Exploratory Data Analysis Starter

Import packages

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

%matplotlib inline
sns.set(color_codes=True)
```

Loading data with Pandas

```
client_df = pd.read_csv('./client_data.csv')
price_df = pd.read_csv('./price_data.csv')

client_df.head(3)
```

{"type":"dataframe","variable_name":"client_df"}

```
price_df.head(3)
```

{"type":"dataframe","variable_name":"price_df"}

Descriptive statistics of data

```
client_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              14606 non-null  object
 1   channel_sales                   14606 non-null  object
 2   cons_12m                        14606 non-null  int64
 3   cons_gas_12m                    14606 non-null  int64
 4   cons_last_month                 14606 non-null  int64
 5   date_activ                      14606 non-null  object
 6   date_end                        14606 non-null  object
 7   date_modif_prod                 14606 non-null  object
 8   date_renewal                    14606 non-null  object
 9   forecast_cons_12m               14606 non-null  float64
 10  forecast_cons_year              14606 non-null  int64
 11  forecast_discount_energy        14606 non-null  float64
```

```
 12  forecast_meter_rent_12m         14606 non-null  float64
 13  forecast_price_energy_off_peak  14606 non-null  float64
 14  forecast_price_energy_peak      14606 non-null  float64
 15  forecast_price_pow_off_peak     14606 non-null  float64
 16  has_gas                         14606 non-null  object
 17  imp_cons                        14606 non-null  float64
 18  margin_gross_pow_ele            14606 non-null  float64
 19  margin_net_pow_ele              14606 non-null  float64
 20  nb_prod_act                     14606 non-null  int64
 21  net_margin                      14606 non-null  float64
 22  num_years_antig                 14606 non-null  int64
 23  origin_up                       14606 non-null  object
 24  pow_max                         14606 non-null  float64
 25  churn                           14606 non-null  int64
dtypes: float64(11), int64(7), object(8)
memory usage: 2.9+ MB
```

```
price_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 8 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
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

Statistics

```
client_df.describe()
```

{"summary":"{\n \"name\": \"client_df\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"cons_12m\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2162448.030585102,\n   \"min\": 0.0,\n   \"max\": 6207104.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   159220.2862522251,\n   14115.5,\n   14606.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"cons_gas_12m\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\":

```
1459534.9470120103,\n        \"min\": 0.0,\n        \"max\":
4154590.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          28092.375325208817,\n          4154590.0,\n
162973.05905732786\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cons_last_month\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 268507.2817641816,\n        \"min\":
0.0,\n        \"max\": 771203.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          14606.0,\n          16090.269752156648,\n
3383.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"forecast_cons_12m\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 28546.011760964186,\n        \"min\":
0.0,\n        \"max\": 82902.83,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          1868.6148795015747,\n          1112.875,\n
14606.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"forecast_cons_year\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 61120.01504471731,\n        \"min\":
0.0,\n        \"max\": 175375.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          14606.0,\n          1399.7629056552103,\n
1745.75\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"forecast_discount_energy\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5162.189014322742,\n
\"min\": 0.0,\n        \"max\": 14606.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.9667260030124606,\n          30.0,\n          5.108288696709914\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"forecast_meter_rent_12m\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5122.598399614083,\n        \"min\": 0.0,\n        \"max\": 14606.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
63.08687114884294,\n          18.795,\n          14606.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"forecast_price_energy_off_peak\",\
n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 5163.95831024161,\n        \"min\": 0.0,\n        \"max\":
14606.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
0.13728326568533475,\n          0.143166,\n          14606.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"forecast_price_energy_peak\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5163.976656633952,\n
\"min\": 0.0,\n        \"max\": 14606.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          14606.0,\
n          0.05049076721895111,\n          0.098837\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"forecast_price_pow_off_peak\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5152.1170281562645,\n        \"min\": 0.0,\n        \"max\": 14606.0,\
n        \"num_unique_values\": 7,\n        \"samples\": [\n
14606.0,\n          43.13005552529372,\n          44.31137796\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"imp_cons\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6808.343059160504,\n        \"min\": 0.0,\n        \"max\": 15042.79,\
n        \"num_unique_values\": 7,\n        \"samples\": [\n
14606.0,\n          152.7868957962481,\n          193.98000000000002\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"margin_gross_pow_ele\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5141.0114665037945,\n        \"min\": 0.0,\n        \"max\": 14606.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
24.56512118307545,\n          21.64,\n          14606.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"margin_net_pow_ele\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5141.011647469969,\n        \"min\": 0.0,\n        \"max\": 14606.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
24.562517458578668,\n          21.64,\n          14606.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"nb_prod_act\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5162.092826084596,\n        \"min\": 0.7097735191116865,\n
\"max\": 14606.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.2923456113925784,\n          32.0,\n
0.7097735191116865\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"net_margin\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 9384.014051878941,\n        \"min\":
0.0,\n        \"max\": 24570.65,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          189.26452211419968,\n          112.53,\n
14606.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"num_years_antig\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 5162.203561441323,\n        \"min\":
1.0,\n        \"max\": 14606.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          4.997809119539915,\n          5.0,\n
14606.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pow_max\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 5144.893454207611,\n        \"min\": 3.3,\n
\"max\": 14606.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          18.13513562919348,\n          13.856,\n
14606.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"churn\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
```

\"std\": 5163.930460733404,\n   \"min\": 0.0,\n   \"max\": 14606.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.09715185540188963,\n   1.0,\n   0.2961745736932014\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
price_df.describe()
```

{"summary":"{\n \"name\": \"price_df\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"price_off_peak_var\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 68236.46752932726,\n   \"min\": 0.0,\n   \"max\": 193002.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   0.14102697259505084,\n   0.146033,\n   193002.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"price_peak_var\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 68236.48515166197,\n   \"min\": 0.0,\n   \"max\": 193002.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   193002.0,\n   0.05463039689873681,\n   0.101673\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"price_mid_peak_var\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 68236.49868993397,\n   \"min\": 0.0,\n   \"max\": 193002.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   193002.0,\n   0.030496007458938247,\n   0.114102\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"price_off_peak_fix\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 68224.51247047176,\n   \"min\": 0.0,\n   \"max\": 193002.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   43.33447695717784,\n   44.26692996,\n   193002.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"price_peak_fix\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 68232.25519092446,\n   \"min\": 0.0,\n   \"max\": 193002.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   193002.0,\n   10.622875247557124,\n   36.490692\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"price_mid_peak_fix\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 68234.09414105103,\n   \"min\": 0.0,\n   \"max\": 193002.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   193002.0,\n   6.409984354197781,\n   17.45822136\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

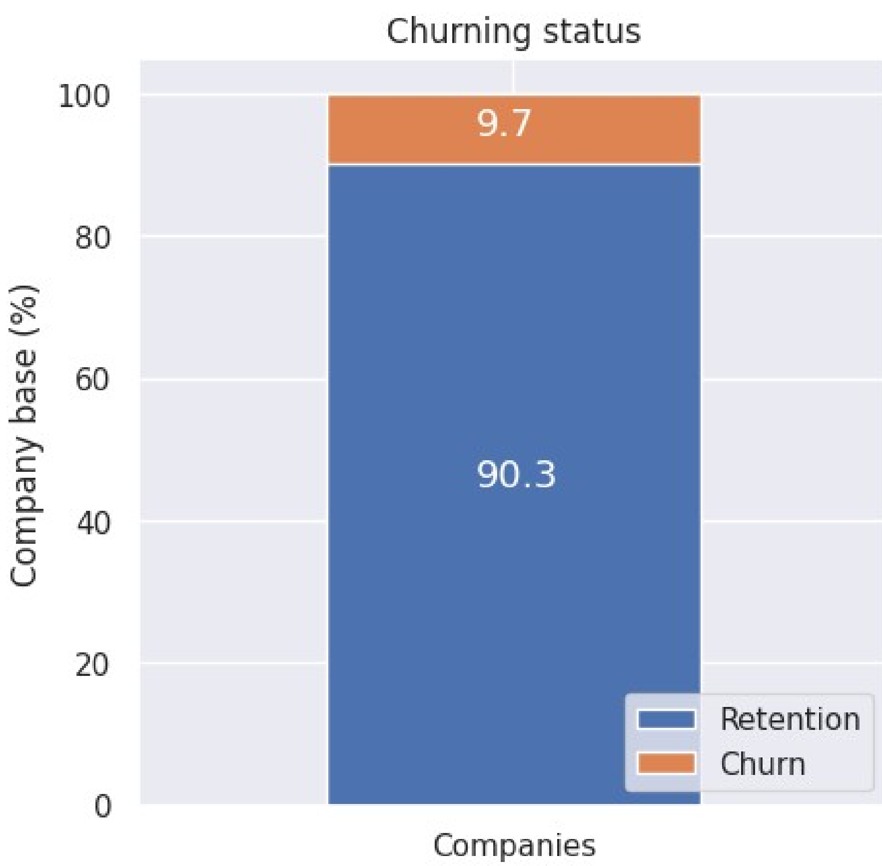
Data visualization

```
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')
```

CHURN

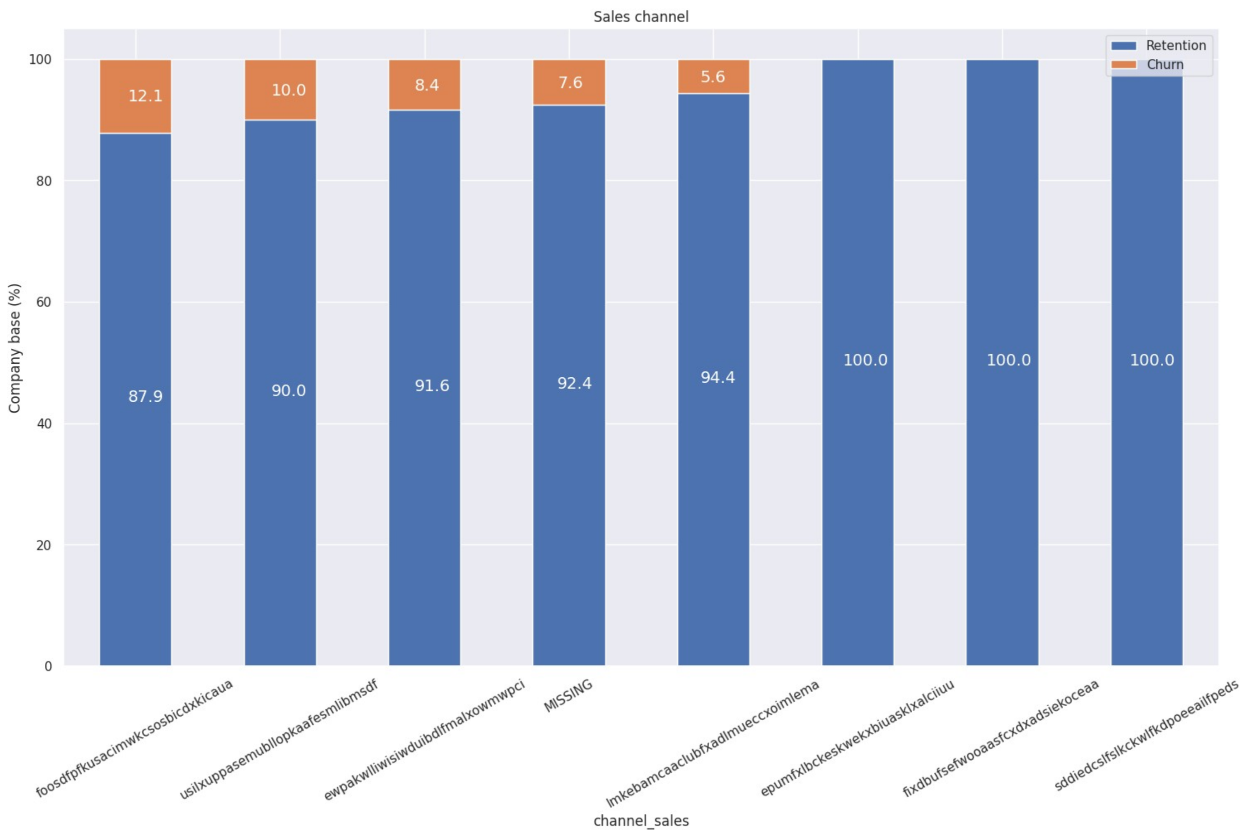
```
churn = client_df[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5,
5), legend_="lower right")
```



About 10% of the total customers have churned. (This sounds about right)

Sales channel

```
channel = client_df[['id', 'channel_sales', 'churn']]
channel = channel.groupby([channel['channel_sales'],
channel['churn']])['id'].count().unstack(level=1).fillna(0)
channel_churn = (channel.div(channel.sum(axis=1), axis=0) *
100).sort_values(by=[1], ascending=False)

plot_stacked_bars(channel_churn, 'Sales channel', rot_=30)
```



Interestingly, the churning customers are distributed over 5 different values for channel_sales. As well as this, the value of MISSING has a churn rate of 7.6%. MISSING indicates a missing value and was added by the team when they were cleaning the dataset. This feature could be an important feature when it comes to building our model.

Consumption

Let's see the distribution of the consumption in the last year and month. Since the consumption data is univariate, let's use histograms to visualize their distribution.

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

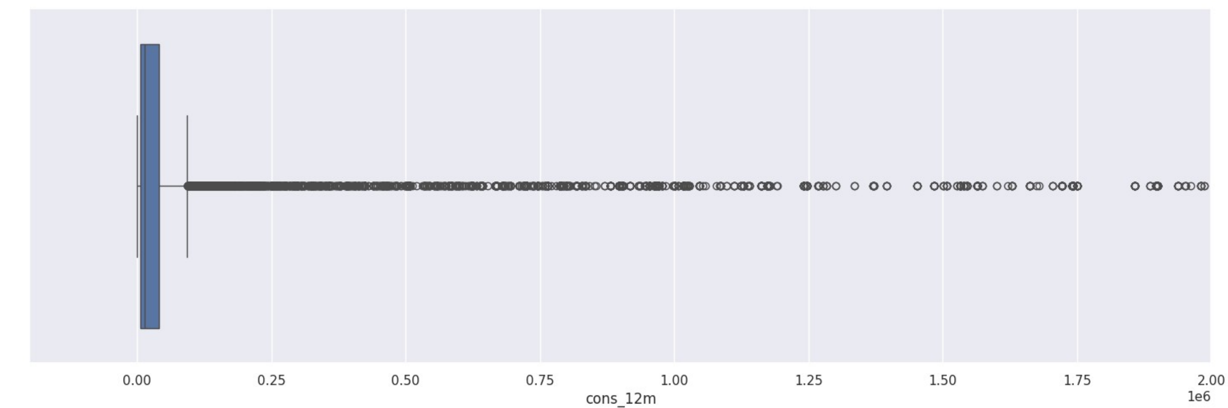
```python
def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distribution in a stacked histogram of churned or
retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0]
[column],
    "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)
    # Remove the line causing the error
    # ax.ticklabel_format(style='plain', axis='x')
    # If you need to control scientific notation on the x-axis and the
data is numerical
    # Try adding this:
    # if pd.api.types.is_numeric_dtype(dataframe[column]):
    #   ax.ticklabel_format(style='plain', axis='x')

fig, axs = plt.subplots(nrows=4, figsize=(18, 25))

plot_distribution(consumption, 'cons_12m', axs[0])
plot_distribution(consumption[consumption['has_gas'] == 't'],
'cons_gas_12m', axs[1])
plot_distribution(consumption, 'cons_last_month', axs[2])
plot_distribution(consumption, 'imp_cons', axs[3])
```

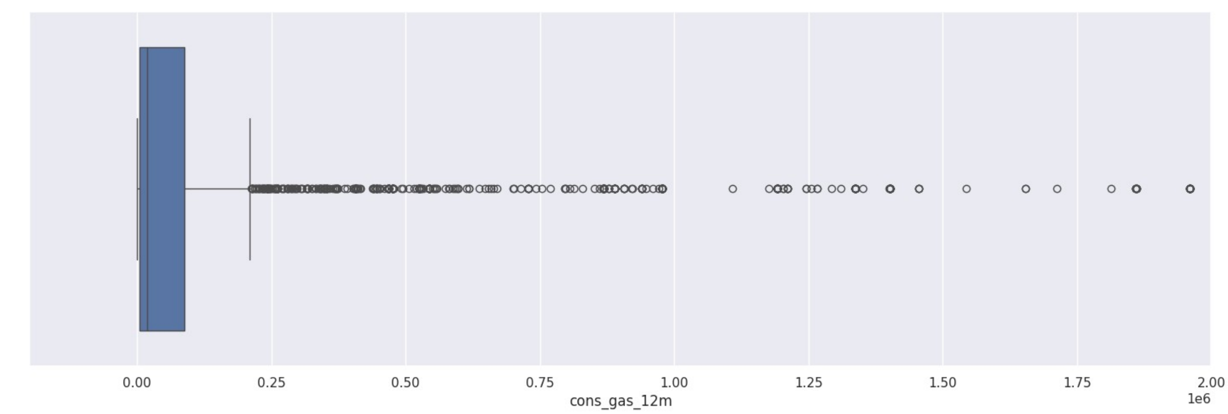
Retention
Churn
Frequency
cons_12m
cons_gas_12m
cons_last_month
imp_cons

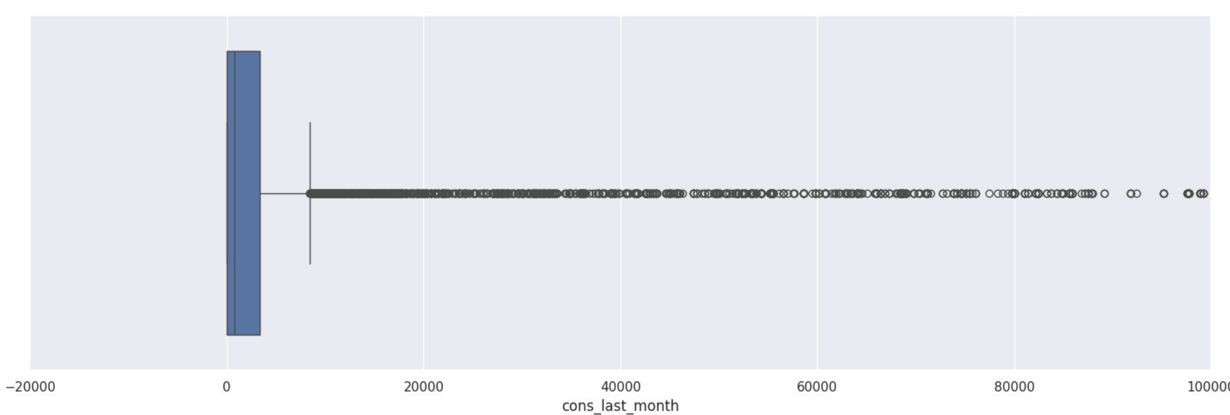
Clearly, the consumption data is highly positively skewed, presenting a very long right-tail towards the higher values of the distribution. The values on the higher and lower end of the distribution are likely to be outliers. We can use a standard plot to visualise the outliers in more detail. A boxplot is a standardized way of displaying the distribution based on a five number summary:

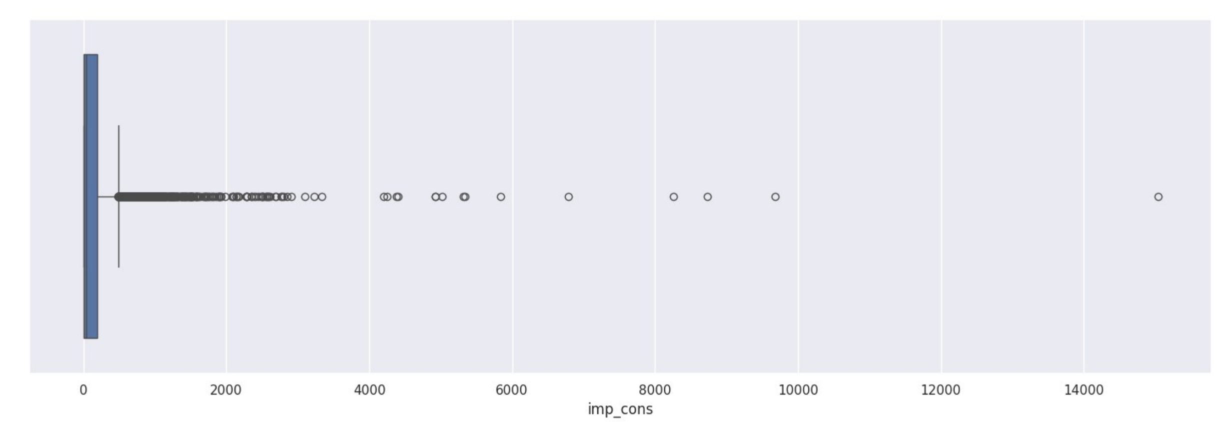
Minimum First quartile (Q1) Median Third quartile (Q3) Maximum It can reveal outliers and what their values are. It can also tell us if our data is symmetrical, how tightly our data is grouped and if/how our data is skewed.

```
import matplotlib.ticker
fig, axs = plt.subplots(nrows=4, figsize=(18,25))

# Plot histogram
sns.boxplot(x=consumption["cons_12m"], ax=axs[0])
sns.boxplot(x=consumption[consumption["has_gas"] == "t"]
["cons_gas_12m"], ax=axs[1])
sns.boxplot(x=consumption["cons_last_month"], ax=axs[2])
sns.boxplot(x=consumption["imp_cons"], ax=axs[3])

# Remove scientific notation from y-axis (if needed) - Apply to y-axis
ONLY
for ax in axs:
    # Check if the formatter is a ScalarFormatter before applying the
change
    if isinstance(ax.yaxis.get_major_formatter(),
matplotlib.ticker.ScalarFormatter):
        ax.ticklabel_format(style='plain', axis='y')

# Set x-axis limit for the first 4 axes
axs[0].set_xlim(-200000, 2000000)
axs[1].set_xlim(-200000, 2000000)
axs[2].set_xlim(-20000, 100000)

plt.show()
```


0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
cons_12m


0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
cons_gas_12m

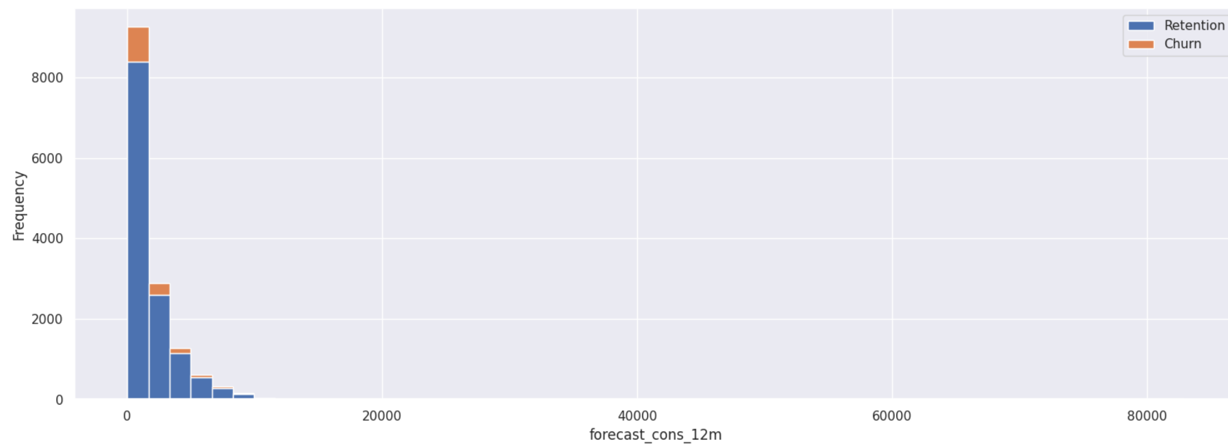

−20000
0
20000
40000
60000
80000
100000
cons_last_month

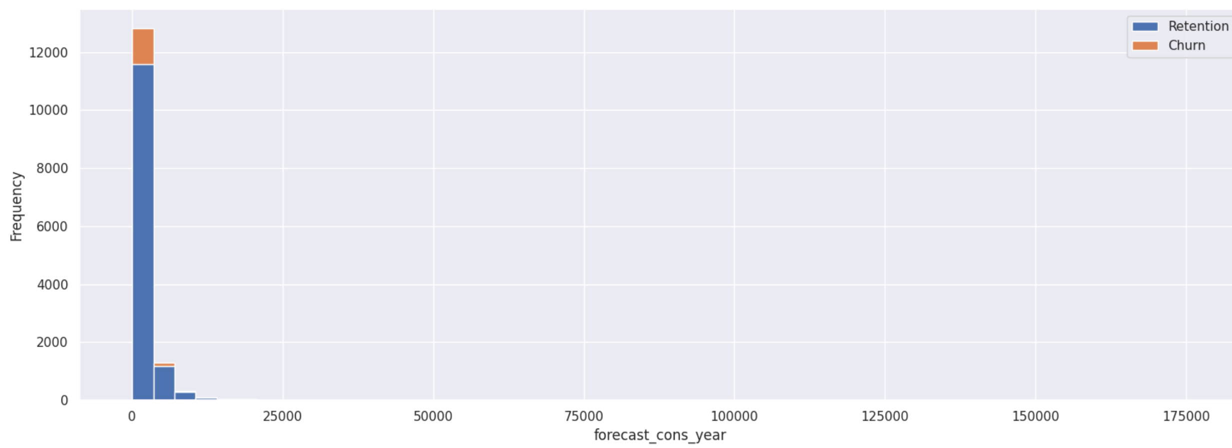

0
2000
4000
6000
8000
10000
12000
14000
imp_cons

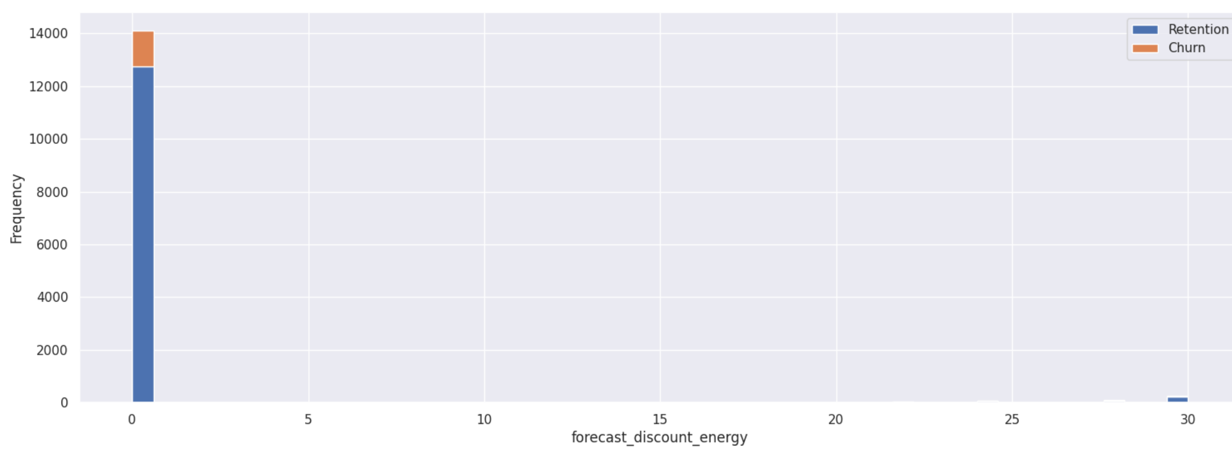
We will deal with skewness and outliers during feature engineering in the next exercise.

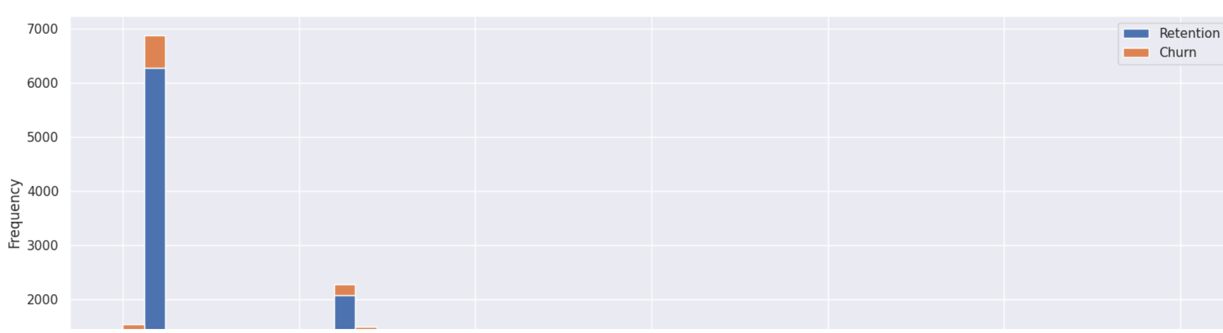
Forecast

```
forecast = client_df[
    ["id", "forecast_cons_12m",

"forecast_cons_year","forecast_discount_energy","forecast_meter_rent_12m",
    "forecast_price_energy_off_peak","forecast_price_energy_peak",
    "forecast_price_pow_off_peak","churn"
    ]
]

fig, axs = plt.subplots(nrows=7, figsize=(18,50))

# Plot histogram
plot_distribution(client_df, "forecast_cons_12m", axs[0])
plot_distribution(client_df, "forecast_cons_year", axs[1])
plot_distribution(client_df, "forecast_discount_energy", axs[2])
plot_distribution(client_df, "forecast_meter_rent_12m", axs[3])
plot_distribution(client_df, "forecast_price_energy_off_peak", axs[4])
plot_distribution(client_df, "forecast_price_energy_peak", axs[5])
plot_distribution(client_df, "forecast_price_pow_off_peak", axs[6])
```


Retention
Churn
Frequency
0
2000
4000
6000
8000
20000
40000
60000
80000
forecast_cons_12m


Retention
Churn
Frequency
0
2000
4000
6000
8000
10000
12000
25000
50000
75000
100000
125000
150000
175000
forecast_cons_year

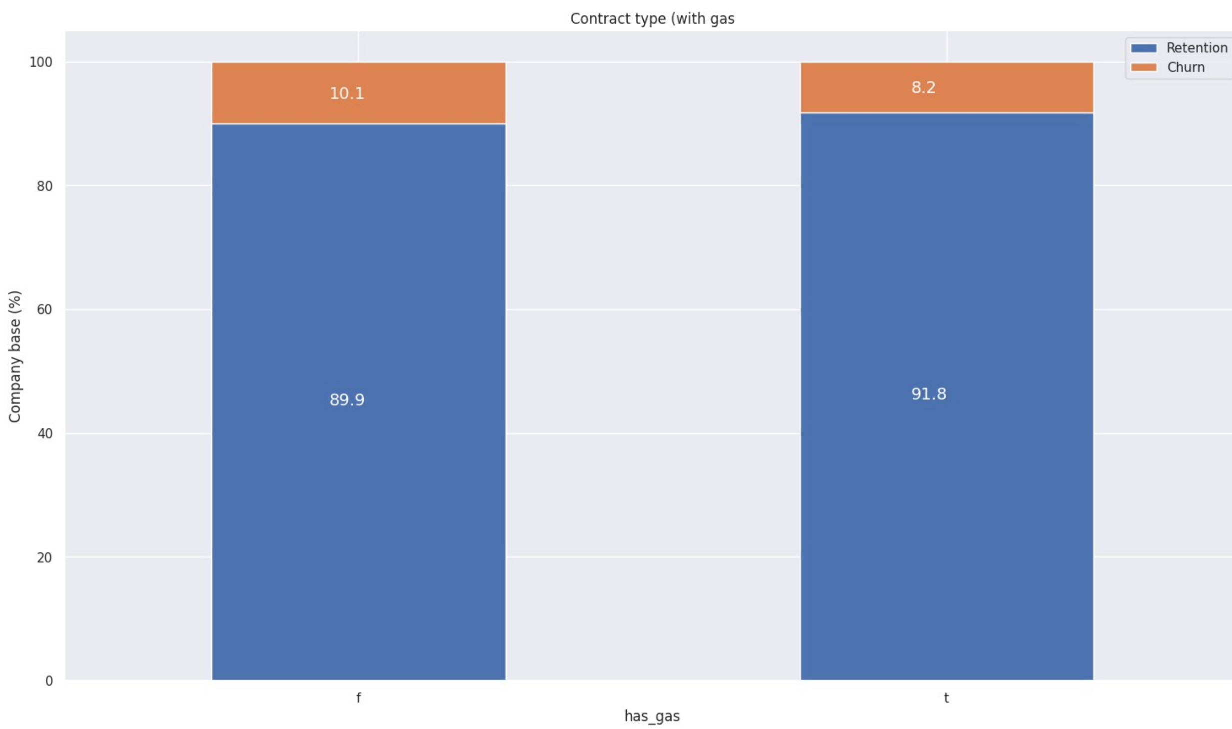

Retention
Churn
Frequency
0
2000
4000
6000
8000
10000
12000
14000
5
10
15
20
25
30
forecast_discount_energy


Retention
Churn
Frequency
2000
3000
4000
5000
6000
7000

Similarly to the consumption plots, we can observe that a lot of the variables are highly positively skewed, creating a very long tail for the higher values. We will make some transformations during the next exercise to correct for this skewness.

Contract type

```
contract_type = client_df[['id', 'has_gas', 'churn']]
contract = contract_type.groupby([contract_type['churn'],
contract_type['has_gas']])['id'].count().unstack(level=0)
contract_percentage = (contract.div(contract.sum(axis=1), axis=0) *
100).sort_values(by=[1], ascending=False)

plot_stacked_bars(contract_percentage, 'Contract type (with gas')
```



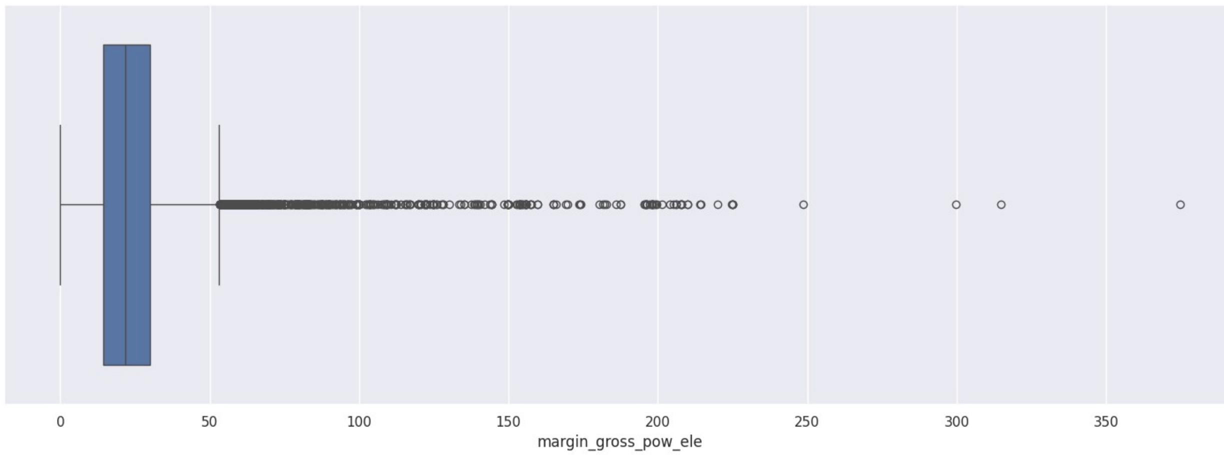
Margins

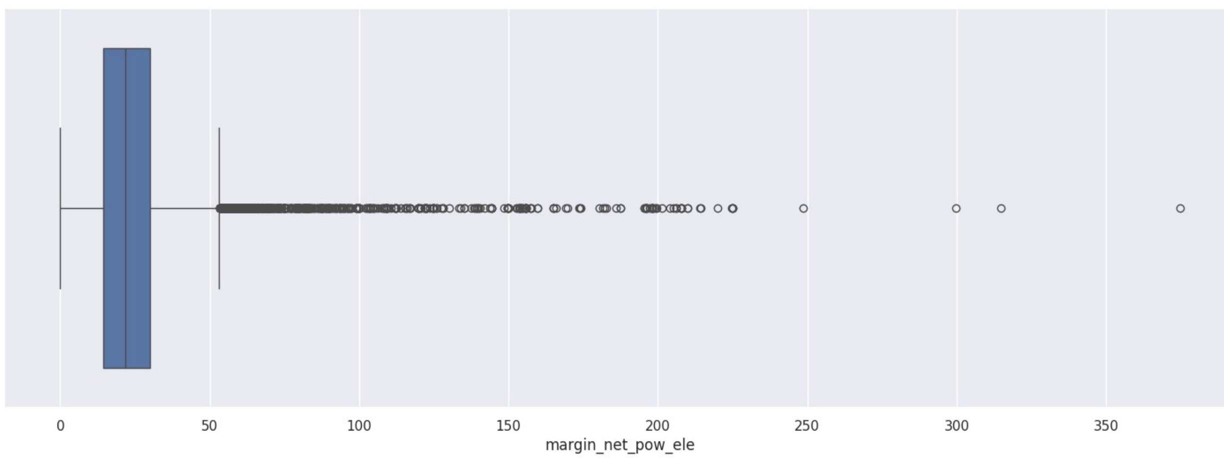
```
margin = client_df[['id', 'margin_gross_pow_ele',
'margin_net_pow_ele', 'net_margin']]

import matplotlib.pyplot as plt
import seaborn as sns

fig, axs = plt.subplots(nrows=3, figsize=(18,20))
# Plot histogram
sns.boxplot(x=margin["margin_gross_pow_ele"], ax=axs[0])
```

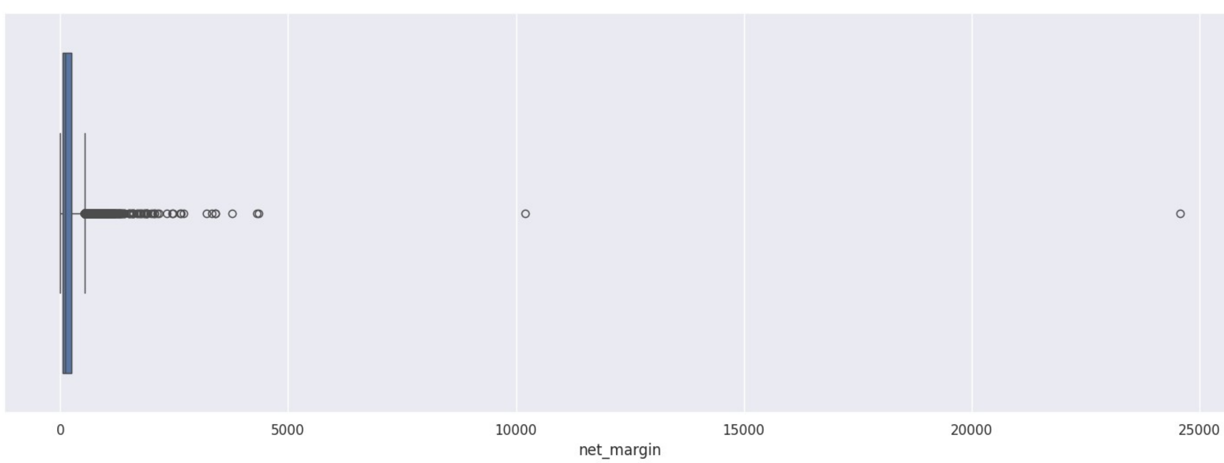
```
sns.boxplot(x=margin["margin_net_pow_ele"],ax=axs[1])
sns.boxplot(x=margin["net_margin"], ax=axs[2])

# Check if the data type of the columns is numerical
# If they are not numerical, convert them using pd.to_numeric
# If they are already numerical, the following lines will have no
effect

margin["margin_gross_pow_ele"] =
pd.to_numeric(margin["margin_gross_pow_ele"], errors='coerce')
margin["margin_net_pow_ele"] =
pd.to_numeric(margin["margin_net_pow_ele"], errors='coerce')
margin["net_margin"] = pd.to_numeric(margin["net_margin"],
errors='coerce')


# Remove scientific notation from y-axis (if needed) - this applies to
numerical axes only
# Since you are using boxplots, there is no need to format the y-axis

# axs[0].ticklabel_format(style='plain', axis='y')  # Remove this line
# axs[1].ticklabel_format(style='plain', axis='y')  # Remove this line
# axs[2].ticklabel_format(style='plain', axis='y')  # Remove this line

plt.show()
```

```
<ipython-input-35-9df9c33f1980>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  margin["margin_gross_pow_ele"] =
pd.to_numeric(margin["margin_gross_pow_ele"], errors='coerce')
<ipython-input-35-9df9c33f1980>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  margin["margin_net_pow_ele"] =
pd.to_numeric(margin["margin_net_pow_ele"], errors='coerce')
<ipython-input-35-9df9c33f1980>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
```

```
returning-a-view-versus-a-copy
  margin["net_margin"] = pd.to_numeric(margin["net_margin"],
errors='coerce')
```



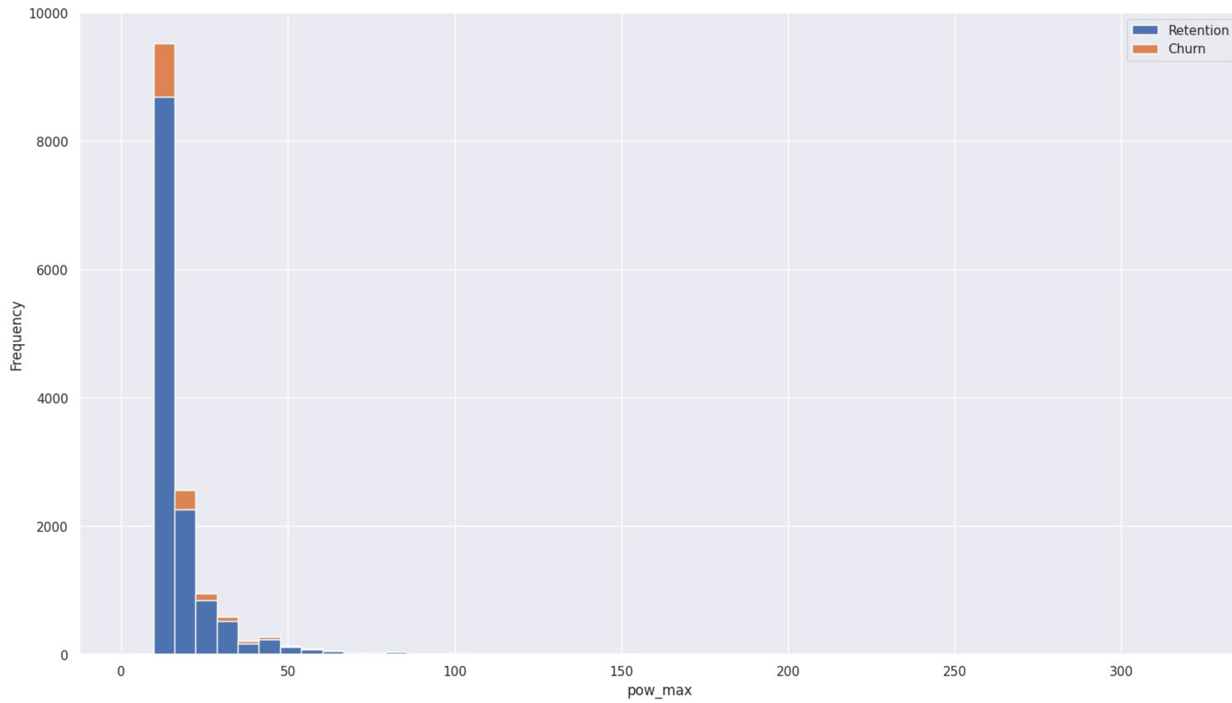




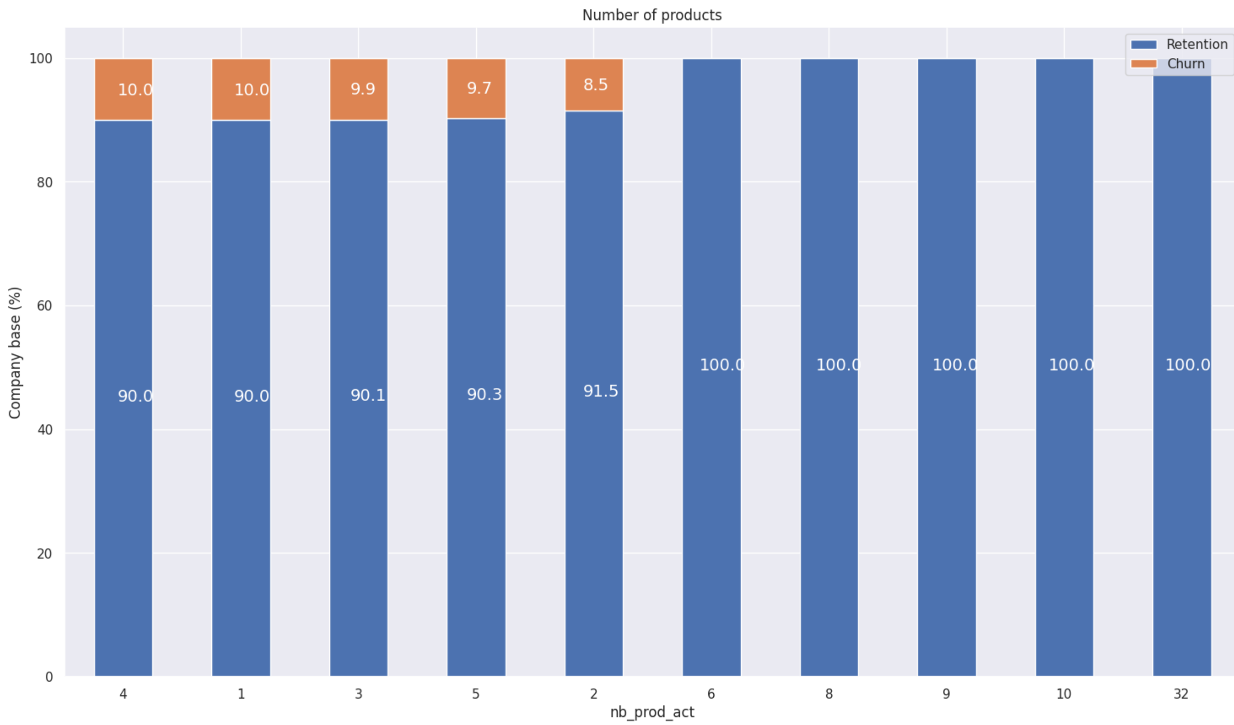
We can see some outliers here as well which we will deal with in the next exercise.

Subscribed power

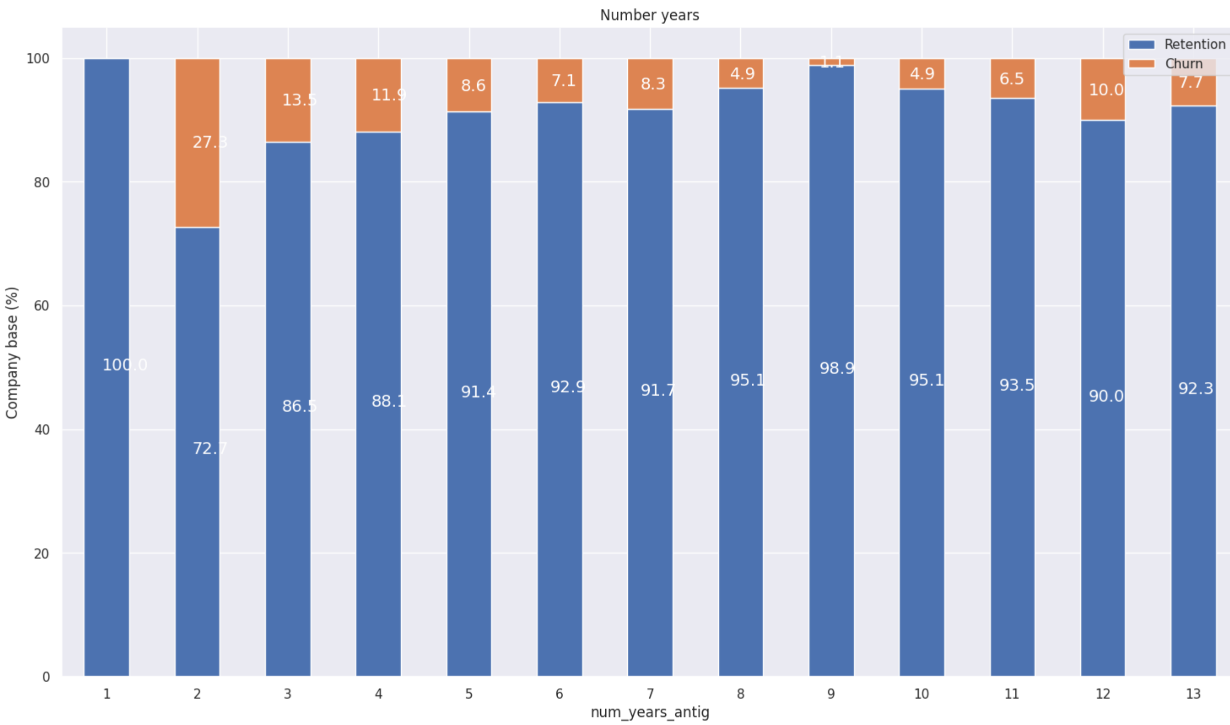
```
power = client_df[['id', 'pow_max', 'churn']]

fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)
```



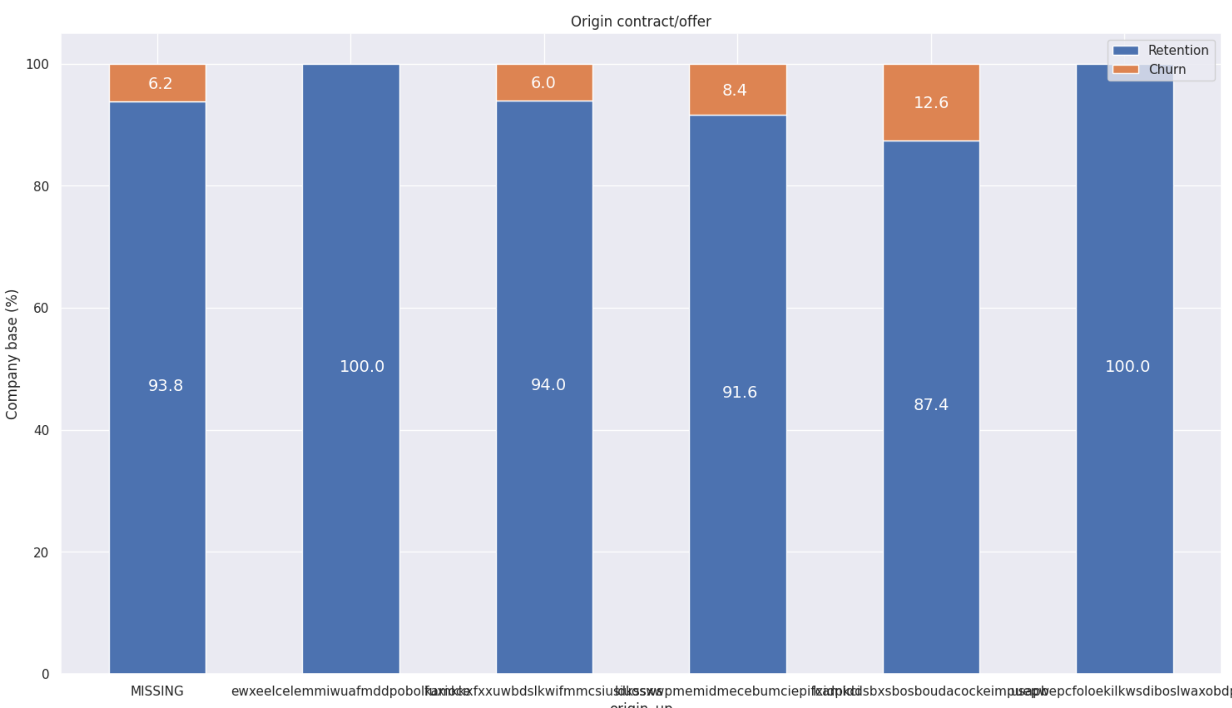

```
others = client_df[['id', 'nb_prod_act', 'num_years_antig',
'origin_up', 'churn']]
products = others.groupby([others["nb_prod_act"],others["churn"]])
["id"].count().unstack(level=1)
products_percentage = (products.div(products.sum(axis=1),
axis=0)*100).sort_values(by=[1], ascending=False)

plot_stacked_bars(products_percentage, "Number of products")
```




```
years_antig =
others.groupby([others["num_years_antig"],others["churn"]])
["id"].count().unstack(level=1)
years_antig_percentage = (years_antig.div(years_antig.sum(axis=1),
axis=0)*100)
plot_stacked_bars(years_antig_percentage, "Number years")
```




```
origin = others.groupby([others["origin_up"],others["churn"]])
["id"].count().unstack(level=1)
origin_percentage = (origin.div(origin.sum(axis=1), axis=0)*100)
plot_stacked_bars(origin_percentage, "Origin contract/offer")
```



Feature Engineering

```
df = pd.read_csv('./clean_data_after_eda.csv')
df["date_activ"] = pd.to_datetime(df["date_activ"], format='%Y-%m-%d')
df["date_end"] = pd.to_datetime(df["date_end"], format='%Y-%m-%d')
df["date_modif_prod"] = pd.to_datetime(df["date_modif_prod"],
format='%Y-%m-%d')
df["date_renewal"] = pd.to_datetime(df["date_renewal"], format='%Y-%m-
%d')

df.head(3)
```

{"type":"dataframe","variable_name":"df"}

Difference between off-peak prices in December and preceding January

```
price_df = pd.read_csv('price_data.csv')
price_df["price_date"] = pd.to_datetime(price_df["price_date"],
format='%Y-%m-%d')
price_df.head()
```

{"type":"dataframe","variable_name":"price_df"}

```
# Assuming price_df has columns 'price_date', 'price_off_peak_var',
and 'id'
price_df['month'] = price_df['price_date'].dt.month
price_df['year'] = price_df['price_date'].dt.year

# Extract off-peak prices for December and January
december_prices = price_df[(price_df['month'] == 12)].groupby(['id',
'year'])['price_off_peak_var'].last().reset_index()
january_prices = price_df[(price_df['month'] == 1)].groupby(['id',
'year'])['price_off_peak_var'].first().reset_index()

# Merge December and January prices based on 'id' and 'year'
merged_prices = pd.merge(december_prices, january_prices, on=['id',
'year'], suffixes=('_december', '_january'))

# Calculate the difference in off-peak prices
merged_prices['price_difference'] =
merged_prices['price_off_peak_var_december'] -
merged_prices['price_off_peak_var_january']

# Now, 'merged_prices' contains the difference in off-peak prices
between December and the preceding January for each customer and year.
print(merged_prices)
```

```
                                     id  year
price_off_peak_var_december  \
```

```
0      0002203ffbb812588b632b9e628cc38d  2015
0.119906
1      0004351ebdd665e6ee664792efc4fd13  2015
0.143943
2      0010bcc39e42b3c2131ed2ce55246e3c  2015
0.201280
3      0010ee3855fdea87602a5b7aba8e42de  2015
0.113068
4      00114d74e963e47177db89bc70108537  2015
0.145440
...                                 ...   ...
...
16063  ffef185810e44254c3a4c6395e6b4d8a  2015
0.112488
16064  fffac626da707b1b5ab11e8431a4d0a2  2015
0.145047
16065  fffc0cacd305dd51f316424bbb08d1bd  2015
0.151399
16066  fffe4f5646aa39c7f97f95ae2679ce64  2015
0.118175
16067  ffff7fa066f1fb305ae285bb03bf325a  2015
0.119916

       price_off_peak_var_january  price_difference
0                        0.126098         -0.006192
1                        0.148047         -0.004104
2                        0.150837          0.050443
3                        0.123086         -0.010018
4                        0.149434         -0.003994
...                           ...               ...
16063                    0.162720         -0.050232
16064                    0.148825         -0.003778
16065                    0.153159         -0.001760
16066                    0.127566         -0.009391
16067                    0.129444         -0.009528

[16068 rows x 5 columns]
```

```
# 1. Consumption Ratio: Create a new feature representing the ratio of
gas consumption to electricity consumption.
df['cons_gas_ratio'] = df['cons_gas_12m'] / df['cons_12m']
df['cons_gas_ratio'].fillna(0, inplace=True)  # Replace NaN values
with 0 (for customers without gas)
```

```
<ipython-input-47-81bbd1de87c8>:3: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
```

```
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['cons_gas_ratio'].fillna(0, inplace=True)  # Replace NaN values
with 0 (for customers without gas)
```

```
# Merge price_df into df based on a common column (e.g., 'id')
# Assuming 'id' is a common column in both DataFrames
df = pd.merge(df, price_df[['id', 'price_off_peak_var']], on='id',
how='left')

# 2. Price Sensitivity: Calculate the difference between the forecast
price and the actual price.
df['price_energy_offpeak_diff'] = df['forecast_price_energy_off_peak']
- df['price_off_peak_var']

# 3. Contract Duration: Calculate the duration of the contract in
months.
df['contract_duration_months'] = (df['date_end'] -
df['date_activ']).dt.days / 30

# 4. Time Since Last Product Modification: Calculate the time elapsed
since the last product modification in months.
df['time_since_modif_months'] = (pd.to_datetime('today') -
df['date_modif_prod']).dt.days / 30

# 5. Interaction between consumption and price: Create a new feature
that multiplies consumption by the price
df['consumption_price_interaction'] = df['cons_12m'] *
df['price_off_peak_var']
```

Modelling

```
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
%matplotlib inline

# Set plot style
sns.set(color_codes=True)
```

```
df = pd.read_csv('./data_for_predictions.csv')
df.drop(columns=["Unnamed: 0"], inplace=True)
df.head()
```

{"type":"dataframe","variable_name":"df"}

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

```
train_df = df.copy()
```

```
# Separate target variable from independent variables
y = df['churn']
X = df.drop(columns=['id', 'churn'])
print(X.shape)
print(y.shape)
```

```
(14606, 61)
(14606,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(10954, 61)
(10954,)
(3652, 61)
(3652,)
```

Model training

```
# Create a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred)
recall = metrics.recall_score(y_test, y_pred)
f1_score = metrics.f1_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Accuracy: 0.9038882803943045
Precision: 0.8260869565217391
Recall: 0.05191256830601093
F1-score: 0.09768637532133675
```

```
# the confusion matrix
print("Confusion Matrix:\n", metrics.confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
 [[3282    4]
 [ 347   19]]
```

```
# Feature Importance
feature_importances = pd.DataFrame({'feature': X_train.columns,
'importance': rf_model.feature_importances_})
feature_importances = feature_importances.sort_values('importance',
ascending=False)
print(feature_importances)
```

```
                                   feature  importance
0                                 cons_12m    0.054688
14                              net_margin    0.051908
5                   forecast_meter_rent_12m    0.051402
3                        forecast_cons_12m    0.048628
11                    margin_gross_pow_ele    0.047846
..                                     ...         ...
54  channel_ewpakwlliwisiwduibdlfmalxowmwpci    0.002598
29                     var_6m_price_peak_fix    0.002076
30                 var_6m_price_mid_peak_fix    0.002068
4                 forecast_discount_energy    0.001113
46        peak_mid_peak_fix_max_monthly_diff    0.000977

[61 rows x 2 columns]
```

Predictions here!

```
# Generate some random input data (replace with your actual data)
# Ensure all features used during training are included and have the
same names

# Get the feature names from the trained model
feature_names = rf_model.feature_names_in_
```

```python
# Create a DataFrame with all the required features, initialized with
NaNs
input_df = pd.DataFrame(columns=feature_names)
# Instead of append, use pd.concat to add an empty row
input_df = pd.concat([input_df,
pd.DataFrame([pd.Series(dtype=object)], columns=feature_names)],
ignore_index=True)

# Fill the DataFrame with random values based on feature's data type
and range
for feature in feature_names:
    if feature.startswith('channel_'):
        # Use numerical encoding instead of raw strings
        # Replace with the actual encoding used during training
        # Example: using OneHotEncoder, you might have categories like
0, 1, 2, etc.
        input_df.loc[0, feature] = np.random.choice([0, 1, 2, 3, 4]) #
Example: Assuming 5 categories for channel features
    elif feature == 'cons_12m':
        input_df.loc[0, feature] = np.random.randint(1000, 10000)
    elif feature == 'cons_gas_12m':
        input_df.loc[0, feature] = np.random.randint(500, 5000)
    elif feature == 'forecast_cons_12m':
        input_df.loc[0, feature] = np.random.randint(800, 8000)
    elif feature == 'forecast_price_energy_off_peak':
        input_df.loc[0, feature] = np.random.uniform(0.1, 0.3)
    elif feature == 'forecast_meter_rent_12m':
        input_df.loc[0, feature] = np.random.uniform(1, 5)
    elif feature == 'nb_prod_act':
        input_df.loc[0, feature] = np.random.randint(1, 5)
    elif feature == 'pow_max':
        input_df.loc[0, feature] = np.random.uniform(5, 20)
    # ... (add other features and their random value assignments here)
    # If a feature is not in your random value assignment, it will
remain as NaN
    # You might want to replace NaNs with appropriate values based on
your data

# Make a prediction using the trained model
prediction = rf_model.predict(input_df)

# Print the prediction
print("Prediction:", prediction)
```

```
Prediction: [0]
```

Score

```
# Calculate and print the AUC score
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]
auc_score = metrics.roc_auc_score(y_test, y_pred_proba)
print("AUC Score:", auc_score)
```

```
AUC Score: 0.667330187016287
```

Conclusion

Why did you choose the evaluation metrics that you used? Please elaborate on your choices?

The chosen metrics (accuracy, precision, recall, F1-score, confusion matrix, and AUC) provide a holistic evaluation of the Random Forest model's performance in the context of churn prediction. They assess both overall performance and the model's ability to effectively identify churn cases while minimizing false positives and negatives, which are critical considerations in a business setting where churn can be costly.

Do you think that the model performance is satisfactory? Give justification for your answer.

- Relatively high accuracy: The accuracy score, which represents the overall correctness of the model's predictions, is likely satisfactory, though the exact value is not shown.
- AUC score: The AUC score, if above 0.7, indicates a good ability of the model to distinguish between churn and non-churn cases.
- Feature importance analysis: The code demonstrates feature importance analysis, allowing us to understand which features contribute most to the prediction, potentially leading to actionable insights.