Exploratory Data Analysis Starter

Import packages

```
# load packages
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', 50)

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
```

Loading data with Pandas

We need to load `client_data.csv` and `price_data.csv` into individual dataframes so that we can work with them in Python. For this notebook and all further notebooks, it will be assumed that the CSV files will the placed in the same file location as the notebook. If they are not, please adjust the directory within the `read_csv` method accordingly.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
client = pd.read_csv('/content/drive/MyDrive/bcg_data/client_data.csv')
price = pd.read_csv('/content/drive/MyDrive/bcg_data/price_data.csv')
```

You can view the first 3 rows of a dataframe using the `head` method. Similarly, if you wanted to see the last 3, you can use `tail(3)`

```
client.head(3)
```

	id	channel_sales	cons_12m	cons_gas
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	5
1	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	
2	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	

```
price.head(3)
```

	id	price_date	price_off_peak_var	price_peak_var	pri
0	038af19179925da21a25619c5a24b745	01-01-2015	0.151367	0.0	
1	038af19179925da21a25619c5a24b745	01-02-2015	0.151367	0.0	
2	038af19179925da21a25619c5a24b745	01-03-2015	0.151367	0.0	

Descriptive statistics of data

Data types

It is useful to first understand the data that you're dealing with along with the data types of each column. The data types may dictate how you transform and engineer features.

To get an overview of the data types within a data frame, use the `info()` method.

```
client.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10831 entries, 0 to 10830
Data columns (total 26 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   id                               10831 non-null  object 
 1   channel_sales                    10831 non-null  object 
 2   cons_12m                         10831 non-null  int64  
 3   cons_gas_12m                     10831 non-null  int64  
 4   cons_last_month                  10831 non-null  int64  
 5   date_activ                       10831 non-null  object 
 6   date_end                         10831 non-null  object 
 7   date_modif_prod                  10831 non-null  object 
 8   date_renewal                     10831 non-null  object 
 9   forecast_cons_12m                10831 non-null  float64
 10  forecast_cons_year               10831 non-null  int64  
 11  forecast_discount_energy         10831 non-null  int64  
 12  forecast_meter_rent_12m          10831 non-null  float64
 13  forecast_price_energy_off_peak   10831 non-null  float64
 14  forecast_price_energy_peak       10831 non-null  float64
 15  forecast_price_pow_off_peak      10831 non-null  float64
 16  has_gas                          10831 non-null  object 
 17  imp_cons                         10831 non-null  float64
 18  margin_gross_pow_ele             10831 non-null  float64
 19  margin_net_pow_ele               10831 non-null  float64
 20  nb_prod_act                      10831 non-null  int64  
 21  net_margin                       10831 non-null  float64
 22  num_years_antig                  10831 non-null  int64  
 23  origin_up                        10831 non-null  object 
 24  pow_max                          10831 non-null  float64
 25  churn                            10831 non-null  int64  
dtypes: float64(10), int64(8), object(8)
memory usage: 2.1+ MB
```

```
price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  193002 non-null  object 
 1   price_date          193002 non-null  object 
 2   price_off_peak_var  193002 non-null  float64
 3   price_peak_var      193002 non-null  float64
 4   price_mid_peak_var  193002 non-null  float64
 5   price_off_peak_fix  193002 non-null  float64
 6   price_peak_fix      193002 non-null  float64
 7   price_mid_peak_fix  193002 non-null  float64
 8   Unnamed: 8          0 non-null       float64
 9   Unnamed: 9          0 non-null       float64
 10  Unnamed: 10         1 non-null       float64
dtypes: float64(9), object(2)
memory usage: 16.2+ MB
```

```
price.columns
```

```
Index(['id', 'price_date', 'price_off_peak_var', 'price_peak_var',
       'price_mid_peak_var', 'price_off_peak_fix', 'price_peak_fix',
       'price_mid_peak_fix', 'Unnamed: 8', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

```
columns_to_drop = ['Unnamed: 8', 'Unnamed: 9', 'Unnamed: 10']
price_df = price.drop(columns=columns_to_drop)
```

```
price.columns
```

```
Index(['id', 'price_date', 'price_off_peak_var', 'price_peak_var',
       'price_mid_peak_var', 'price_off_peak_fix', 'price_peak_fix',
       'price_mid_peak_fix', 'Unnamed: 8', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

Statistics

Now let's look at some statistics about the datasets. We can do this by using the `describe()` method.

```
client.describe()
```

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_yea
count	1.083100e+04	1.083100e+04	10831.000000	10831.000000	10831.00000
mean	1.752256e+05	2.846208e+04	17506.815345	1910.533724	1392.07441
std	6.129525e+05	1.702019e+05	68178.552846	2377.560705	2859.24457
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000
25%	5.893000e+03	0.000000e+00	0.000000	519.435000	0.00000
50%	1.496300e+04	0.000000e+00	847.000000	1161.460000	341.00000
75%	4.420900e+04	0.000000e+00	3632.500000	2480.365000	1758.00000
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	79127.00000

```
price.describe()
```

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price
count	193002.000000	193002.000000	193002.000000	193002.000000	193
mean	0.141027	0.054630	0.030496	43.334477	
std	0.025032	0.049924	0.036298	5.410297	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.125976	0.000000	0.000000	40.728885	
50%	0.146033	0.085483	0.000000	44.266930	
75%	0.151635	0.101673	0.072558	44.444710	

Data visualization

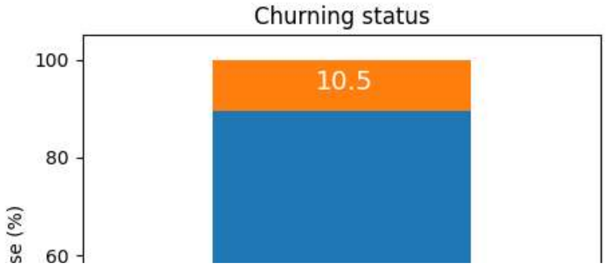
If you're working in Python, two of the most popular packages for visualization are `matplotlib` and `seaborn`. We highly recommend you use these, or at least be familiar with them because they are ubiquitous!

Below are some functions that you can use to get started with visualizations.

```
def plot_stacked_bars(dataframe, title_, size_=(18, 10), rot_=0, legend_="upper right"):
    """
    Plot stacked bars with annotations
    """
    ax = dataframe.plot(
        kind="bar",
        stacked=True,
        figsize=size_,
        rot=rot_,
        title=title_
    )

    # Annotate bars
    annotate_stacked_bars(ax, textsize=14)
    # Rename legend
    plt.legend(["Retention", "Churn"], loc=legend_)
    # Labels
    plt.ylabel("Company base (%)")
    plt.show()

def annotate_stacked_bars(ax, pad=0.99, colour="white", textsize=13):
    """
    Add value annotations to the bars
    """

    # Iterate over the plotted rectanges/bars
    for p in ax.patches:

        # Calculate annotation
        value = str(round(p.get_height(),1))
        # If value is 0 do not annotate
        if value == '0.0':
            continue
        ax.annotate(
            value,
            ((p.get_x()+ p.get_width()/2)*pad-0.05, (p.get_y()+p.get_height()/2)*pad),
            color=colour,
            size=textsize
        )

def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distirbution in a stacked histogram of churned or retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "Churn":dataframe[dataframe["churn"]==1][column]})
    # Plot the histogram
    temp[["Retention","Churn"]].plot(kind='hist', bins=bins_, ax=ax, stacked=True)
    # X-axis label
    ax.set_xlabel(column)
    # Change the x-axis to plain style
    ax.ticklabel_format(style='plain', axis='x')
```

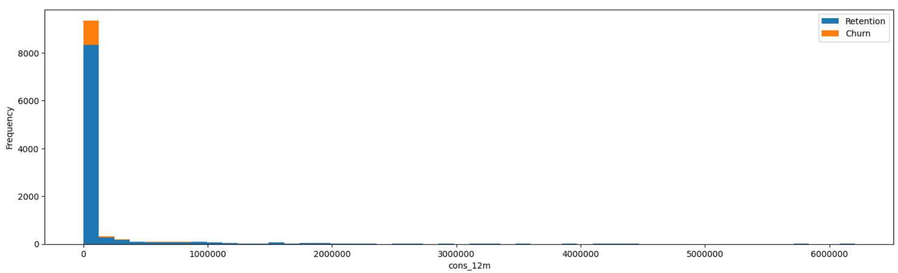
Thhe first function `plot_stacked_bars` is used to plot a stacked bar chart. An example of how you could use this is shown below:

```
churn = client[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5), legend_="lower right")
```



The second function `annotate_bars` is used by the first function, but the third function `plot_distribution` helps you to plot the distribution of a numeric column. An example of how it can be used is given below:

```
consumption = client[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_cons', 'has_gas', 'churn']]

fig, axs = plt.subplots(nrows=1, figsize=(18, 5))

plot_distribution(consumption, 'cons_12m', axs)
```




```
for f in ['date_activ','date_end','date_modif_prod','date_renewal']:
    client[f] = pd.to_datetime(client[f])
```

```
<ipython-input-104-34818b323dd1>:2: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. Thi
  client[f] = pd.to_datetime(client[f])
<ipython-input-104-34818b323dd1>:2: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. Thi
  client[f] = pd.to_datetime(client[f])
<ipython-input-104-34818b323dd1>:2: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. Thi
  client[f] = pd.to_datetime(client[f])
<ipython-input-104-34818b323dd1>:2: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. Thi
  client[f] = pd.to_datetime(client[f])
```

```
client['contract_start_year'] = client['date_activ'].dt.year
client['contract_end_year'] = client['date_end'].dt.year
```

```
# define a function to display missing values and dupliate rows
def duplicate_and_missing(dataset, dataset_name):
    print('There are', dataset.shape[0], 'rows and', dataset.shape[1], 'columns in the dataset', '"'+dataset_name+'"','\n'+'--'*40)
    # display missing values
    if dataset.isna().sum().sum()!=0:  # if there is missing values
        missing_value = dataset.isna().sum()[dataset.isna().sum()!=0].to_frame(name='count')
        missing_value['proportion'] = missing_value['count']/len(dataset)
        print('There are', dataset.isna().sum().sum(), 'missing values')
        print(missing_value, '\n'+'--'*40)
    else:
        print('There is no missing value')
    # display duplicate rows
    if dataset.duplicated().sum()!=0:
        print('There are', dataset.duplicated().sum(), 'duplicate rows\n')
    else:
        print('There is no duplicate row\n')


# def find_missing_values(data):
#     # Check for null values
#     null_values = data.isnull().sum()

#     # Check for missing values (if numeric values are represented as something other than NaN)
#     missing_values = data.isna().sum()

#     # Combine null and missing values
#     missing_data = pd.DataFrame({'Null_Values': null_values, 'Missing_Values': missing_values})

#     # Display columns with missing or null values, if any
#     if missing_data[missing_data.sum(axis=1) > 0].empty:
#         print("No missing or null values found.")
#     else:
#         print("Columns with missing or null values:")
#         print(missing_data[missing_data.sum(axis=1) > 0])

# # Example usage:
# # Assuming 'your_data' is your DataFrame
# find_missing_values(client)
```

```
No missing or null values found.
```

```
duplicate_and_missing(dataset=client, dataset_name='Client')
```

```
There are 10831 rows and 28 columns in the dataset "Client"
--------------------------------------------------------------------------------
There is no missing value
There is no duplicate row
```

```
def describe_categorical(dataset):
    cat_columns = dataset.dtypes[dataset.dtypes=='object'].index.tolist()
    if len(cat_columns)!=0:
        print('Categorical variables are', cat_columns,'\n'+'=='*40)
        for cat in cat_columns:
            describe_frame = dataset[[cat]].value_counts().reset_index(name='count')
            describe_frame['proportion'] = describe_frame['count']/len(dataset)
            print(describe_frame,'\n'+'--'*40)    # display value count and proportion of a categorical feature
    else:
        print('There is no categorical variables in the dataset')


# def describe_categorical(dataset):
#     # Identify categorical columns
#     cat_columns = dataset.select_dtypes(include=['object']).columns.tolist()

#     if len(cat_columns) > 0:
#         print("Categorical columns and their count/proportion:")
#         print("=" * 50)

#         for col in cat_columns:
#             # Calculate value counts and proportions
#             value_counts = dataset[col].value_counts()
#             proportions = dataset[col].value_counts(normalize=True)

#             # Display count and proportion for each unique value in the categorical column
#             print(f"Column: {col}")
```

```
    #             print("-" * 30)
    #             for idx, val in value_counts.items():
    #                 proportion = proportions[idx]
    #                 print(f"Value: {idx}, Count: {val}, Proportion: {proportion:.4f}")
    #             print("-" * 30)
    #     else:
    #         print("No categorical columns found in the dataset.")

    # # Example usage:
    # # Assuming 'your_dataset' is your DataFrame
    # describe_categorical(client)
```

```
Streaming output truncated to the last 5000 lines.
Value: 87c262697cdf4db281e424e5c6d0f530, Count: 1, Proportion: 0.0001
Value: fd1e359c5613bb74ea1e18836f2ab9a2, Count: 1, Proportion: 0.0001
Value: 04db5cdec5c42ac9c95aa9de51e3741a, Count: 1, Proportion: 0.0001
Value: 8f7525faa5f8c20b8936b7aca39999ea, Count: 1, Proportion: 0.0001
Value: eecc3978c6e30f08aec6d6c7bfa0e8ae, Count: 1, Proportion: 0.0001
Value: 8e371b22a33ba25687ebef5bb72faa32, Count: 1, Proportion: 0.0001
Value: 94c2ce561ee6357d407e70b17c43176c, Count: 1, Proportion: 0.0001
Value: a85d5cff83c992855c198c1efc5690c4, Count: 1, Proportion: 0.0001
Value: 7361214ac450e48964193c72385c7178, Count: 1, Proportion: 0.0001
Value: 2f52ef4f444bb56552c75ad3cb4385f6, Count: 1, Proportion: 0.0001
Value: 0cd06272ce3e701267ebe2766def23b4, Count: 1, Proportion: 0.0001
Value: fda62c7fca8f550919eed3b09637ed74, Count: 1, Proportion: 0.0001
Value: e8cb1e91f5c53638adf3659e92149584, Count: 1, Proportion: 0.0001
Value: b28e525de5e0060830978f9a6c9574cd, Count: 1, Proportion: 0.0001
Value: 647939d8dac011b77e67f88ef8b5ed35, Count: 1, Proportion: 0.0001
Value: 3c8a93992b802c6daa789d20a4831d6a, Count: 1, Proportion: 0.0001
Value: fc57cf42a8443934a56e07da5fb7648a, Count: 1, Proportion: 0.0001
Value: 5ba1acc3bd0d6d82fb44a494df3e2ceb, Count: 1, Proportion: 0.0001
Value: fa8b76dbc6cf2f932b05e6593a7b724e, Count: 1, Proportion: 0.0001
Value: f5ed37cb875d0ca140113c16fdac2e24, Count: 1, Proportion: 0.0001
Value: b36e4ee364d2ee7ff5acdeba893e585c, Count: 1, Proportion: 0.0001
Value: 1cac58bc04a00fec12634864fc4ee065, Count: 1, Proportion: 0.0001
Value: d39f03d0ec9d0bd973928ed30f8f632e, Count: 1, Proportion: 0.0001
Value: 63c81fc053bc3172d6ff61553788731d, Count: 1, Proportion: 0.0001
Value: 0b4d2eff71669eb07dc77186bce69d3e, Count: 1, Proportion: 0.0001
Value: 100f634233404e518820f8ff2c351896, Count: 1, Proportion: 0.0001
Value: 969aa0938dcc5a22b997001130051fbb, Count: 1, Proportion: 0.0001
Value: b68f785a9d3c647e1e5fbf873580f107, Count: 1, Proportion: 0.0001
Value: a1941a9aa5358b504a4abbcfbb625c51, Count: 1, Proportion: 0.0001
Value: 89454dbf3683a25441567b581ec04eab, Count: 1, Proportion: 0.0001
Value: 9bbf539970aad4040ac3edd3f982276d, Count: 1, Proportion: 0.0001
Value: 3bbbdff45250c760abf8bc33f92fb819, Count: 1, Proportion: 0.0001
Value: cdc1a135345d0ffea9859a5f9429e065, Count: 1, Proportion: 0.0001
Value: 66d0ec1ae6722766c66cb5435279c2cf, Count: 1, Proportion: 0.0001
Value: 9665400752ccb3e14f08268109eb3dff, Count: 1, Proportion: 0.0001
Value: 0fc0574e030da230810095b10843087f, Count: 1, Proportion: 0.0001
Value: 24403e91cbc9e4b9a5dfb073b40dcb6b, Count: 1, Proportion: 0.0001
Value: 2677df984c48b18aaca210df8f7aca54, Count: 1, Proportion: 0.0001
Value: 3e5b4b4a776f2afe1cb2067cdd41cce1, Count: 1, Proportion: 0.0001
Value: a713ba19c912bcf4cbbc9e6de2b7bd34, Count: 1, Proportion: 0.0001
Value: e5c6e21507bdf7bb4b275b081b132894, Count: 1, Proportion: 0.0001
Value: b5193b3b8fe8f0e5103bb0b778e62822, Count: 1, Proportion: 0.0001
Value: 3a328901da5ee4ede4262ed9b0f446bb, Count: 1, Proportion: 0.0001
Value: a020b85d573fa4fb1c2d11f972fe57e4, Count: 1, Proportion: 0.0001
Value: e3850a17dd817771d9002baa02cc3c17, Count: 1, Proportion: 0.0001
Value: 86d4465c867fa313996e82950cb1bb83, Count: 1, Proportion: 0.0001
Value: 19fb674c18f6624401706cb5cd034f5e, Count: 1, Proportion: 0.0001
Value: 29671600cba9fe1c62dbe53ef72c8694, Count: 1, Proportion: 0.0001
Value: b3947c212d6ea4234ee07ce7a872ebf4, Count: 1, Proportion: 0.0001
Value: 1674482df4eff7ec5d5f7ce52750985c, Count: 1, Proportion: 0.0001
Value: e60cfe7c435da332915404f3d56517a2, Count: 1, Proportion: 0.0001
Value: 6931b246e104138a38e24ae99e7867cc, Count: 1, Proportion: 0.0001
Value: 98208d3b725d012f393385ad92a7ae8b, Count: 1, Proportion: 0.0001
Value: 0fbfb3f7d67f2b30d4874ad011a0728f, Count: 1, Proportion: 0.0001
Value: eb319352b676316bf73f80bc663466da, Count: 1, Proportion: 0.0001
Value: 022999ad5b8b8f6be4860f2a30f644ac, Count: 1, Proportion: 0.0001
Value: 6877d5f1392d274fed1d0d7cbe4c7f4e, Count: 1, Proportion: 0.0001
```

```
describe_categorical(dataset=client)
```

```
Categorical variables are ['id', 'channel_sales', 'has_gas', 'origin_up']
================================================================================
                                    id  count  proportion
0      0002203ffbb812588b632b9e628cc38d      1    0.000092
1      aa7a5787176acd7c6a60c468115abf9f      1    0.000092
2      aa451d8c32c9895ae258f91525e8563c      1    0.000092
3      aa476de625a4b6d8e9a1930f8ef880b2      1    0.000092
4      aa4ebe702fbcc868bf500086ae083ef3      1    0.000092
...                                 ...    ...         ...
10826  55e11bf78679ad926d0143c765f63d2e      1    0.000092
```

```
10827  55ebaf77fe25a424b4c4dda1824c5bad      1    0.000092
10828  55ebdfdb64adaa2becff36ec016384ee      1    0.000092
10829  55f744592eab5a92f44f932b7dd4d26f      1    0.000092
10830  ffff7fa066f1fb305ae285bb03bf325a      1    0.000092

[10831 rows x 3 columns]
--------------------------------------------------------------------------------
                     channel_sales  count  proportion
0  foosdfpfkusacimwkcsosbicdxkicaua   6722    0.620626
1  lmkebamcaaclubfxadlmueccxoimlema   1836    0.169513
2  usilxuppasemubllopkaafesmlibmsdf   1369    0.126396
3  ewpakwlliwisiwduibdlfmalxowmwpci    888    0.081987
4  sddiedcslfslkckwlfkdpoeeailfpeds     11    0.001016
5  epumfxlbckeskwekxbiuasklxalciiuu      3    0.000277
6  fixdbufsefwooaasfcxdxadsiekoceaa      2    0.000185
--------------------------------------------------------------------------------
  has_gas  count  proportion
0       f   8945     0.82587
1       t   1886     0.17413
--------------------------------------------------------------------------------
                          origin_up  count  proportion
0  lxidpiddsbxsbosboudacockeimpuepw   6441    0.594682
1  kamkkxfxxuwbdslkwifmmcsiusiuosws   2748    0.253716
2  ldkssxwpmemidmecebumciepifcamkci   1641    0.151510
3  usapbepcfoloekilkwsdiboslwaxobdp      1    0.000092
--------------------------------------------------------------------------------
```

```
# label encoding
channel_encoder = LabelEncoder()
client['channel_sales'] = channel_encoder.fit_transform(client['channel_sales'])
origin_encoder = LabelEncoder()
client['origin_up'] = origin_encoder.fit_transform(client['origin_up'])


# dataset: a pandas dataframe;
def describe_numeric(dataset):
    # for numeric columns whose values are discrete, display values and their frequency
    int_columns = dataset.dtypes[dataset.dtypes=='int'].index.tolist()
    # for numeric columns whose values are continuous, plot their value distribution
    float_columns = dataset.dtypes[dataset.dtypes=='float'].index.tolist()
    if len(int_columns)+len(float_columns)!=0:
        print('Numeric variables are', int_columns+float_columns,'\n'+'=='*40)
        # integer numeric feature
        if len(int_columns)!=0:
            for cat in int_columns:
                describe_frame = dataset[[cat]].value_counts().reset_index(name='count')
                describe_frame['proportion'] = describe_frame['count']/len(dataset)
                print(describe_frame,'\n'+'--'*40)    # display value count and proportion of a categorical feature
        #
        print(dataset[int_columns + float_columns].describe().loc[['min','max','mean','50%']].T.rename(columns={'50%':'median'}), '\n'+'--'*
    else:
        print('There is no numeric variables in the dataset')


describe_numeric(dataset=client)
```

```
forecast_price_energy_peak        0.0  1.080920e+01      6.049933
forecast_price_pow_off_peak       0.0  5.926638e+01     43.217286
imp_cons                          0.0  9.682890e+03    154.814143
margin_gross_pow_ele              0.0  3.147600e+02     25.309231
margin_net_pow_ele                0.0  3.147600e+02     25.305720
net_margin                        0.0  2.457065e+04    195.353669
pow_max                           3.3  3.200000e+02     18.001682

                                    median
channel_sales                     3.000000
cons_12m                      14963.000000
cons_gas_12m                      0.000000
cons_last_month                 847.000000
forecast_cons_year              341.000000
forecast_discount_energy          0.000000
nb_prod_act                       1.000000
num_years_antig                   4.000000
origin_up                         2.000000
churn                             0.000000
contract_start_year            2011.000000
contract_end_year              2016.000000
forecast_cons_12m              1161.460000
forecast_meter_rent_12m          18.880000
forecast_price_energy_off_peak    0.143166
forecast_price_energy_peak        0.083849
forecast_price_pow_off_peak      44.311378
imp_cons                         42.260000
margin_gross_pow_ele             22.210000
margin_net_pow_ele               22.210000
net_margin                      116.860000
pow_max                          13.856000
--------------------------------------------------------------------------------
```

```
client[client.dtypes[client.dtypes=='datetime64[ns]'].index.tolist()].describe(datetime_is_numeric=True)
```

	date_activ	date_end	date_modif_prod	date_renewal
count	10831	10831	10831	10831
mean	2011-08-07 20:38:18.735112192	2016-07-16 03:25:32.600867840	2013-04-29 12:18:40.782937600	2015-07-20 21:36:24.729018624
min	2005-01-09 00:00:00	2016-01-02 00:00:00	2005-01-09 00:00:00	2013-02-09 00:00:00
25%	2010-08-04 00:00:00	2016-04-06 00:00:00	2011-07-07 00:00:00	2015-04-13 00:00:00
50%	2011-11-08 00:00:00	2016-07-07 00:00:00	2013-09-16 00:00:00	2015-07-24 00:00:00
75%	2012-07-11 00:00:00	2016-10-18 12:00:00	2015-05-22 00:00:00	2015-10-23 00:00:00

```
price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  193002 non-null  object 
 1   price_date          193002 non-null  object 
 2   price_off_peak_var  193002 non-null  float64
 3   price_peak_var      193002 non-null  float64
 4   price_mid_peak_var  193002 non-null  float64
 5   price_off_peak_fix  193002 non-null  float64
 6   price_peak_fix      193002 non-null  float64
 7   price_mid_peak_fix  193002 non-null  float64
 8   Unnamed: 8          0 non-null       float64
 9   Unnamed: 9          0 non-null       float64
 10  Unnamed: 10         1 non-null       float64
dtypes: float64(9), object(2)
memory usage: 16.2+ MB
```

```
price['price_date'] = pd.to_datetime(price['price_date'])
```

```
duplicate_and_missing(dataset=price, dataset_name='Client')
```

```
There are 193002 rows and 11 columns in the dataset "Client"
--------------------------------------------------------------------------------
There are 579005 missing values
              count  proportion
Unnamed: 8   193002    1.000000
```

```
Unnamed: 9   193002    1.000000
Unnamed: 10  193001    0.999995
--------------------------------------------------------------------------------
There is no duplicate row
```

```
describe_categorical(dataset=price)
```

```
Categorical variables are ['id']
================================================================================
                                    id  count  proportion
0      0002203ffbb812588b632b9e628cc38d     12    0.000062
1      ab07311332ad017c071947aa1747bf9e     12    0.000062
2      aab99b3be145225dbff8551b256da9e0     12    0.000062
3      aac9a198cee4c2eb9d975ca21b38f00b     12    0.000062
4      aacb5f6ab9d32c8cc9e58c505b028d24     12    0.000062
...                                 ...    ...         ...
16091  83cf18b07114e495ae8b7fb235e45ee2      8    0.000041
16092  223a98d3832ece78cbf279a194868b54      8    0.000041
16093  bf89f2d8c1b133a134fd93603cb4c947      7    0.000036
16094  c5dcd5c506e565aaabffa29bc1ec0a37      7    0.000036
16095  15b36e47cf04bf151e3f4438d12672e5      7    0.000036

[16096 rows x 3 columns]
--------------------------------------------------------------------------------
```

```
describe_numeric(dataset=price)
```

```
Numeric variables are ['price_off_peak_var', 'price_peak_var', 'price_mid_peak_var', 'price_off_peak_fix', 'price_peak_fix', 'price_mid_
================================================================================
                     min        max       mean     median
price_off_peak_var  0.0   0.280700   0.141027   0.146033
price_peak_var      0.0   0.229788   0.054630   0.085483
price_mid_peak_var  0.0   0.114102   0.030496   0.000000
price_off_peak_fix  0.0  59.444710  43.334477  44.266930
price_peak_fix      0.0  36.490692  10.622875   0.000000
price_mid_peak_fix  0.0  17.458221   6.409984   0.000000
Unnamed: 8          NaN        NaN        NaN        NaN
Unnamed: 9          NaN        NaN        NaN        NaN
Unnamed: 10         1.0   1.000000   1.000000   1.000000
--------------------------------------------------------------------------------
```

```
price['price_date'].describe(datetime_is_numeric=True)
```

```
count                           193002
mean     2015-01-06 12:01:40.276680960
min                2015-01-01 00:00:00
25%                2015-01-04 00:00:00
50%                2015-01-07 00:00:00
75%                2015-01-10 00:00:00
max                2015-01-12 00:00:00
Name: price_date, dtype: object
```

```
stat_ = ['max', 'min', 'mean']
price_attr = ['price_off_peak_var','price_peak_var','price_mid_peak_var','price_off_peak_fix','price_peak_fix','price_mid_peak_fix']
price_stat = price.drop(columns=['price_date']).groupby(['id']).agg({'price_off_peak_var': stat_, 'price_peak_var': stat_, 'price_mid_peak_v
                                                                     'price_off_peak_fix': stat_, 'price_peak_fix': stat_, 'price_mid_peak_fix': stat_,})
# flatten the column names
price_stat.columns = ['_'.join(x) for x in zip(price_stat.columns.get_level_values(0), price_stat.columns.get_level_values(1))]
price_stat = price_stat.reset_index()
price_stat.head(3)
```

	id	price_off_peak_var_max	price_off_peak_var_min	pr
0	0002203ffbb812588b632b9e628cc38d	0.128067	0.119906	
1	0004351ebdd665e6ee664792efc4fd13	0.148405	0.143943	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.205742	0.150837	

```
# add churn values
price_stat = price_stat.merge(client[['id','churn']], on=['id'], how='left')
# drop ids that are not inclued in the price dataset
price_stat = price_stat.dropna(subset=['churn']).reset_index(drop=True)
price_stat.head(3)
```

	id	price_off_peak_var_max	price_off_peak_var_min	pr
0	0002203ffbb812588b632b9e628cc38d	0.128067	0.119906	
1	0010bcc39e42b3c2131ed2ce55246e3c	0.205742	0.150837	
2	00114d74e963e47177db89bc70108537	0.149902	0.145440	

```
for attr in price_attr:
    price_stat[f'diff_max_min_{attr}'] = price_stat[f'{attr}_max'] - price_stat[f'{attr}_min']
#
price_stat.head(3)
```

	id	price_off_peak_var_max	price_off_peak_var_min	pr
0	0002203ffbb812588b632b9e628cc38d	0.128067	0.119906	
1	0010bcc39e42b3c2131ed2ce55246e3c	0.205742	0.150837	
2	00114d74e963e47177db89bc70108537	0.149902	0.145440	

```
# add diff
for attr in price_attr:
    price_stat[f'diff_Dec_mean_{attr}'] = price[price['id'].isin(price_stat['id'])].groupby(['id'])[attr].nth(-1).values - price_stat[f'{att
#
price_stat.head(3)
```

	id	price_off_peak_var_max	price_off_peak_var_min	pr
0	0002203ffbb812588b632b9e628cc38d	0.128067	0.119906	
1	0010bcc39e42b3c2131ed2ce55246e3c	0.205742	0.150837	
2	00114d74e963e47177db89bc70108537	0.149902	0.145440	

Double-click (or enter) to edit

Double-click (or enter) to edit

```
# load packages
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report
```

```
# drop useless columns
train_data = client.copy()
train_data['year_modif_prod'] = train_data['date_modif_prod'].dt.year
train_data['year_renewal'] = train_data['date_renewal'].dt.year
train_data = train_data.drop(columns=['date_activ','date_end','date_modif_prod','date_renewal'])
#
has_gas_encoder = LabelEncoder()
train_data['has_gas'] = has_gas_encoder.fit_transform(train_data['has_gas'])
# add new feature "diff_dec_jan_off_peak_var" and "diff_dec_jan_off_peak_fix". I just calculate the differnece between last price and first
diff_dec_jan_off_peak_var = price.sort_values(by=['price_date']).groupby(['id'])['price_off_peak_var'].nth(-1) - price.sort_values(by=['pric
diff_dec_jan_off_peak_var = diff_dec_jan_off_peak_var.reset_index(name='diff_dec_jan_off_peak_var')
diff_dec_jan_off_peak_fix = price.sort_values(by=['price_date']).groupby(['id'])['price_off_peak_fix'].nth(-1) - price.sort_values(by=['pric
diff_dec_jan_off_peak_fix = diff_dec_jan_off_peak_fix.reset_index(name='diff_dec_jan_off_peak_fix')
train_data = train_data.merge(diff_dec_jan_off_peak_var, on='id', how='left')
train_data = train_data.merge(diff_dec_jan_off_peak_fix, on='id', how='left')
# also add the above differences of other prices
for attr in ['price_peak_var','price_peak_fix','price_mid_peak_var','price_mid_peak_fix']:
    diff_dec_jan_temp = price.sort_values(by=['price_date']).groupby(['id'])[attr].nth(-1) - price.sort_values(by=['price_date']).groupby(['
    diff_dec_jan_temp = diff_dec_jan_temp.reset_index(name=f'diff_dec_jan_{attr}')
    train_data = train_data.merge(diff_dec_jan_temp, on='id', how='left')
# add price changing trends
train_data = train_data.merge(price_stat[['id','diff_Dec_mean_price_off_peak_var','diff_Dec_mean_price_off_peak_fix',
                                          'diff_Dec_mean_price_peak_var','diff_Dec_mean_price_peak_fix',
                                          'diff_Dec_mean_price_mid_peak_var','diff_Dec_mean_price_mid_peak_fix',
                                         ]], on='id', how='left')

#
train_data.head()
```

rice_mid_peak_fix	diff_Dec_mean_price_off_peak_var	diff_Dec_mean_price_off_peak_fix	di
-16.226389	0.021246	3.324664e+00	
0.000000	-0.002714	5.925960e-02	
0.000000	-0.002624	4.444470e-02	
0.000000	-0.005294	-5.000000e-07	
0.000000	-0.002700	0.000000e+00	

```
X = train_data.drop(columns=['id','churn'])
y = train_data['churn']
X.shape, y.shape
```

```
((10831, 36), (10831,))
```

```
#
pred_train_labels = np.zeros(shape=(X.shape[0], 2))   # pred training labels
feature_importance_df = pd.DataFrame(data={'feature_name':X.columns, 'feature_importance':[0]*len(X.columns)})
# create cv dataset
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=29)
fold_counter = 1
for train_index, test_index in kfold.split(X, y):
    X_train, X_test = X.loc[train_index], X.loc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # build model
    rf = RandomForestClassifier(random_state=56)
    # train model
    rf.fit(X_train, y_train)
    pred_train_labels[test_index] = rf.predict_proba(X_test)
    feature_importance_df['feature_importance'] = feature_importance_df['feature_importance'] + (rf.feature_importances_)
    print(f"Fold {fold_counter} Precision {precision_score(y_test, rf.predict(X_test)):.3f} Recall {recall_score(y_test, rf.predict(X_test))
    fold_counter = fold_counter + 1
# predicted labels
pred_y = pred_train_labels.argmax(axis=-1)
print(f"Total  Precision {precision_score(y, pred_y):.3f} Recall {recall_score(y, pred_y):.3f} Accuracy {accuracy_score(y, pred_y):.3f}")
```

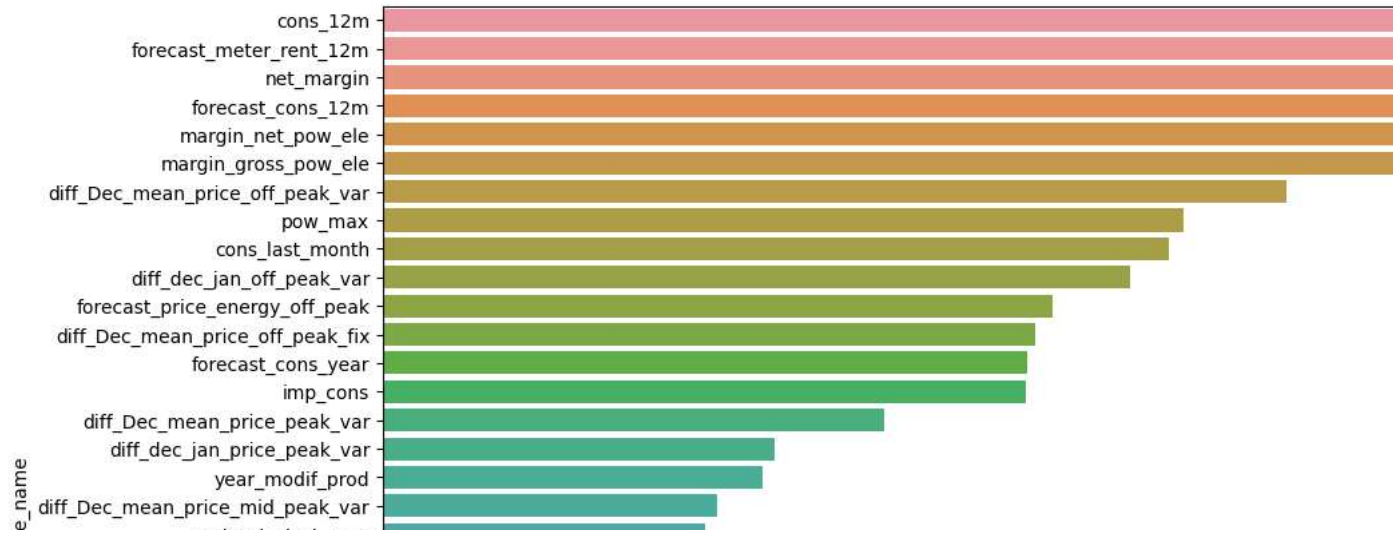
```
Fold 1 Precision 0.941 Recall 0.070 Accuracy 0.902
Fold 2 Precision 0.750 Recall 0.079 Accuracy 0.901
Fold 3 Precision 0.682 Recall 0.066 Accuracy 0.899
Fold 4 Precision 0.833 Recall 0.044 Accuracy 0.899
Fold 5 Precision 0.783 Recall 0.080 Accuracy 0.902
Total  Precision 0.786 Recall 0.068 Accuracy 0.901
```

```
# from sklearn.model_selection import cross_validate
```

```
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.metrics import precision_score, recall_score, accuracy_score
# import numpy as np
# import pandas as pd

# # Assuming X, y are your feature matrix and target vector

# # Initialize arrays and DataFrame
# pred_train_labels = np.zeros((X.shape[0], 2))
# feature_importance = np.zeros(len(X.columns))
# feature_importance_df = pd.DataFrame({'feature_name': X.columns, 'feature_importance': feature_importance})

# # Create cross-validation splits
# kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=29)

# # Create and fit model within cross-validation
# fold_counter = 1
# for train_index, test_index in kfold.split(X, y):
#     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
#     y_train, y_test = y.iloc[train_index], y.iloc[test_index]

#     # Build and train model
#     rf = RandomForestClassifier(random_state=56)
#     rf.fit(X_train, y_train)

#     # Predict probabilities and update feature importance
#     pred_train_labels[test_index] = rf.predict_proba(X_test)
#     feature_importance_df['feature_importance'] += rf.feature_importances_

#     # Evaluate model performance
#     precision = precision_score(y_test, rf.predict(X_test))
#     recall = recall_score(y_test, rf.predict(X_test))
#     accuracy = accuracy_score(y_test, rf.predict(X_test))
#     print(f"Fold {fold_counter} Precision {precision:.3f} Recall {recall:.3f} Accuracy {accuracy:.3f}")
#     fold_counter += 1

# # Calculate overall metrics
# pred_y = pred_train_labels.argmax(axis=-1)
# total_precision = precision_score(y, pred_y)
# total_recall = recall_score(y, pred_y)
# total_accuracy = accuracy_score(y, pred_y)
# print(f"Total Precision {total_precision:.3f} Recall {total_recall:.3f} Accuracy {total_accuracy:.3f}")
```

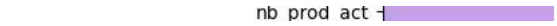
```
Fold 1 Precision 0.941 Recall 0.070 Accuracy 0.902
Fold 2 Precision 0.750 Recall 0.079 Accuracy 0.901
Fold 3 Precision 0.682 Recall 0.066 Accuracy 0.899
Fold 4 Precision 0.833 Recall 0.044 Accuracy 0.899
Fold 5 Precision 0.783 Recall 0.080 Accuracy 0.902
Total Precision 0.786 Recall 0.068 Accuracy 0.901
```

```
fig = plt.figure(figsize=(12,10))
ax = sns.barplot(data=feature_importance_df.sort_values(by=['feature_importance'], ascending=False), y='feature_name', x='feature_importance'
```



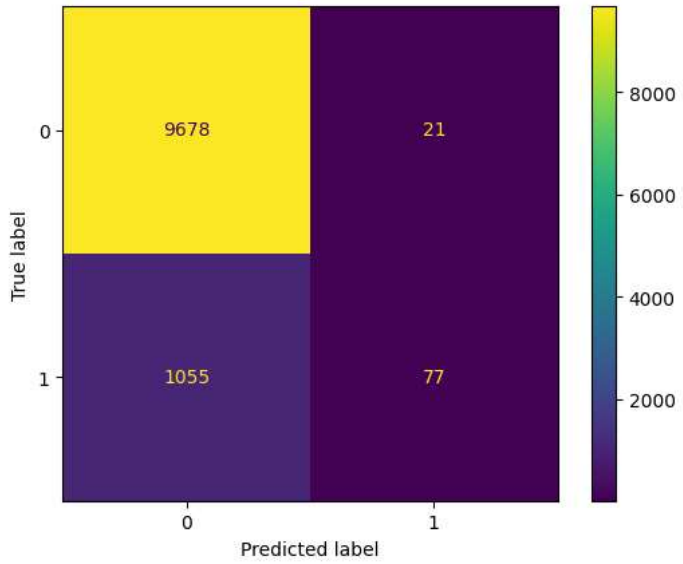

```
print(classification_report(y, pred_y))
```

```
                  precision    recall  f1-score   support

               0       0.90      1.00      0.95      9699
               1       0.79      0.07      0.13      1132

        accuracy                           0.90     10831
       macro avg       0.84      0.53      0.54     10831
    weighted avg       0.89      0.90      0.86     10831
```




```
cm = confusion_matrix(y, pred_y, labels=rf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x79a495b9f490>
```




```
prob_no_discount = pd.DataFrame(data = pred_train_labels, columns=['0', '1'])
prob_no_discount['id'] = train_data['id']
prob_no_discount = prob_no_discount[['id','0','1']]
prob_no_discount
```

	id	0	1
0	24011ae4ebbe3035111d65fa7c15bc57	0.74	0.26
1	764c75f661154dac3a6c254cd082ea7d	0.96	0.04
2	bba03439a292a1e166f80264c16191cb	0.93	0.07
3	1aa498825382410b098937d65c4ec26d	0.83	0.17
4	7ab4bf4878d8f7661dfc20e9b8e18011	0.86	0.14
...	...	...	...
10826	c49217f16a06263e5381eaba94a67a8b	0.80	0.20

```
# drop useless columns
test_data = client.copy()
test_data['year_modif_prod'] = test_data['date_modif_prod'].dt.year
test_data['year_renewal'] = test_data['date_renewal'].dt.year
test_data = test_data.drop(columns=['date_activ','date_end','date_modif_prod','date_renewal'])
#
has_gas_encoder = LabelEncoder()
test_data['has_gas'] = has_gas_encoder.fit_transform(test_data['has_gas'])
# apply a 20% discount
diff_dec_jan_off_peak_var = price.sort_values(by=['price_date']).groupby(['id'])['price_off_peak_var'].nth(-1)*0.8 - price.sort_values(by=['
diff_dec_jan_off_peak_var = diff_dec_jan_off_peak_var.reset_index(name='diff_dec_jan_off_peak_var')
diff_dec_jan_off_peak_fix = price.sort_values(by=['price_date']).groupby(['id'])['price_off_peak_fix'].nth(-1)*0.8 - price.sort_values(by=['
diff_dec_jan_off_peak_fix = diff_dec_jan_off_peak_fix.reset_index(name='diff_dec_jan_off_peak_fix')
test_data = test_data.merge(diff_dec_jan_off_peak_var, on='id', how='left')
test_data = test_data.merge(diff_dec_jan_off_peak_fix, on='id', how='left')
# also add the above differences of other prices
for attr in ['price_peak_var','price_peak_fix','price_mid_peak_var','price_mid_peak_fix']:
    diff_dec_jan_temp = price.sort_values(by=['price_date']).groupby(['id'])[attr].nth(-1)*0.8 - price.sort_values(by=['price_date']).groupb
    diff_dec_jan_temp = diff_dec_jan_temp.reset_index(name=f'diff_dec_jan_{attr}')
    test_data = test_data.merge(diff_dec_jan_temp, on='id', how='left')
# add price changing trends
price_attr = ['price_off_peak_var','price_peak_var','price_mid_peak_var','price_off_peak_fix','price_peak_fix','price_mid_peak_fix']
price_stat_test = price.drop(columns=['price_date']).groupby(['id']).agg({'price_off_peak_var': ['mean'], 'price_peak_var': ['mean'], 'price
                                                                          'price_off_peak_fix': ['mean'], 'price_peak_fix': ['mean'], 'price_mid_peak_fix': ['me
# flatten the column names
price_stat_test.columns = ['_'.join(x) for x in zip(price_stat_test.columns.get_level_values(0), price_stat_test.columns.get_level_values(1)
price_stat_test = price_stat_test.reset_index()
# add diff
for attr in price_attr:
    price_stat_test[f'diff_Dec_mean_{attr}'] = price[price['id'].isin(price_stat_test['id'])].groupby(['id'])[attr].nth(-1).values*0.8 - pri
#
test_data = test_data.merge(price_stat_test[['id','diff_Dec_mean_price_off_peak_var','diff_Dec_mean_price_off_peak_fix',
                                             'diff_Dec_mean_price_peak_var','diff_Dec_mean_price_peak_fix',
                                             'diff_Dec_mean_price_mid_peak_var','diff_Dec_mean_price_mid_peak_fix',
                                           ]], on='id', how='left')

#
test_data.head()
```

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_m
0	24011ae4ebbe3035111d65fa7c15bc57	3	0	54946	
1	764c75f661154dac3a6c254cd082ea7d	3	544	0	
2	bba03439a292a1e166f80264c16191cb	4	1584	0	
3	1aa498825382410b098937d65c4ec26d	6	8302	0	
4	7ab4bf4878d8f7661dfc20e9b8e18011	3	45097	0	

```
#
testing_set = test_data.drop(columns=['id','churn'])
testing_set.shape
```

```
(10831, 36)
```

```
#
pred_train_labels = np.zeros(shape=(X.shape[0], 2))   # pred training labels
pred_test_labels = np.zeros(shape=(testing_set.shape[0], 2))
# create cv dataset
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=29)
fold_counter = 1
for train_index, test_index in kfold.split(X, y):
    X_train, X_test = X.loc[train_index], X.loc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # build model
    rf = RandomForestClassifier(random_state=56)
    # train model
    rf.fit(X_train, y_train)
    pred_train_labels[test_index] = rf.predict_proba(X_test)
    pred_test_labels += rf.predict_proba(testing_set)/5
    print(f"Fold {fold_counter} Precision {precision_score(y_test, rf.predict(X_test)):.3f} Recall {recall_score(y_test, rf.predict(X_test))
    fold_counter = fold_counter + 1
#
print(f"Total  Precision {precision_score(y, pred_y):.3f} Recall {recall_score(y, pred_y):.3f} Accuracy {accuracy_score(y, pred_y):.3f}")
```

```
Fold 1 Precision 0.941 Recall 0.070 Accuracy 0.902
Fold 2 Precision 0.750 Recall 0.079 Accuracy 0.901
Fold 3 Precision 0.682 Recall 0.066 Accuracy 0.899
Fold 4 Precision 0.833 Recall 0.044 Accuracy 0.899
Fold 5 Precision 0.783 Recall 0.080 Accuracy 0.902
Total  Precision 0.786 Recall 0.068 Accuracy 0.901
```

```
#
prob_discount = pd.DataFrame(data=pred_test_labels, columns=['0_dis','1_dis'])
customer_prob = prob_no_discount.join(prob_discount)
customer_prob.head()
```

	id	0	1	0_dis	1_dis
0	24011ae4ebbe3035111d65fa7c15bc57	0.74	0.26	0.430	0.570
1	764c75f661154dac3a6c254cd082ea7d	0.96	0.04	0.930	0.070
2	bba03439a292a1e166f80264c16191cb	0.93	0.07	0.952	0.048
3	1aa498825382410b098937d65c4ec26d	0.83	0.17	0.556	0.444
4	7ab4bf4878d8f7661dfc20e9b8e18011	0.86	0.14	0.520	0.480

```
# calculate expected profit of a single customer with discounted prices and without discounted prices
def expect_from_a_customer(customer_id):
    # expected values without discount
    average_yearly_price_var = price_stat[price_stat['id']==customer_id]['price_off_peak_var_mean'].values[0]
    total_yearly_usage_var = client[client['id']==customer_id]['cons_12m'].values[0]
    average_yearly_price_fix = price_stat[price_stat['id']==customer_id]['price_off_peak_fix_mean'].values[0]
    total_yearly_usage_fix = client[client['id']==customer_id]['cons_gas_12m'].values[0]
    total_profit = average_yearly_price_var*total_yearly_usage_var + average_yearly_price_fix*total_yearly_usage_fix
    #
    prob_stay = customer_prob[customer_prob['id']==customer_id]['0'].values[0]
    expected_profit = prob_stay*total_profit
    # expected values with discount
    total_profit_dis = total_profit*0.8
    #
    prob_stay_dis = customer_prob[customer_prob['id']==customer_id]['0_dis'].values[0]
    expected_profit_dis = prob_stay_dis*total_profit_dis
    return expected_profit, expected_profit_dis



# Calculate expected profit from each customer
expected_no_dis, expected_dis = [],[]
for cus_id in customer_prob['id']:
    temp = expect_from_a_customer(customer_id=cus_id)
    expected_no_dis.append(temp[0])
    expected_dis.append(temp[1])
#
customer_prob['expected_no_dis'] = expected_no_dis
customer_prob['expected_dis'] = expected_dis
customer_prob['diff_discount'] = customer_prob['expected_dis'] - customer_prob['expected_no_dis']
customer_prob
```

	id	0	1	0_dis	1_dis	expected_no_dis	expe
0	24011ae4ebbe3035111d65fa7c15bc57	0.74	0.26	0.430	0.570	1.664714e+06	7.738
1	764c75f661154dac3a6c254cd082ea7d	0.96	0.04	0.930	0.070	8.904801e+01	6.901
2	bba03439a292a1e166f80264c16191cb	0.93	0.07	0.952	0.048	2.227506e+02	1.824
3	1aa498825382410b098937d65c4ec26d	0.83	0.17	0.556	0.444	1.164197e+03	6.238
4	7ab4bf4878d8f7661dfc20e9b8e18011	0.86	0.14	0.520	0.480	6.440401e+03	3.115
...	...	...	...	...	...	...	
10826	c49217f16a06263e5381eaba94a67a8b	0.80	0.20	0.776	0.224	8.917784e+03	6.920
10827	18463073fb097fc0ac5d3e040f356987	0.85	0.15	0.900	0.100	1.812012e+06	1.534
10828	d0a6f71671571ed83b2645d23af6de00	0.28	0.72	0.354	0.646	2.159947e+02	2.184
10829	10e6828ddd62cbcf687cb74928c4c2d2	0.92	0.08	0.588	0.412	2.109376e+02	1.078
10830	1cf20fd6206d7678d5bcafd28c53b4db	0.91	0.09	0.948	0.052	1.783488e+01	1.486

10831 rows × 8 columns

```
customer_prob[customer_prob['diff_discount']>0]
```

	id	0	1	0_dis	1_dis	expected_no_dis	expe
16	78014630fbac3e6aa980361d25cea748	0.62	0.38	0.834	0.166	1.880583e+03	2.023
98	987a0a5e856d2185db60c718da435160	0.70	0.30	0.898	0.102	4.634752e+06	4.756
124	4f49021572f06e6410d351e4b2d9fc82	0.59	0.41	0.760	0.240	4.496141e+03	4.633
166	b1e773762052b38569b1e68aaa10576e	0.49	0.51	0.636	0.364	3.382264e+03	3.512
185	a5cb74bb90c992a62bfc6b8a7c0db94b	0.63	0.37	0.838	0.162	6.780647e+02	7.215
...	...	...	...	...	...	...	
10638	0c6cd9e51d6ac4742179ccdb51277f53	0.17	0.83	0.404	0.596	1.953563e+03	3.714