Topic 1. Exploratory data analysis with Pandas

mlcourse.ai – Open Machine Learning Course

Article outline

1 Demonstration of main Pandas methods

Well... There are dozens of cool tutorials on Pandas and visual data analysis. If you are already familiar with these topics, you can wait for the 3rd article in the series, where we get into machine learning.

Pandas is a Python library that provides extensive means for data analysis. Data scientists often work with data stored in table formats like `.csv`, `.tsv`, or `.xlsx`. Pandas makes it very convenient to load, process, and analyze such tabular data using SQL-like queries. In conjunction with `Matplotlib` and `Seaborn`, `Pandas` provides a wide range of opportunities for visual analysis of tabular data.

The main data structures in `Pandas` are implemented with **Series** and **DataFrame** classes. The former is a one-dimensional indexed array of some fixed data type. The latter is a two-dimensional data structure - a table - where each column contains data of the same type. You can see it as a dictionary of `Series` instances. `DataFrames` are great for representing real data: rows correspond to instances (examples, observations, etc.), and columns correspond to features of these instances.

```
In [1]: import numpy as np
        import pandas as pd
        # we don't like warnings
        # you can comment the following 2 lines if you'd like to
        import warnings
        warnings.filterwarnings('ignore')
```

We'll demonstrate the main methods in action by analyzing a dataset on the churn rate of telecom operator clients. Let's read the data (using `read_csv`), and take a look at the first 5 lines using the `head` method:

```
In [2]: df = pd.read_csv('../../data/telecom_churn.csv')
        df.head()
```

```
Out[2]:   State  Account length  Area code International plan Voice mail plan  \
        0    KS             128        415                 No             Yes
        1    OH             107        415                 No             Yes
        2    NJ             137        415                 No              No
        3    OH              84        408                Yes              No
        4    OK              75        415                Yes              No

           Number vmail messages  Total day minutes  Total day calls  \
        0                     25              265.1              110
        1                     26              161.6              123
        2                      0              243.4              114
        3                      0              299.4               71
        4                      0              166.7              113

           Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
        0             45.07              197.4               99             16.78
        1             27.47              195.5              103             16.62
        2             41.38              121.2              110             10.30
        3             50.90               61.9               88              5.26
        4             28.34              148.3              122             12.61

           Total night minutes  Total night calls  Total night charge  \
        0                244.7                 91               11.01
        1                254.4                103               11.45
        2                162.6                104                7.32
        3                196.9                 89                8.86
        4                186.9                121                8.41

           Total intl minutes  Total intl calls  Total intl charge  \
        0                10.0                 3               2.70
        1                13.7                 3               3.70
        2                12.2                 5               3.29
        3                 6.6                 7               1.78
        4                10.1                 3               2.73

           Customer service calls  Churn
        0                       1  False
        1                       1  False
        2                       0  False
        3                       2  False
        4                       3  False
```

About printing DataFrames in Jupyter notebooks

In Jupyter notebooks, Pandas DataFrames are printed as these pretty tables seen above while `print(df.head())` is less nicely formatted. By default, Pandas displays 20 columns and 60 rows, so, if your DataFrame is bigger, use the `set_option` function as shown in the example below:

```
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
```

Recall that each row corresponds to one client, an **instance**, and columns are **features** of this instance.

Let's have a look at data dimensionality, feature names, and feature types.

```
In [3]: print(df.shape)
```

```
(3333, 20)
```

From the output, we can see that the table contains 3333 rows and 20 columns.

Now let's try printing out column names using `columns`:

```
In [4]: print(df.columns)
```

```
Index(['State', 'Account length', 'Area code', 'International plan',
       'Voice mail plan', 'Number vmail messages', 'Total day minutes',
       'Total day calls', 'Total day charge', 'Total eve minutes',
       'Total eve calls', 'Total eve charge', 'Total night minutes',
       'Total night calls', 'Total night charge', 'Total intl minutes',
       'Total intl calls', 'Total intl charge', 'Customer service calls',
       'Churn'],
      dtype='object')
```

We can use the `info()` method to output some general information about the dataframe:

```
In [5]: print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
State                     3333 non-null object
Account length            3333 non-null int64
Area code                 3333 non-null int64
International plan        3333 non-null object
Voice mail plan           3333 non-null object
Number vmail messages     3333 non-null int64
Total day minutes         3333 non-null float64
Total day calls           3333 non-null int64
Total day charge          3333 non-null float64
Total eve minutes         3333 non-null float64
Total eve calls           3333 non-null int64
Total eve charge          3333 non-null float64
Total night minutes       3333 non-null float64
Total night calls         3333 non-null int64
Total night charge        3333 non-null float64
```

```
Total intl minutes        3333 non-null float64
Total intl calls          3333 non-null int64
Total intl charge         3333 non-null float64
Customer service calls    3333 non-null int64
Churn                     3333 non-null bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 498.1+ KB
None
```

`bool`, `int64`, `float64` and `object` are the data types of our features. We see that one feature is logical (`bool`), 3 features are of type `object`, and 16 features are numeric. With this same method, we can easily see if there are any missing values. Here, there are none because each column contains 3333 observations, the same number of rows we saw before with `shape`.

We can **change the column type** with the `astype` method. Let's apply this method to the `Churn` feature to convert it into `int64`:

```
In [6]: df['Churn'] = df['Churn'].astype('int64')
```

The `describe` method shows basic statistical characteristics of each numerical feature (`int64` and `float64` types): number of non-missing values, mean, standard deviation, range, median, 0.25 and 0.75 quartiles.

```
In [7]: df.describe()
```

```
Out[7]:        Account length    Area code  Number vmail messages  Total day minutes  \
        count     3333.000000  3333.000000            3333.000000        3333.000000
        mean       101.064806   437.182418               8.099010         179.775098
        std         39.822106    42.371290              13.688365          54.467389
        min          1.000000   408.000000               0.000000           0.000000
        25%         74.000000   408.000000               0.000000         143.700000
        50%        101.000000   415.000000               0.000000         179.400000
        75%        127.000000   510.000000              20.000000         216.400000
        max        243.000000   510.000000              51.000000         350.800000

               Total day calls  Total day charge  Total eve minutes  Total eve calls  \
        count      3333.000000       3333.000000        3333.000000      3333.000000
        mean        100.435644         30.562307         200.980348       100.114311
        std          20.069084          9.259435          50.713844        19.922625
        min           0.000000          0.000000           0.000000         0.000000
        25%          87.000000         24.430000         166.600000        87.000000
        50%         101.000000         30.500000         201.400000       100.000000
        75%         114.000000         36.790000         235.300000       114.000000
        max         165.000000         59.640000         363.700000       170.000000

               Total eve charge  Total night minutes  Total night calls  \
        count       3333.000000          3333.000000        3333.000000
        mean          17.083540           200.872037         100.107711
        std            4.310668            50.573847          19.568609
```

```
min               0.000000            23.200000           33.000000
25%              14.160000           167.000000           87.000000
50%              17.120000           201.200000          100.000000
75%              20.000000           235.300000          113.000000
max              30.910000           395.000000          175.000000

       Total night charge  Total intl minutes  Total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       Total intl charge  Customer service calls        Churn
count        3333.000000             3333.000000  3333.000000
mean            2.764581                1.562856     0.144914
std             0.753773                1.315491     0.352067
min             0.000000                0.000000     0.000000
25%             2.300000                1.000000     0.000000
50%             2.780000                1.000000     0.000000
75%             3.270000                2.000000     0.000000
max             5.400000                9.000000     1.000000
```

In order to see statistics on non-numerical features, one has to explicitly indicate data types of interest in the `include` parameter.

```
In [8]: df.describe(include=['object', 'bool'])
```

```
Out[8]:        State International plan Voice mail plan
        count   3333               3333            3333
        unique    51                  2               2
        top       WV                 No              No
        freq     106               3010            2411
```

For categorical (type `object`) and boolean (type `bool`) features we can use the `value_counts` method. Let's have a look at the distribution of `Churn`:

```
In [9]: df['Churn'].value_counts()
```

```
Out[9]: 0    2850
        1     483
        Name: Churn, dtype: int64
```

2850 users out of 3333 are *loyal*; their `Churn` value is `0`. To calculate fractions, pass `normalize=True` to the `value_counts` function.

```
In [10]: df['Churn'].value_counts(normalize=True)
```

```
Out[10]: 0    0.855086
         1    0.144914
         Name: Churn, dtype: float64
```

1.1 Sorting

A DataFrame can be sorted by the value of one of the variables (i.e columns). For example, we can sort by *Total day charge* (use `ascending=False` to sort in descending order):

```
In [11]: df.sort_values(by='Total day charge', ascending=False).head()
```

```
Out[11]:      State  Account length  Area code International plan Voice mail plan  \
         365     CO             154        415                 No              No
         985     NY              64        415                Yes              No
         2594    OH             115        510                Yes              No
         156     OH              83        415                 No              No
         605     MO             112        415                 No              No

               Number vmail messages  Total day minutes  Total day calls  \
         365                       0              350.8               75
         985                       0              346.8               55
         2594                      0              345.3               81
         156                       0              337.4              120
         605                       0              335.5               77

               Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
         365              59.64              216.5               94             18.40
         985              58.96              249.5               79             21.21
         2594             58.70              203.4              106             17.29
         156              57.36              227.4              116             19.33
         605              57.04              212.5              109             18.06

               Total night minutes  Total night calls  Total night charge  \
         365                 253.9                100               11.43
         985                 275.4                102               12.39
         2594                217.5                107                9.79
         156                 153.9                114                6.93
         605                 265.0                132               11.93

               Total intl minutes  Total intl calls  Total intl charge  \
         365                 10.1                 9               2.73
         985                 13.3                 9               3.59
         2594                11.8                 8               3.19
         156                 15.8                 7               4.27
         605                 12.7                 8               3.43

               Customer service calls  Churn
         365                        1      1
```

```
985                        1      1
2594                       1      1
156                        0      1
605                        2      1
```

We can also sort by multiple columns:

```
In [12]: df.sort_values(by=['Churn', 'Total day charge'],
                 ascending=[True, False]).head()
```

```
Out[12]:      State  Account length  Area code International plan Voice mail plan  \
        688     MN              13        510                 No             Yes
        2259    NC             210        415                 No             Yes
        534     LA              67        510                 No              No
        575     SD             114        415                 No             Yes
        2858    AL             141        510                 No             Yes

              Number vmail messages  Total day minutes  Total day calls  \
        688                      21              315.6              105
        2259                     31              313.8               87
        534                       0              310.4               97
        575                      36              309.9               90
        2858                     28              308.0              123

              Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
        688              53.65              208.9               71             17.76
        2259             53.35              147.7              103             12.55
        534              52.77               66.5              123              5.65
        575              52.68              200.3               89             17.03
        2858             52.36              247.8              128             21.06

              Total night minutes  Total night calls  Total night charge  \
        688                 260.1                123               11.70
        2259                192.7                 97                8.67
        534                 246.5                 99               11.09
        575                 183.5                105                8.26
        2858                152.9                103                6.88

              Total intl minutes  Total intl calls  Total intl charge  \
        688                 12.1                 3               3.27
        2259                10.1                 7               2.73
        534                  9.2                10               2.48
        575                 14.2                 2               3.83
        2858                 7.4                 3               2.00

              Customer service calls  Churn
        688                        3      0
        2259                       3      0
```

```
534                          4       0
575                          1       0
2858                         1       0
```

1.2 Indexing and retrieving data

A DataFrame can be indexed in a few different ways.

To get a single column, you can use a `DataFrame['Name']` construction. Let's use this to answer a question about that column alone: **what is the proportion of churned users in our dataframe?**

```
In [13]: df['Churn'].mean()

Out[13]: 0.14491449144914492
```

14.5% is actually quite bad for a company; such a churn rate can make the company go bankrupt.

Boolean indexing with one column is also very convenient. The syntax is `df[P(df['Name'])]`, where `P` is some logical condition that is checked for each element of the `Name` column. The result of such indexing is the DataFrame consisting only of rows that satisfy the `P` condition on the `Name` column.

Let's use it to answer the question:

What are average values of numerical features for churned users?

```
In [14]: df[df['Churn'] == 1].mean()

Out[14]: Account length            102.664596
         Area code                 437.817805
         Number vmail messages       5.115942
         Total day minutes         206.914079
         Total day calls           101.335404
         Total day charge           35.175921
         Total eve minutes         212.410145
         Total eve calls           100.561077
         Total eve charge           18.054969
         Total night minutes       205.231677
         Total night calls         100.399586
         Total night charge          9.235528
         Total intl minutes         10.700000
         Total intl calls            4.163561
         Total intl charge           2.889545
         Customer service calls      2.229814
         Churn                       1.000000
         dtype: float64
```

How much time (on average) do churned users spend on the phone during daytime?

```
In [15]: df[df['Churn'] == 1]['Total day minutes'].mean()

Out[15]: 206.91407867494814
```

What is the maximum length of international calls among loyal users (`Churn == 0`) who do not have an international plan?

```
In [16]: df[(df['Churn'] == 0) & (df['International plan'] == 'No')]
                              ['Total intl minutes'].max()
```

```
Out[16]: 18.9
```

DataFrames can be indexed by column name (label) or row name (index) or by the serial number of a row. The `loc` method is used for **indexing by name**, while `iloc()` is used for **indexing by number**.

In the first case below, we say *"give us the values of the rows with index from 0 to 5 (inclusive) and columns labeled from State to Area code (inclusive)"*. In the second case, we say *"give us the values of the first five rows in the first three columns"* (as in a typical Python slice: the maximal value is not included).

```
In [17]: df.loc[0:5, 'State':'Area code']
```

```
Out[17]:   State  Account length  Area code
         0    KS             128        415
         1    OH             107        415
         2    NJ             137        415
         3    OH              84        408
         4    OK              75        415
         5    AL             118        510
```

```
In [18]: df.iloc[0:5, 0:3]
```

```
Out[18]:   State  Account length  Area code
         0    KS             128        415
         1    OH             107        415
         2    NJ             137        415
         3    OH              84        408
         4    OK              75        415
```

If we need the first or the last line of the data frame, we can use the `df[:1]` or `df[-1:]` construct:

```
In [19]: df[-1:]
```

```
Out[19]:      State  Account length  Area code International plan Voice mail plan  \
         3332    TN              74        415                 No             Yes

               Number vmail messages  Total day minutes  Total day calls  \
         3332                     25              234.4              113

               Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
         3332             39.85              265.9               82              22.6
```

```
      Total night minutes  Total night calls  Total night charge  \
3332                241.4                 77               10.86

      Total intl minutes  Total intl calls  Total intl charge  \
3332                13.7                 4                3.7

      Customer service calls  Churn
3332                       0      0
```

1.3 Applying Functions to Cells, Columns and Rows

To apply functions to each column, use `apply()`**:**

```
In [20]: df.apply(np.max)
```

```
Out[20]: State                         WY
         Account length               243
         Area code                    510
         International plan           Yes
         Voice mail plan              Yes
         Number vmail messages         51
         Total day minutes          350.8
         Total day calls              165
         Total day charge           59.64
         Total eve minutes          363.7
         Total eve calls              170
         Total eve charge           30.91
         Total night minutes          395
         Total night calls            175
         Total night charge         17.77
         Total intl minutes            20
         Total intl calls              20
         Total intl charge            5.4
         Customer service calls         9
         Churn                          1
         dtype: object
```

The `apply` method can also be used to apply a function to each row. To do this, specify `axis=1`. Lambda functions are very convenient in such scenarios. For example, if we need to select all states starting with W, we can do it like this:

```
In [21]: df[df['State'].apply(lambda state: state[0] == 'W')].head()
```

```
Out[21]:    State  Account length  Area code International plan Voice mail plan  \
         9     WV             141        415                Yes             Yes
         26    WY              57        408                 No             Yes
         44    WI              64        510                 No              No
         49    WY              97        415                 No             Yes
         54    WY              87        415                 No              No
```

```
    Number vmail messages  Total day minutes  Total day calls  \
9                      37              258.6               84
26                     39              213.0              115
44                      0              154.0               67
49                     24              133.2              135
54                      0              151.0               83

    Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
9              43.96              222.0              111             18.87
26             36.21              191.1              112             16.24
44             26.18              225.8              118             19.19
49             22.64              217.2               58             18.46
54             25.67              219.7              116             18.67

    Total night minutes  Total night calls  Total night charge  \
9                 326.4                 97               14.69
26                182.7                115                8.22
44                265.3                 86               11.94
49                 70.6                 79                3.18
54                203.9                127                9.18

    Total intl minutes  Total intl calls  Total intl charge  \
9                 11.2                 5               3.02
26                 9.5                 3               2.57
44                 3.5                 3               0.95
49                11.0                 3               2.97
54                 9.7                 3               2.62

    Customer service calls  Churn
9                        0      0
26                       0      0
44                       1      0
49                       1      0
54                       5      1
```

The map method can be used to **replace values in a column** by passing a dictionary of the form `{old_value: new_value}` as its argument:

```
In [22]: d = {'No' : False, 'Yes' : True}
         df['International plan'] = df['International plan'].map(d)
         df.head()
```

```
Out[22]:   State  Account length  Area code  International plan Voice mail plan  \
         0    KS             128        415               False             Yes
         1    OH             107        415               False             Yes
         2    NJ             137        415               False              No
         3    OH              84        408                True              No
```

```
4    OK              75        415                True                No

   Number vmail messages  Total day minutes  Total day calls  \
0                     25              265.1              110
1                     26              161.6              123
2                      0              243.4              114
3                      0              299.4               71
4                      0              166.7              113

   Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
0             45.07              197.4               99             16.78
1             27.47              195.5              103             16.62
2             41.38              121.2              110             10.30
3             50.90               61.9               88              5.26
4             28.34              148.3              122             12.61

   Total night minutes  Total night calls  Total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   Total intl minutes  Total intl calls  Total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   Customer service calls  Churn
0                       1      0
1                       1      0
2                       0      0
3                       2      0
4                       3      0
```

The same thing can be done with the `replace` method:

```
In [23]: df = df.replace({'Voice mail plan': d})
         df.head()
```

```
Out[23]:   State  Account length  Area code  International plan  Voice mail plan  \
         0    KS             128        415               False             True
         1    OH             107        415               False             True
         2    NJ             137        415               False            False
         3    OH              84        408                True            False
         4    OK              75        415                True            False
```

```
   Number vmail messages  Total day minutes  Total day calls  \
0                     25              265.1              110
1                     26              161.6              123
2                      0              243.4              114
3                      0              299.4               71
4                      0              166.7              113

   Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
0             45.07              197.4               99             16.78
1             27.47              195.5              103             16.62
2             41.38              121.2              110             10.30
3             50.90               61.9               88              5.26
4             28.34              148.3              122             12.61

   Total night minutes  Total night calls  Total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   Total intl minutes  Total intl calls  Total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   Customer service calls  Churn
0                       1      0
1                       1      0
2                       0      0
3                       2      0
4                       3      0
```

1.4 Grouping

In general, grouping data in Pandas works as follows:

```
df.groupby(by=grouping_columns)[columns_to_show].function()
```

1. First, the `groupby` method divides the `grouping_columns` by their values. They become a new index in the resulting dataframe.
2. Then, columns of interest are selected (`columns_to_show`). If `columns_to_show` is not included, all non groupby clauses will be included.
3. Finally, one or several functions are applied to the obtained groups per selected columns.

Here is an example where we group the data according to the values of the Churn variable and display statistics of three columns in each group:

```
In [24]: columns_to_show = ['Total day minutes', 'Total eve minutes',
                            'Total night minutes']

         df.groupby(['Churn'])[columns_to_show].describe(percentiles=[])
```

```
Out[24]:       Total day minutes                                           \
                           count        mean        std  min    50%    max
         Churn
         0                2850.0  175.175754  50.181655  0.0  177.2  315.6
         1                 483.0  206.914079  68.997792  0.0  217.6  350.8

               Total eve minutes                                           \
                           count        mean        std   min    50%    max
         Churn
         0                2850.0  199.043298  50.292175   0.0  199.6  361.8
         1                 483.0  212.410145  51.728910  70.9  211.3  363.7

               Total night minutes
                             count        mean        std   min     50%    max
         Churn
         0                  2850.0  200.133193  51.105032  23.2  200.25  395.0
         1                   483.0  205.231677  47.132825  47.4  204.80  354.9
```

Let's do the same thing, but slightly differently by passing a list of functions to agg():

```
In [25]: columns_to_show = ['Total day minutes', 'Total eve minutes',
                            'Total night minutes']

         df.groupby(['Churn'])[columns_to_show].agg([np.mean, np.std, np.min,
                                                      np.max])
```

```
Out[25]:       Total day minutes                           Total eve minutes             \
                            mean        std amin   amax                mean        std
         Churn
         0            175.175754  50.181655  0.0  315.6          199.043298  50.292175
         1            206.914079  68.997792  0.0  350.8          212.410145  51.728910

                        Total night minutes
                amin   amax                mean        std  amin   amax
         Churn
         0       0.0  361.8          200.133193  51.105032  23.2  395.0
         1      70.9  363.7          205.231677  47.132825  47.4  354.9
```

1.5 Summary tables

Suppose we want to see how the observations in our sample are distributed in the context of two variables - Churn and International plan. To do so, we can build a **contingency table** using the crosstab method:

```
In [26]: pd.crosstab(df['Churn'], df['International plan'])
```

```
Out[26]: International plan  False  True
         Churn
         0                    2664   186
         1                     346   137
```

```
In [27]: pd.crosstab(df['Churn'], df['Voice mail plan'], normalize=True)
```

```
Out[27]: Voice mail plan     False     True
         Churn
         0                0.602460  0.252625
         1                0.120912  0.024002
```

We can see that most of the users are loyal and do not use additional services (International Plan/Voice mail).

This will resemble **pivot tables** to those familiar with Excel. And, of course, pivot tables are implemented in Pandas: the `pivot_table` method takes the following parameters:

- `values` – a list of variables to calculate statistics for,
- `index` – a list of variables to group data by,
- `aggfunc` – what statistics we need to calculate for groups, ex. sum, mean, maximum, minimum or something else.

Let's take a look at the average number of day, evening, and night calls by area code:

```
In [28]: df.pivot_table(['Total day calls', 'Total eve calls', 'Total night calls'],
                        ['Area code'], aggfunc='mean')
```

```
Out[28]:            Total day calls  Total eve calls  Total night calls
         Area code
         408             100.496420        99.788783          99.039379
         415             100.576435       100.503927         100.398187
         510             100.097619        99.671429         100.601190
```

1.6 DataFrame transformations

Like many other things in Pandas, adding columns to a DataFrame is doable in many ways.

For example, if we want to calculate the total number of calls for all users, let's create the `total_calls` Series and paste it into the DataFrame:

```
In [29]: total_calls = df['Total day calls'] + df['Total eve calls'] + \
                       df['Total night calls'] + df['Total intl calls']
         df.insert(loc=len(df.columns), column='Total calls', value=total_calls)
         # loc parameter is the number of columns after which to insert the Series object
         # we set it to len(df.columns) to paste it at the very end of the dataframe
         df.head()
```

```
Out[29]:   State  Account length  Area code  International plan  Voice mail plan  \
        0    KS             128        415               False             True
        1    OH             107        415               False             True
        2    NJ             137        415               False            False
        3    OH              84        408                True            False
        4    OK              75        415                True            False

           Number vmail messages  Total day minutes  Total day calls  \
        0                     25              265.1              110
        1                     26              161.6              123
        2                      0              243.4              114
        3                      0              299.4               71
        4                      0              166.7              113

           Total day charge  Total eve minutes     ...       Total eve charge  \
        0             45.07              197.4     ...                  16.78
        1             27.47              195.5     ...                  16.62
        2             41.38              121.2     ...                  10.30
        3             50.90               61.9     ...                   5.26
        4             28.34              148.3     ...                  12.61

           Total night minutes  Total night calls  Total night charge  \
        0                244.7                 91               11.01
        1                254.4                103               11.45
        2                162.6                104                7.32
        3                196.9                 89                8.86
        4                186.9                121                8.41

           Total intl minutes  Total intl calls  Total intl charge  \
        0                10.0                 3               2.70
        1                13.7                 3               3.70
        2                12.2                 5               3.29
        3                 6.6                 7               1.78
        4                10.1                 3               2.73

           Customer service calls  Churn  Total calls
        0                       1      0          303
        1                       1      0          332
        2                       0      0          333
        3                       2      0          255
        4                       3      0          359

        [5 rows x 21 columns]
```

It is possible to add a column more easily without creating an intermediate Series instance:

```
In [30]: df['Total charge'] = df['Total day charge'] + df['Total eve charge'] + \
                              df['Total night charge'] + df['Total intl charge']
         df.head()
```

```
Out[30]:   State  Account length  Area code  International plan  Voice mail plan  \
        0    KS             128        415               False             True
        1    OH             107        415               False             True
        2    NJ             137        415               False            False
        3    OH              84        408                True            False
        4    OK              75        415                True            False

           Number vmail messages  Total day minutes  Total day calls  \
        0                     25              265.1              110
        1                     26              161.6              123
        2                      0              243.4              114
        3                      0              299.4               71
        4                      0              166.7              113

           Total day charge  Total eve minutes     ...       Total night minutes  \
        0             45.07              197.4     ...                     244.7
        1             27.47              195.5     ...                     254.4
        2             41.38              121.2     ...                     162.6
        3             50.90               61.9     ...                     196.9
        4             28.34              148.3     ...                     186.9

           Total night calls  Total night charge  Total intl minutes  \
        0                 91               11.01                10.0
        1                103               11.45                13.7
        2                104                7.32                12.2
        3                 89                8.86                 6.6
        4                121                8.41                10.1

           Total intl calls  Total intl charge  Customer service calls  Churn  \
        0                 3               2.70                       1      0
        1                 3               3.70                       1      0
        2                 5               3.29                       0      0
        3                 7               1.78                       2      0
        4                 3               2.73                       3      0

           Total calls  Total charge
        0          303         75.56
        1          332         59.24
        2          333         62.29
        3          255         66.80
        4          359         52.09

        [5 rows x 22 columns]
```

To delete columns or rows, use the `drop` method, passing the required indexes and the `axis` parameter (`1` if you delete columns, and nothing or `0` if you delete rows). The `inplace` argument tells whether to change the original DataFrame. With `inplace=False`, the `drop` method doesn't change the existing DataFrame and returns a new one with dropped rows or columns. With inplace=True, it alters the DataFrame.

```
In [31]: # get rid of just created columns
         df.drop(['Total charge', 'Total calls'], axis=1, inplace=True)
         # and here's how you can delete rows
         df.drop([1, 2]).head()
```

```
Out[31]:   State  Account length  Area code  International plan  Voice mail plan  \
         0    KS             128        415               False             True
         3    OH              84        408                True            False
         4    OK              75        415                True            False
         5    AL             118        510                True            False
         6    MA             121        510               False             True

            Number vmail messages  Total day minutes  Total day calls  \
         0                     25              265.1              110
         3                      0              299.4               71
         4                      0              166.7              113
         5                      0              223.4               98
         6                     24              218.2               88

            Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
         0             45.07              197.4               99             16.78
         3             50.90               61.9               88              5.26
         4             28.34              148.3              122             12.61
         5             37.98              220.6              101             18.75
         6             37.09              348.5              108             29.62

            Total night minutes  Total night calls  Total night charge  \
         0                244.7                 91               11.01
         3                196.9                 89                8.86
         4                186.9                121                8.41
         5                203.9                118                9.18
         6                212.6                118                9.57

            Total intl minutes  Total intl calls  Total intl charge  \
         0                10.0                 3               2.70
         3                 6.6                 7               1.78
         4                10.1                 3               2.73
         5                 6.3                 6               1.70
         6                 7.5                 7               2.03

            Customer service calls  Churn
         0                       1      0
         3                       2      0
         4                       3      0
         5                       0      0
         6                       3      0
```

2 First attempt at predicting telecom churn

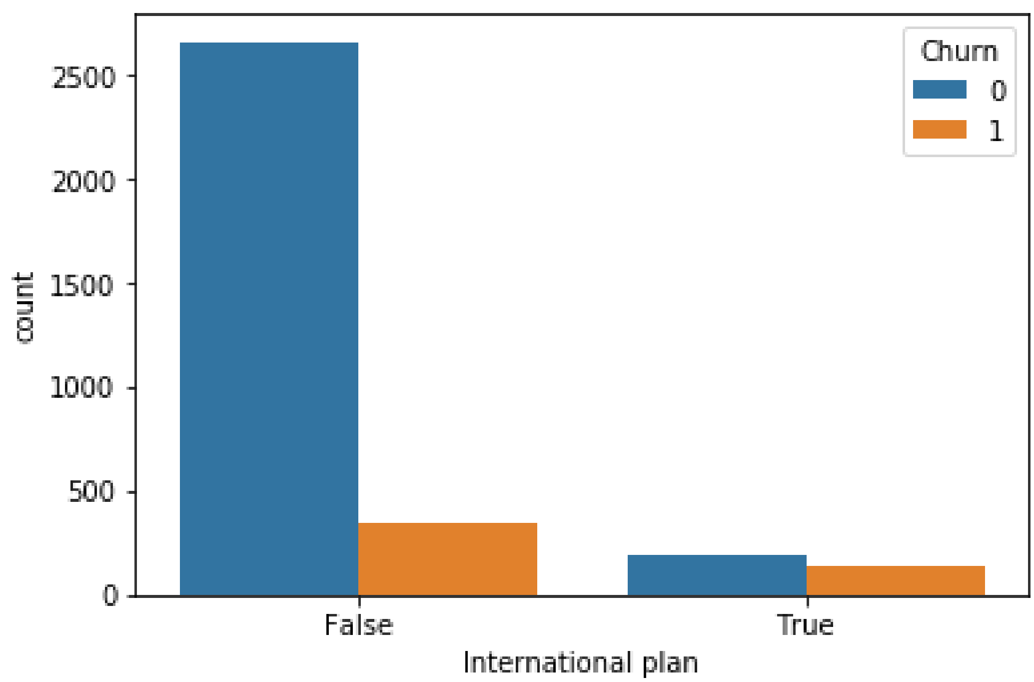
Let's see how churn rate is related to the *International plan* feature. We'll do this using a `crosstab` contingency table and also through visual analysis with `Seaborn` (however, visual analysis will be covered more thoroughly in the next article).

```
In [32]: pd.crosstab(df['Churn'], df['International plan'], margins=True)
```

```
Out[32]: International plan  False  True   All
         Churn
         0                    2664   186  2850
         1                     346   137   483
         All                  3010   323  3333
```

```
In [33]: # some imports to set up plotting
         import matplotlib.pyplot as plt
         # pip install seaborn
         import seaborn as sns
```

```
In [34]: sns.countplot(x='International plan', hue='Churn', data=df);
```



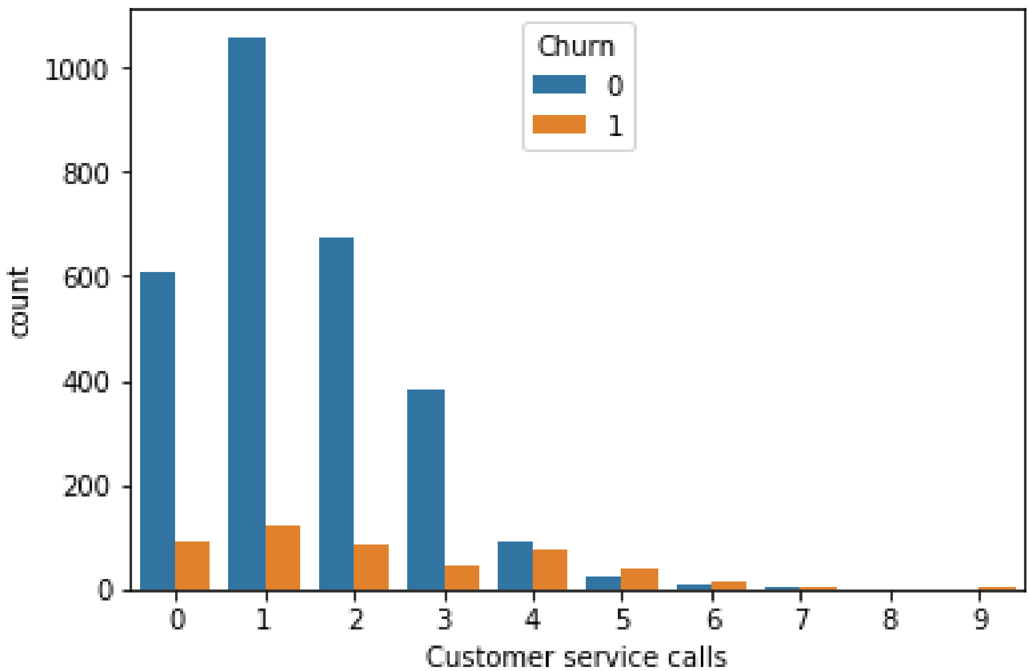
We see that, with *International Plan*, the churn rate is much higher, which is an interesting observation! Perhaps large and poorly controlled expenses with international calls are very conflict-prone and lead to dissatisfaction among the telecom operator's customers.

Next, let's look at another important feature – *Customer service calls*. Let's also make a summary table and a picture.

```
In [35]: pd.crosstab(df['Churn'], df['Customer service calls'], margins=True)
```

```
Out[35]: Customer service calls    0     1    2    3    4   5   6  7  8  9   All
         Churn
         0                       605  1059  672  385   90  26   8  4  1  0  2850
         1                        92   122   87   44   76  40  14  5  1  2   483
         All                     697  1181  759  429  166  66  22  9  2  2  3333
```

```
In [36]: sns.countplot(x='Customer service calls', hue='Churn', data=df);
```



Although it's not so obvious from the summary table, it's easy to see from the above plot that the churn rate increases sharply from 4 customer service calls and above.

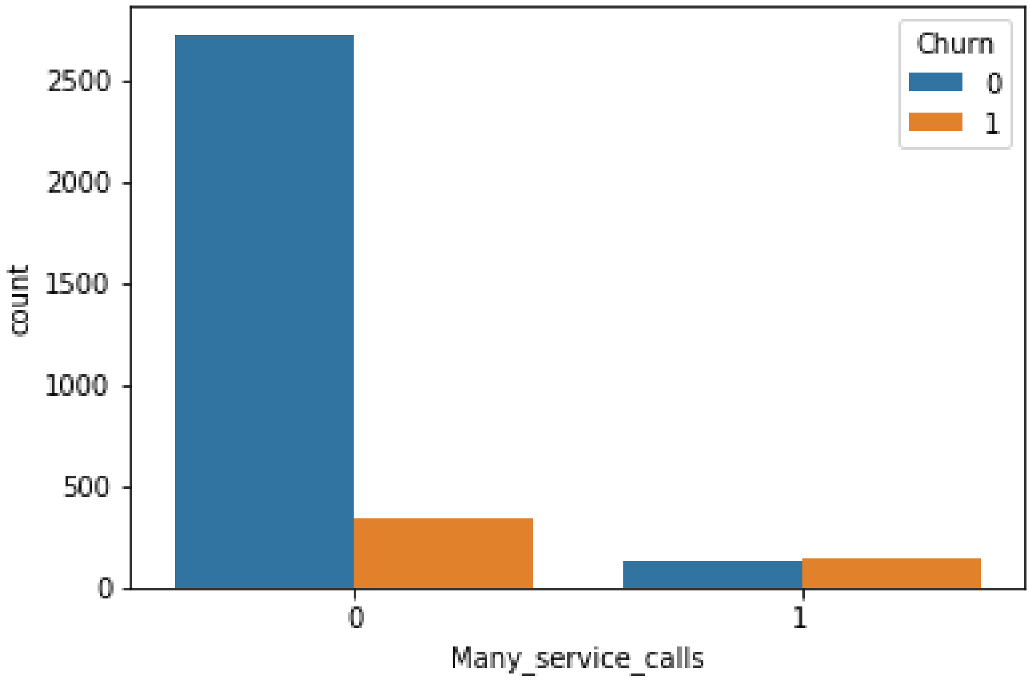
Now let's add a binary feature to our DataFrame – `Customer service calls > 3`. And once again, let's see how it relates to churn.

```
In [37]: df['Many_service_calls'] = (df['Customer service calls'] > 3).astype('int')

         pd.crosstab(df['Many_service_calls'], df['Churn'], margins=True)
```

```
Out[37]: Churn                  0    1   All
         Many_service_calls
         0                   2721  345  3066
         1                    129  138   267
         All                 2850  483  3333
```

```
In [38]: sns.countplot(x='Many_service_calls', hue='Churn', data=df);
```



Let's construct another contingency table that relates *Churn* with both *International plan* and freshly created *Many_service_calls*.

```
In [39]: pd.crosstab(df['Many_service_calls'] & df['International plan'] ,
                               df['Churn'])
```

```
Out[39]: Churn     0    1
         row_0
         False  2841  464
         True      9   19
```

Therefore, predicting that a customer is not loyal (*Churn*=1) in the case when the number of calls to the service center is greater than 3 and the *International Plan* is added (and predicting *Churn*=0 otherwise), we might expect an accuracy of 85.8% (we are mistaken only 464 + 9 times). This number, 85.8%, that we got through this very simple reasoning serves as a good starting point (*baseline*) for the further machine learning models that we will build.

As we move on in this course, recall that, before the advent of machine learning, the data analysis process looked something like this. Let's recap what we've covered:

- The share of loyal clients in the sample is 85.5%. The most naive model that always predicts a "loyal customer" on such data will guess right in about 85.5% of all cases. That is, the proportion of correct answers (*accuracy*) of subsequent models should be no less than this number, and will hopefully be significantly higher;
- With the help of a simple forecast that can be expressed by the following formula: "International plan = True & Customer Service calls > 3 => Churn = 1, else Churn = 0", we can expect a guessing rate of 85.8%, which is just above 85.5%. Subsequently, we'll talk about decision trees and figure out how to find such rules **automatically** based only on the input data;

- We got these two baselines without applying machine learning, and they'll serve as the starting point for our subsequent models. If it turns out that with enormous effort, we increase the share of correct answers by 0.5% per se, then possibly we are doing something wrong, and it suffices to confine ourselves to a simple model with two conditions;
- Before training complex models, it is recommended to manipulate the data a bit, make some plots, and check simple assumptions. Moreover, in business applications of machine learning, they usually start with simple solutions and then experiment with more complex ones.

3 Demo assignment

To practice with Pandas and EDA, you can complete this assignment where you'll be analyzing socio-demographic data.

4 Useful resources

- The same notebook as an interactive web-based Kaggle Kernel
- "Merging DataFrames with pandas" - a tutorial by Max Plako within mlcourse.ai (full list of tutorials is here)
- "Handle different dataset with dask and trying a little dask ML" - a tutorial by Irina Knyazeva within mlcourse.ai
- Main course site, course repo, and YouTube channel
- Official Pandas documentation
- Course materials as a Kaggle Dataset
- Medium "story" based on this notebook
- If you read Russian: an article on Habr.com with ~ the same material. And a lecture on YouTube
- 10 minutes to pandas
- Pandas cheatsheet PDF
- GitHub repos: Pandas exercises and "Effective Pandas"
- scipy-lectures.org — tutorials on pandas, numpy, matplotlib and scikit-learn