 | 4000.0 |
| **1. FC Magdeburg** | 3842.105263 | 1000.0 | 8000.0 | 4000.0 |
| **...** | ... | ... | ... | ... |
| **Zagłębie Sosnowiec** | 1047.619048 | 1000.0 | 2000.0 | 1000.0 |
| **Çaykur Rizespor** | 5095.238095 | 1000.0 | 13000.0 | 4000.0 |
| **Örebro SK** | 1454.545455 | 1000.0 | 2000.0 | 1000.0 |
| **Östersunds FK** | 2055.555556 | 1000.0 | 8000.0 | 1000.0 |
| **Śląsk Wrocław** | 2045.454545 | 1000.0 | 4000.0 | 2000.0 |

650 rows × 4 columns

Сгруппируем одновременно по стране и клубу:

In [44]:
```
df.groupby(['Nationality', 'Club'], as_index=False)['Salary'].mean()
```

Out[44]:

| | Nationality | Club | Salary |
|---|---|---|---|
| **0** | Afghanistan | Notts County | 2000.0 |
| **1** | Afghanistan | SV Meppen | 1000.0 |
| **2** | Afghanistan | Walsall | 1000.0 |
| **3** | Albania | AC Ajaccio | 2000.0 |
| **4** | Albania | Aalborg BK | 1000.0 |
| **...** | ... | ... | ... |

| | Nationality | Club | Salary |
|---|---|---|---|
| **3242** | Zimbabwe | Club Brugge KV | 20000.0 |
| **3243** | Zimbabwe | Hobro IK | 5000.0 |
| **3244** | Zimbabwe | Le Havre AC | 2000.0 |
| **3245** | Zimbabwe | Orlando Pirates | 1000.0 |
| **3246** | Zimbabwe | Sparta Praha | 1000.0 |

3247 rows × 3 columns

Добавим сортировку внутри групп:

In [68]:
```
a = lambda x: x ** 2
```

In [69]:
```
a(4)
```

Out[69]:
16

In [70]:
```
(lambda x: x ** 2)(4)
```

Out[70]:
16

In [45]:
```
df.groupby(['Nationality', 'Club']).apply(lambda x: x.sort_values(by='Salary', ascen
```

Out[45]:

| | | | Name | Age | Nationality | Club | Salary |
|---|---|---|---|---|---|---|---|
| **Nationality** | **Club** | | | | | | |
| **Afghanistan** | **Notts County** | **10007** | N. Husin | 21.0 | Afghanistan | Notts County | 2000.0 |
| | **SV Meppen** | **7695** | H. Amin | 26.0 | Afghanistan | SV Meppen | 1000.0 |
| | **Walsall** | **8395** | M. Kouhyar | 20.0 | Afghanistan | Walsall | 1000.0 |
| **Albania** | **AC Ajaccio** | **4697** | Q. Laçi | 22.0 | Albania | AC Ajaccio | 2000.0 |
| | **Aalborg BK** | **12510** | B. Bytyqi | 21.0 | Albania | Aalborg BK | 1000.0 |
| **...** | **...** | **...** | ... | ... | ... | ... | ... |
| **Zimbabwe** | **Club Brugge KV** | **468** | M. Nakamba | 24.0 | Zimbabwe | Club Brugge KV | 20000.0 |
| | **Hobro IK** | **2772** | Q. Antipas | 34.0 | Zimbabwe | Hobro IK | 5000.0 |
| | **Le Havre AC** | **6090** | T. Kadewere | 22.0 | Zimbabwe | Le Havre AC | 2000.0 |
| | **Orlando Pirates** | **6502** | M. Munetsi | 24.0 | Zimbabwe | Orlando Pirates | 1000.0 |
| | **Sparta Praha** | **1750** | C. Nhamoinesu | 32.0 | Zimbabwe | Sparta Praha | 1000.0 |

11007 rows × 5 columns

Теперь удалим лишние колонки. Обратите внимание на обратный слэш, это line continuation character.

In [46]:
```
df.groupby(['Nationality', 'Club']).apply(lambda x: x.sort_values(by='Salary', ascen
```

```
drop(['Nationality', 'Club'], ax
```

Out[46]:

| Nationality | Club | | Name | Age | Salary |
|---|---|---|---|---|---|
| **Afghanistan** | **Notts County** | **10007** | N. Husin | 21.0 | 2000.0 |
| | **SV Meppen** | **7695** | H. Amin | 26.0 | 1000.0 |
| | **Walsall** | **8395** | M. Kouhyar | 20.0 | 1000.0 |
| **Albania** | **AC Ajaccio** | **4697** | Q. Laçi | 22.0 | 2000.0 |
| | **Aalborg BK** | **12510** | B. Bytyqi | 21.0 | 1000.0 |
| **...** | **...** | **...** | ... | ... | ... |
| **Zimbabwe** | **Club Brugge KV** | **468** | M. Nakamba | 24.0 | 20000.0 |
| | **Hobro IK** | **2772** | Q. Antipas | 34.0 | 5000.0 |
| | **Le Havre AC** | **6090** | T. Kadewere | 22.0 | 2000.0 |
| | **Orlando Pirates** | **6502** | M. Munetsi | 24.0 | 1000.0 |
| | **Sparta Praha** | **1750** | C. Nhamoinesu | 32.0 | 1000.0 |

11007 rows × 3 columns

Посчитаем арифметическое среднее, моду и медиану возраста футболистов (количественной переменной):

In [71]:
```
round(2.5)
```

Out[71]:
```
2
```

In [72]:
```
round(4.5)
```

Out[72]:
```
4
```

In [73]:
```
print('Среднее:', np.round(df['Age'].mean(), 2),
      'Медиана:', df['Age'].median(),
      'Мода:', df['Age'].mode()[0])
```

```
Среднее: 24.78 Медиана: 24.0 Мода: 24.0
```

Для качественных переменных с помощью pandas можно вывести моду. Посмотрим на самую часто встречающуюся национальность:

In [74]:
```
df['Nationality'].mode()
```

Out[74]:
```
0    England
dtype: object
```

Часто возникает необходимость выбрать данные из DataFrame по определённому условию. Например, если в уже известном нам наборе данных о футболистах мы хотим выбрать только тех, у кого возраст больше 20 лет, используется следующий код:

In [75]:
```
df.Age > 20
```

Out[75]:
```
0         True
2         True
3         True
4         True
5         True
         ...
12891    False
12892    False
12893    False
12895    False
12896    False
Name: Age, Length: 11007, dtype: bool
```

In [49]:
```
df[df.Age > 20]
```

Out[49]:

| | Name | Age | Nationality | Club | Salary |
|---|---|---|---|---|---|
| **0** | L. Messi | 31.0 | Argentina | FC Barcelona | 565000.0 |
| **2** | Neymar Jr | 26.0 | Brazil | Paris Saint-Germain | 290000.0 |
| **3** | De Gea | 27.0 | Spain | Manchester United | 260000.0 |
| **4** | K. De Bruyne | 27.0 | Belgium | Manchester City | 355000.0 |
| **5** | E. Hazard | 27.0 | Belgium | Chelsea | 340000.0 |
| **...** | ... | ... | ... | ... | ... |
| **12838** | D. Mackay | 21.0 | Scotland | Kilmarnock | 1000.0 |
| **12855** | H. Norris | 24.0 | England | Oldham Athletic | 1000.0 |
| **12861** | Y. Uchimura | 33.0 | Japan | Hokkaido Consadole Sapporo | 1000.0 |
| **12873** | K. Pilkington | 44.0 | England | Cambridge United | 1000.0 |
| **12889** | M. Baldisimo | 24.0 | Canada | Vancouver Whitecaps FC | 1000.0 |

8716 rows × 5 columns

**Задание:** Выберите футболистов, возраст которых больше среднего возраста футболистов, при условии, что они принадлежат ФК Барселона (Club == 'FC Barcelona').

In [77]:
```
True & False
```

Out[77]:
```
False
```

In [78]:
```
True + True - False
```

Out[78]:
```
2
```

In [50]:
```
df[(df.Age > df.Age.mean()) & (df.Club == 'FC Barcelona')]
```

Out[50]:

| | Name | Age | Nationality | Club | Salary |
|---|---|---|---|---|---|
| **0** | L. Messi | 31.0 | Argentina | FC Barcelona | 565000.0 |
| **7** | L. Suárez | 31.0 | Uruguay | FC Barcelona | 455000.0 |

| | Name | Age | Nationality | Club | Salary |
|---|---|---|---|---|---|
| **18** | M. ter Stegen | 26.0 | Germany | FC Barcelona | 240000.0 |
| **20** | Sergio Busquets | 29.0 | Spain | FC Barcelona | 315000.0 |
| **32** | Coutinho | 26.0 | Brazil | FC Barcelona | 340000.0 |
| **49** | Jordi Alba | 29.0 | Spain | FC Barcelona | 250000.0 |
| **54** | Piqué | 31.0 | Spain | FC Barcelona | 240000.0 |
| **204** | J. Cillessen | 29.0 | Netherlands | FC Barcelona | 135000.0 |
| **605** | T. Vermaelen | 32.0 | Belgium | FC Barcelona | 2000.0 |

In [82]:
```
df['Name_len'] = df['Name'].apply(len)
```

In [85]:
```
df['salary_1000'] = df['Salary'].apply(lambda x: x / 1000)
```

In [86]:
```
df.head()
```

Out[86]:

| | Name | Age | Nationality | Club | Salary | Name_len | salary_1000 |
|---|---|---|---|---|---|---|---|
| 0 | L. Messi | 31.0 | Argentina | FC Barcelona | 565000.0 | 8 | 565.0 |
| 2 | Neymar Jr | 26.0 | Brazil | Paris Saint-Germain | 290000.0 | 9 | 290.0 |
| 3 | De Gea | 27.0 | Spain | Manchester United | 260000.0 | 6 | 260.0 |
| 4 | K. De Bruyne | 27.0 | Belgium | Manchester City | 355000.0 | 12 | 355.0 |
| 5 | E. Hazard | 27.0 | Belgium | Chelsea | 340000.0 | 9 | 340.0 |

Чтобы объединить данные из нескольких датасетов по ключу (общей колонке), в pandas можно воспользовать встроенными аналогами SQL методов. В метод `join` в качестве аргумента how нужно указать тип объединения датасетов: inner, outer, left или right.

In [51]:
```
# Вот так:
```

In [52]:
```
df_info = pd.read_csv('data/data_football_info.csv', sep='\t')

joined_dfs = df_info.set_index('Name').join(df.set_index('Name'), how='inner').reset
joined_dfs.head(5)
```

Out[52]:

| | Name | Unnamed: 0 | Position | Crossing | Finishing | HeadingAccuracy | ShortPassing | Volleys |
|---|---|---|---|---|---|---|---|---|
| **0** | A. Abang | 8574.0 | ST | 30.0 | 61.0 | 67.0 | 52.0 | 58.0 |
| **1** | A. Abdellaoui | 10604.0 | NaN | 56.0 | 25.0 | 56.0 | NaN | 32.0 |
| **2** | A. Abdi | 2188.0 | CM | 68.0 | 61.0 | 59.0 | 74.0 | 64.0 |
| **3** | A. Abdu Jaber | 7807.0 | ST | 39.0 | 66.0 | NaN | 49.0 | 64.0 |

| | Name | Unnamed: 0 | Position | Crossing | Finishing | HeadingAccuracy | ShortPassing | Volleys |
|---|---|---|---|---|---|---|---|---|
| **4** | A. Abdulhameed | 11820.0 | GK | 10.0 | 9.0 | 13.0 | 26.0 | 6.0 |

5 rows × 41 columns

In [ ]: