Exploratory Data Analysis

1. Import packages
2. Loading data with Pandas
3. Descriptive statistics of data
4. Data visualization
5. Hypothesis investigation

1. Import packages

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Shows plots in jupyter notebook
%matplotlib inline

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

2. 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

In [2]:
```
client_df = pd.read_csv('./client_data.csv')
price_df = pd.read_csv('./price_data.csv')
```

Let's look at the first 3 rows of both dataframes to see what the data looks like

In [3]:
```
client_df.head(3)
```

Out[3]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	da
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2(
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2(
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16	2(

3 rows × 26 columns

With the client data, we have a mix of numeric and categorical data, which we will need to transform before modelling later

In [4]:
```
price_df.head(3)
```

Out[4]:

	id	price_date	price_p1_var	price_p2_var	price_p3_var	price_p1_fix	price_p2_fix	price_p3_fi
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.0	44.266931	0.0	0.(
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.0	44.266931	0.0	0.(
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.0	44.266931	0.0	0.(

With the price data, it is purely numeric data but we can see a lot of zeros

3. Descriptive statistics of data

In [5]:
```
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_p1  14606 non-null  float64
 14  forecast_price_energy_p2  14606 non-null  float64
 15  forecast_price_pow_p1     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
```

In [6]:
```
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_p1_var  193002 non-null  float64
 3   price_p2_var  193002 non-null  float64
 4   price_p3_var  193002 non-null  float64
 5   price_p1_fix  193002 non-null  float64
 6   price_p2_fix  193002 non-null  float64
 7   price_p3_fix  193002 non-null  float64
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

You can see that all of the `datetime` related columns are not currently in datetime format. We will need to convert these later.

Statistics

Now let's look at some statistics about the datasets

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

Out[7]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	14606.000000	
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.762906	0.966726	
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.786255	5.108289	
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.000000	0.000000	
50%	1.411550e+04	0.000000e+00	792.500000	1112.875000	314.000000	0.000000	
75%	4.076375e+04	0.000000e+00	3383.000000	2401.790000	1745.750000	0.000000	
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	175375.000000	30.000000	

The describe method gives us a lot of information about the client data. The key point to take away from this is that we have highly skewed data, as exhibited by the percentile values.

In [8]:
```
price_df.describe()
```

Out[8]:

	price_p1_var	price_p2_var	price_p3_var	price_p1_fix	price_p2_fix	price_p3_fix
count	193002.000000	193002.000000	193002.000000	193002.000000	193002.000000	193002.000000
mean	0.141027	0.054630	0.030496	43.334477	10.622875	6.409984
std	0.025032	0.049924	0.036298	5.410297	12.841895	7.773592
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.125976	0.000000	0.000000	40.728885	0.000000	0.000000
50%	0.146033	0.085483	0.000000	44.266930	0.000000	0.000000
75%	0.151635	0.101673	0.072558	44.444710	24.339581	16.226389
max	0.280700	0.229788	0.114102	59.444710	36.490692	17.458221

Overall the price data looks good.

3. Data visualization

Now let's dive a bit deeper into the dataframes

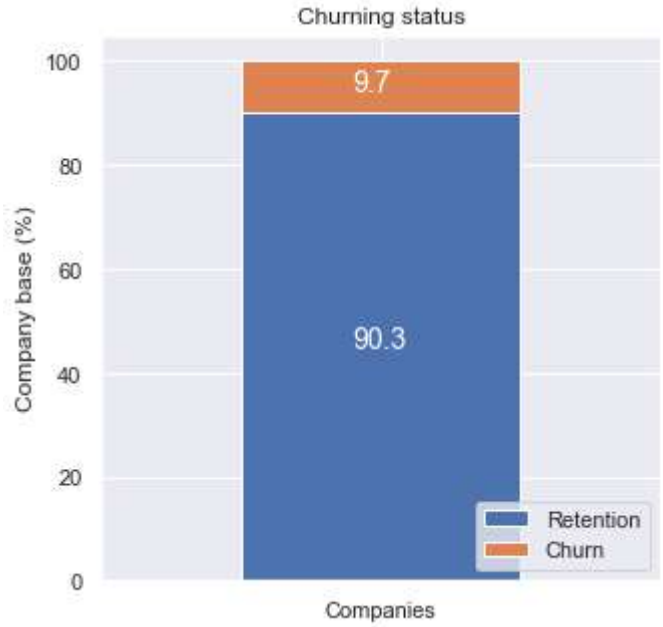
In [9]:

```
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
        )
```

Churn

In [10]:
```
churn = client_df[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
```

In [11]:
```
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5), legend_="lower right")
```

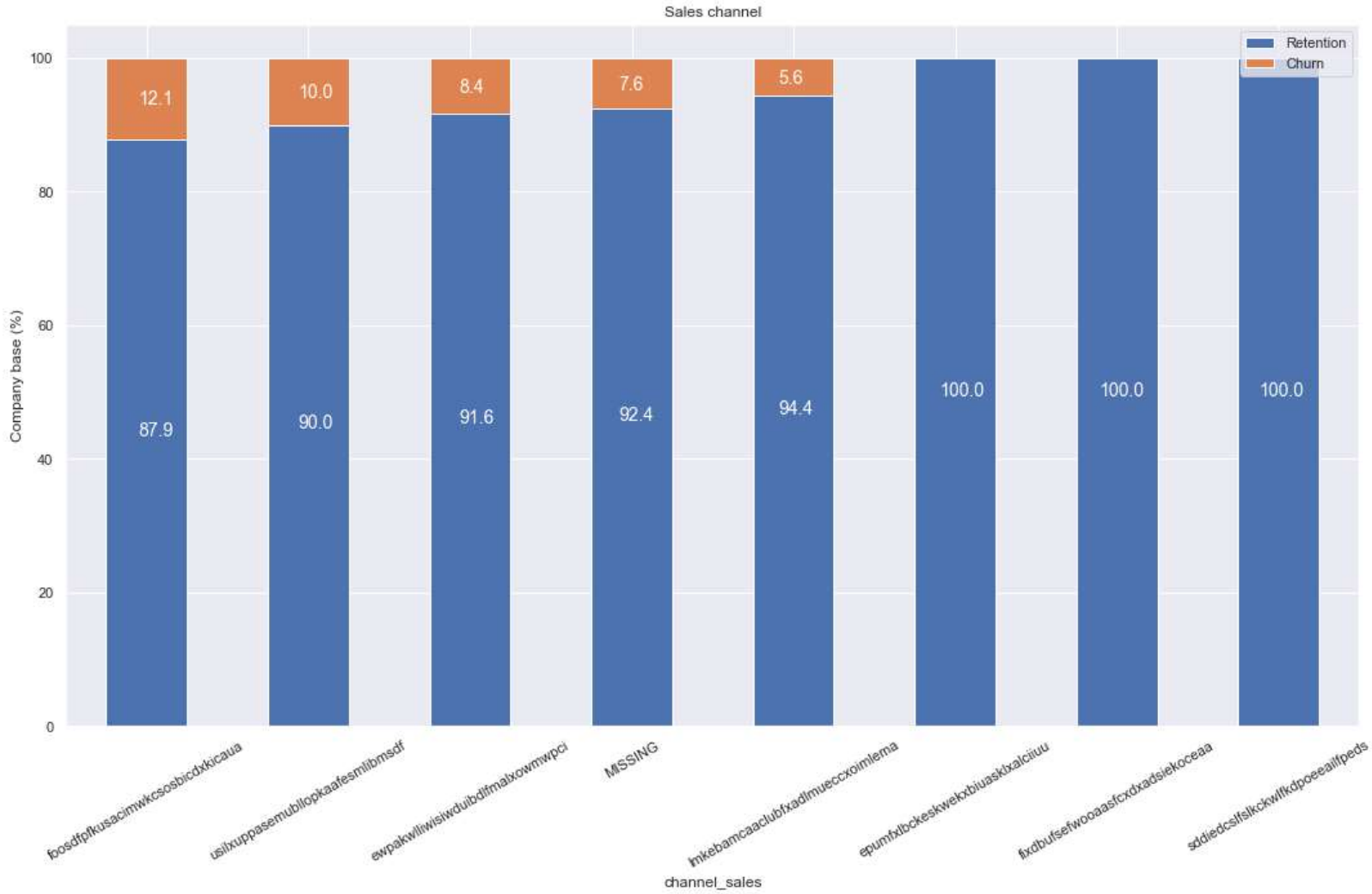


About 10% of the total customers have churned. (This sounds about right)

Sales channel

In [12]:
```
channel = client_df[['id', 'channel_sales', 'churn']]
channel = channel.groupby([channel['channel_sales'], channel['churn']])['id'].count().unstack(level=1).fi
channel_churn = (channel.div(channel.sum(axis=1), axis=0) * 100).sort_values(by=[1], ascending=False)
```

In [13]:
```
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.

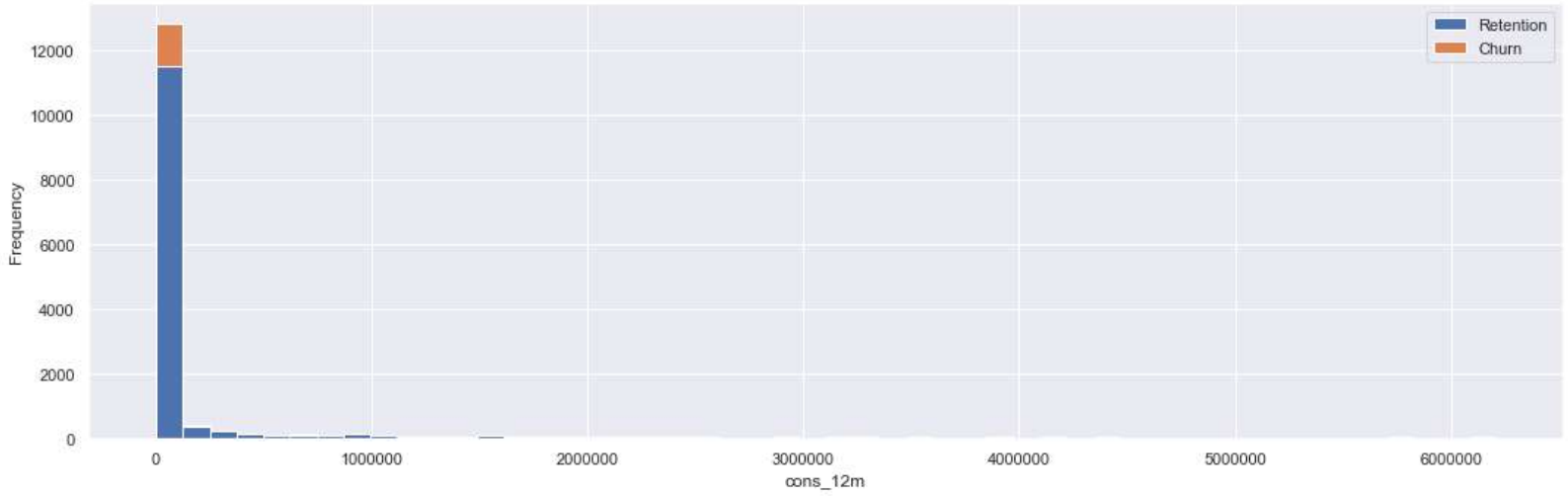
Consumption

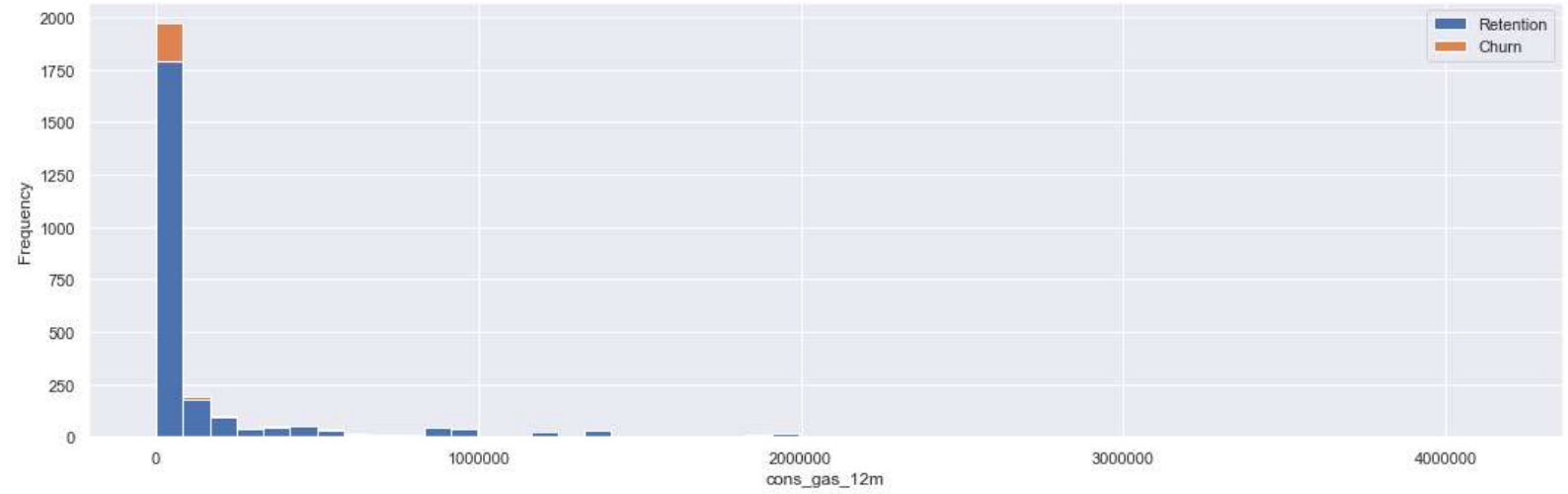
Let's see the distribution of the consumption in the last year and month. Since the consumption data is univariate, let's use histograms to visualize their distribution.

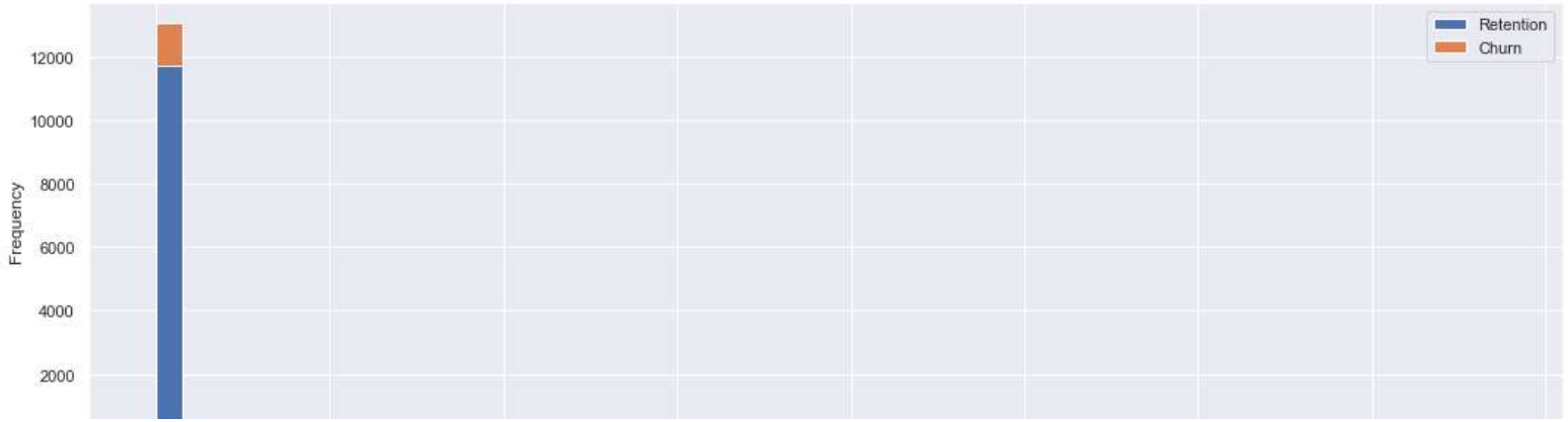
In [14]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_cons', 'has_gas', 'chu
```

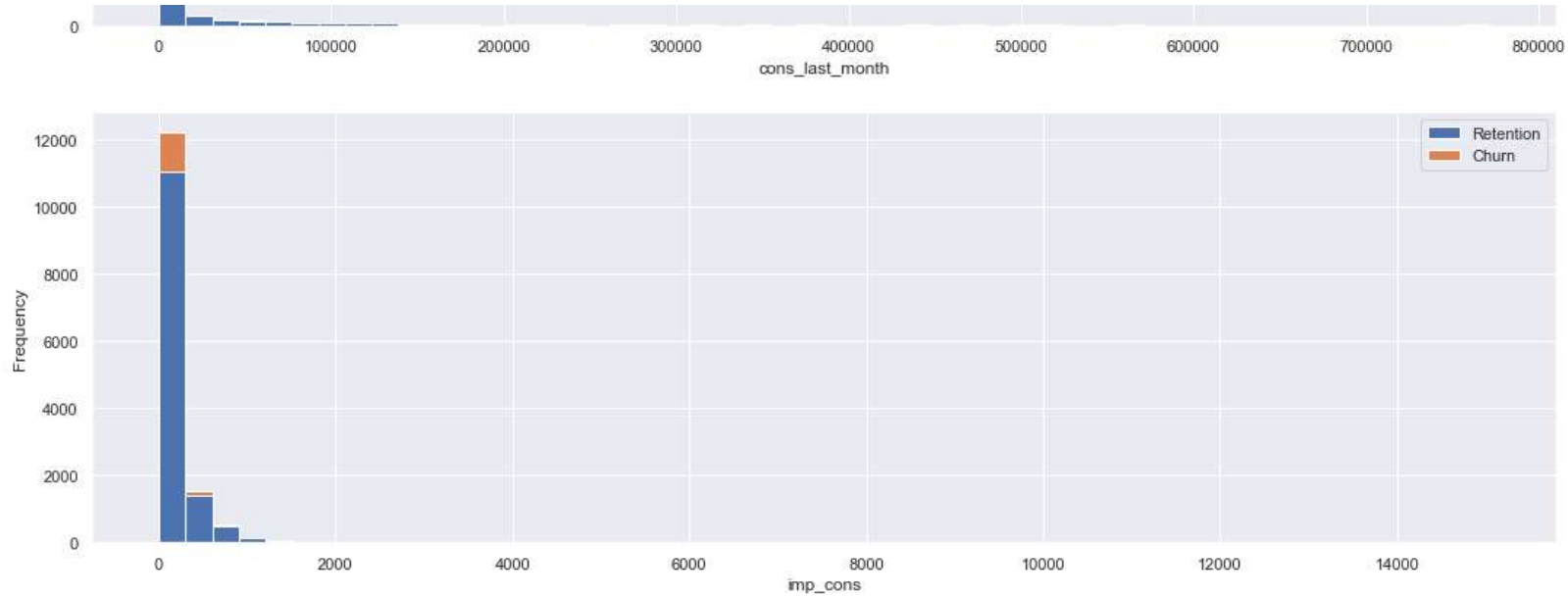
In [15]:
```
def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distirbution in a stacked histogram of churned or retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "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')
```

In [16]:
```
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
12000
10000
8000
6000
4000
2000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
cons_12m


Retention
Churn
Frequency
2000
1750
1500
1250
1000
750
500
250
0
0
1000000
2000000
3000000
4000000
cons_gas_12m


Retention
Churn
Frequency
12000
10000
8000
6000
4000
2000



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:

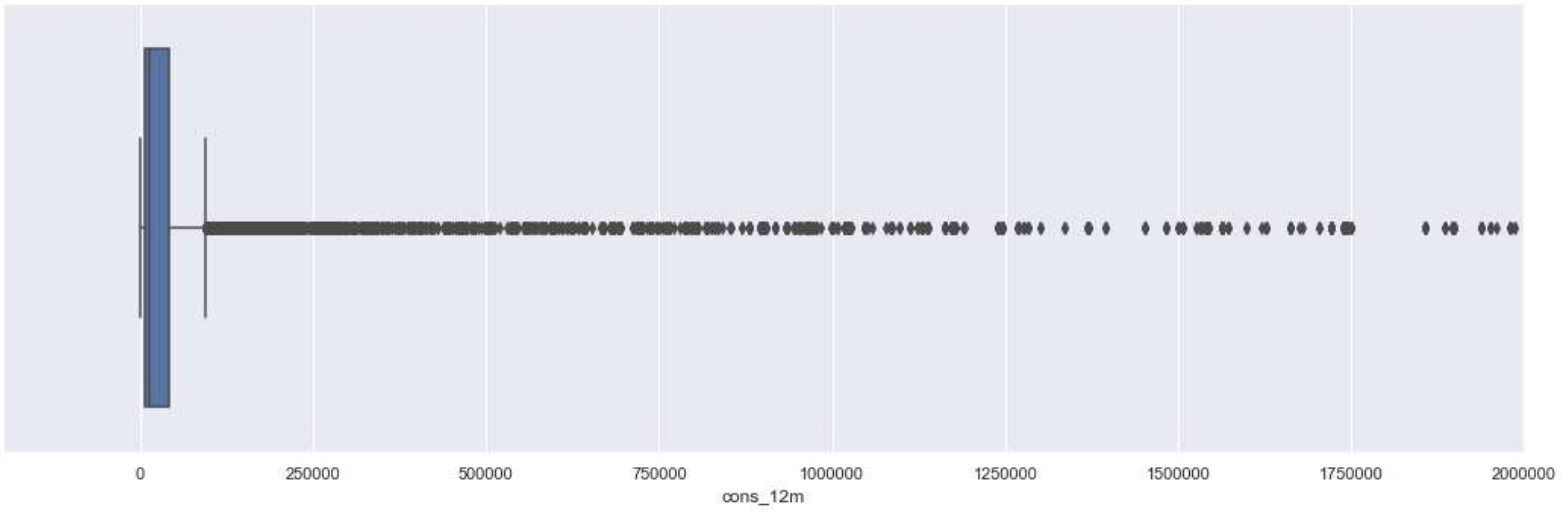
- Minimum
- First quartile (Q1)
- Median
- Third quartile (Q3)
- Maximum

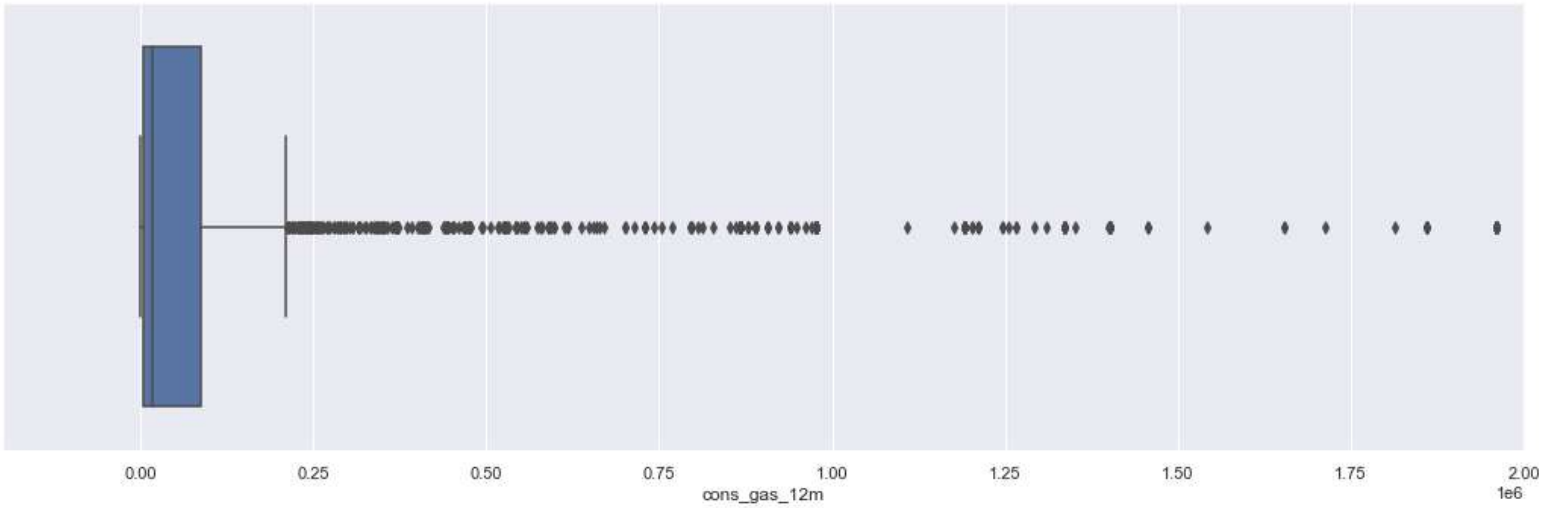
It can reveal outliers and what their values are. It can also tell us if our data is symmetrical, how tightly our data is grouped and if/how our data is skewed.

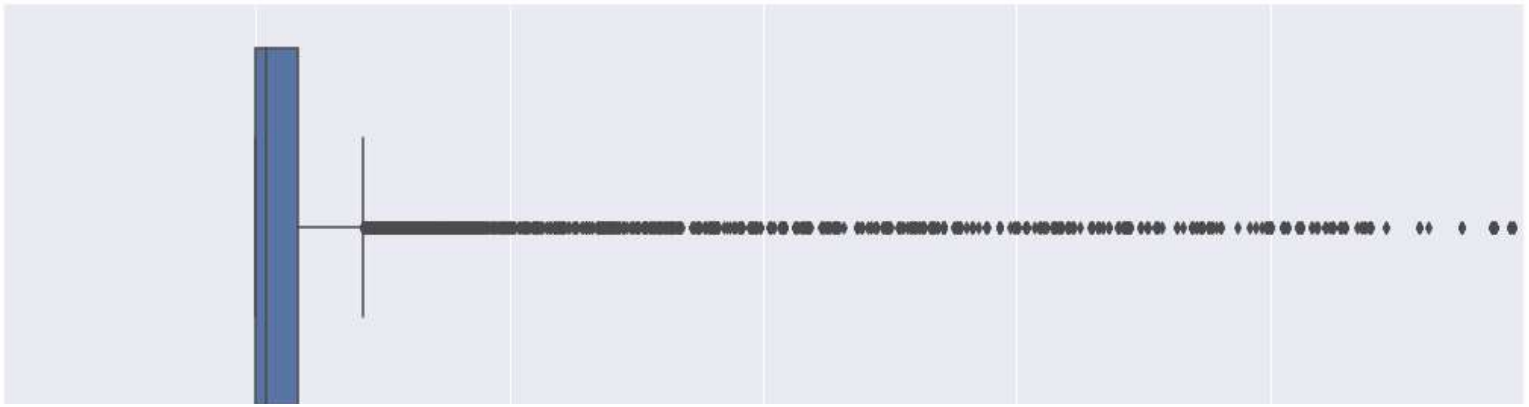
In [17]:

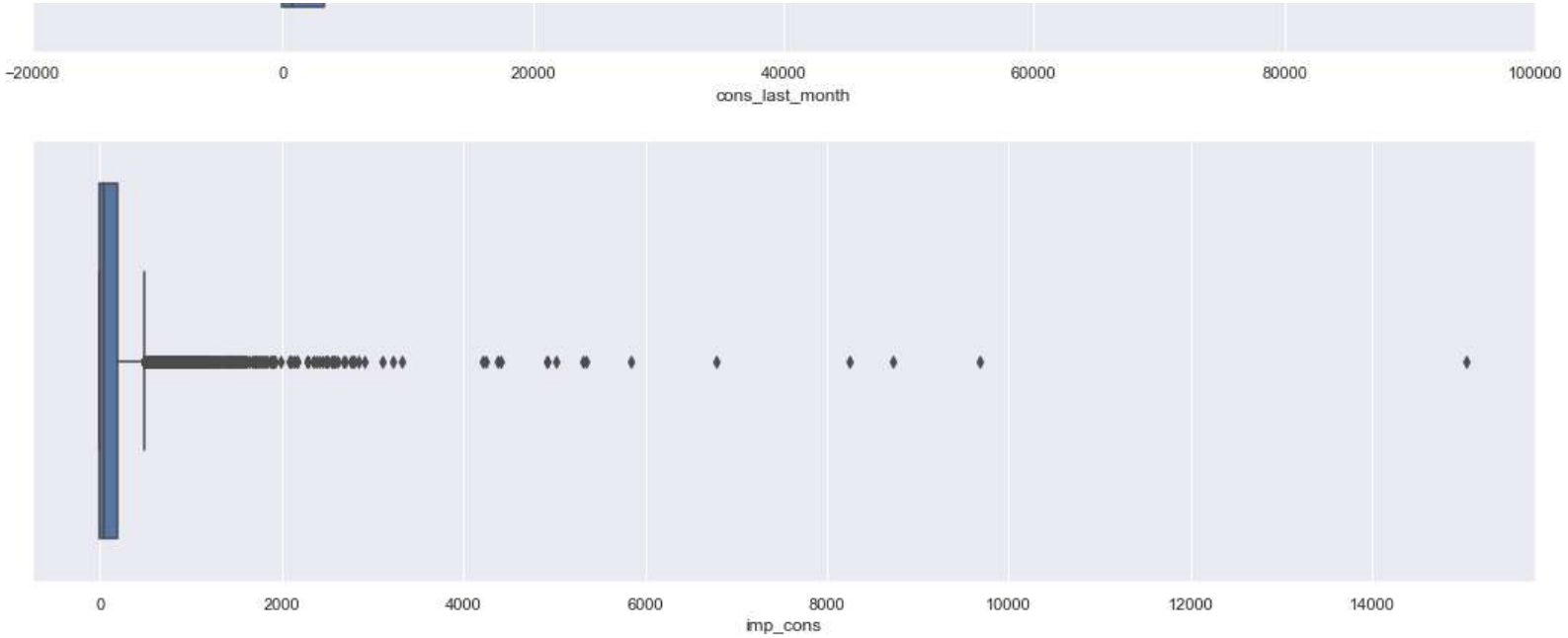
```
fig, axs = plt.subplots(nrows=4, figsize=(18,25))

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

# Remove scientific notation
for ax in axs:
    ax.ticklabel_format(style='plain', axis='x')
    # Set x-axis limit
    axs[0].set_xlim(-200000, 2000000)
    axs[1].set_xlim(-200000, 2000000)
    axs[2].set_xlim(-20000, 100000)
    plt.show()
```

```
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
```


0
250000
500000
750000
1000000
1250000
1500000
1750000
2000000
cons_12m


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



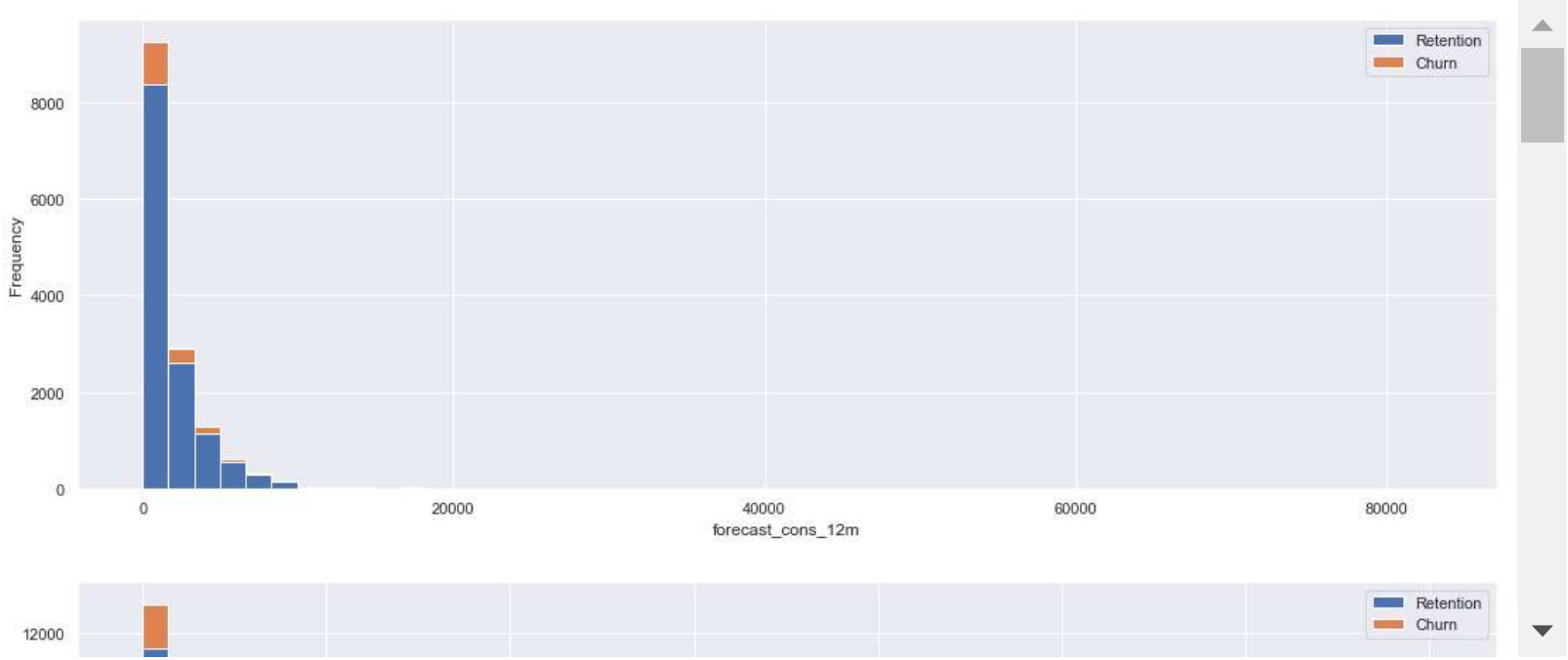


We will deal with skewness and outliers during feature engineering in the next exercise.

Forecast

In [18]:
```
forecast = client_df[
    ["id", "forecast_cons_12m",
    "forecast_cons_year","forecast_discount_energy","forecast_meter_rent_12m",
    "forecast_price_energy_p1","forecast_price_energy_p2",
    "forecast_price_pow_p1","churn"
    ]
]
```

In [19]:
```
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_p1", axs[4])
plot_distribution(client_df, "forecast_price_energy_p2", axs[5])
plot_distribution(client_df, "forecast_price_pow_p1", axs[6])
```

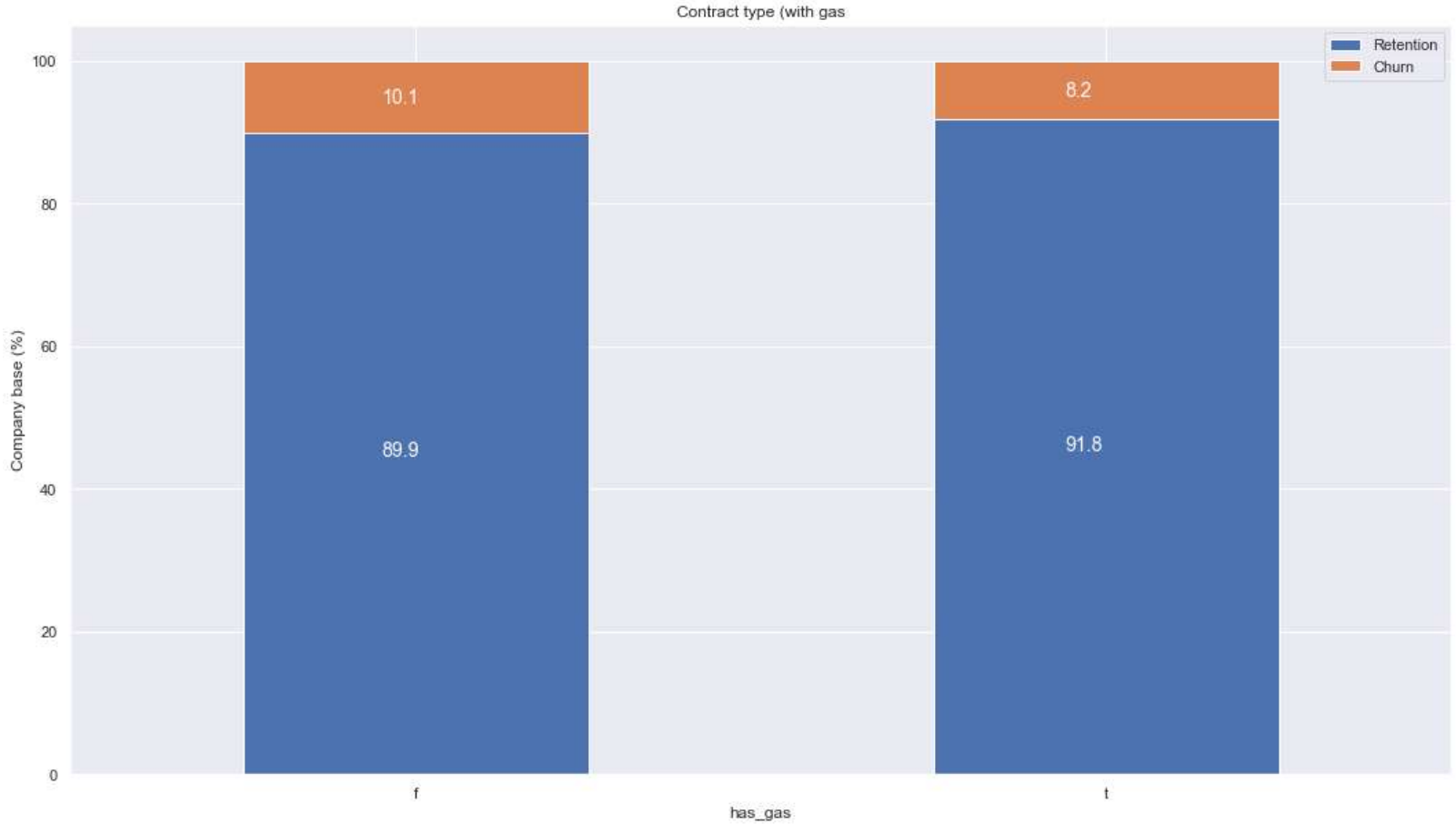


Similarly to the consumption plots, we can observe that a lot of the variables are highly positively skewed, creating a very long tail for the higher values. We will make some transformations during the next exercise to correct for this skewness.

Contract type

In [20]:
```
contract_type = client_df[['id', 'has_gas', 'churn']]
contract = contract_type.groupby([contract_type['churn'], contract_type['has_gas']])['id'].count().unstac
contract_percentage = (contract.div(contract.sum(axis=1), axis=0) * 100).sort_values(by=[1], ascending=Fa
```

In [21]:
```
plot_stacked_bars(contract_percentage, 'Contract type (with gas')
```



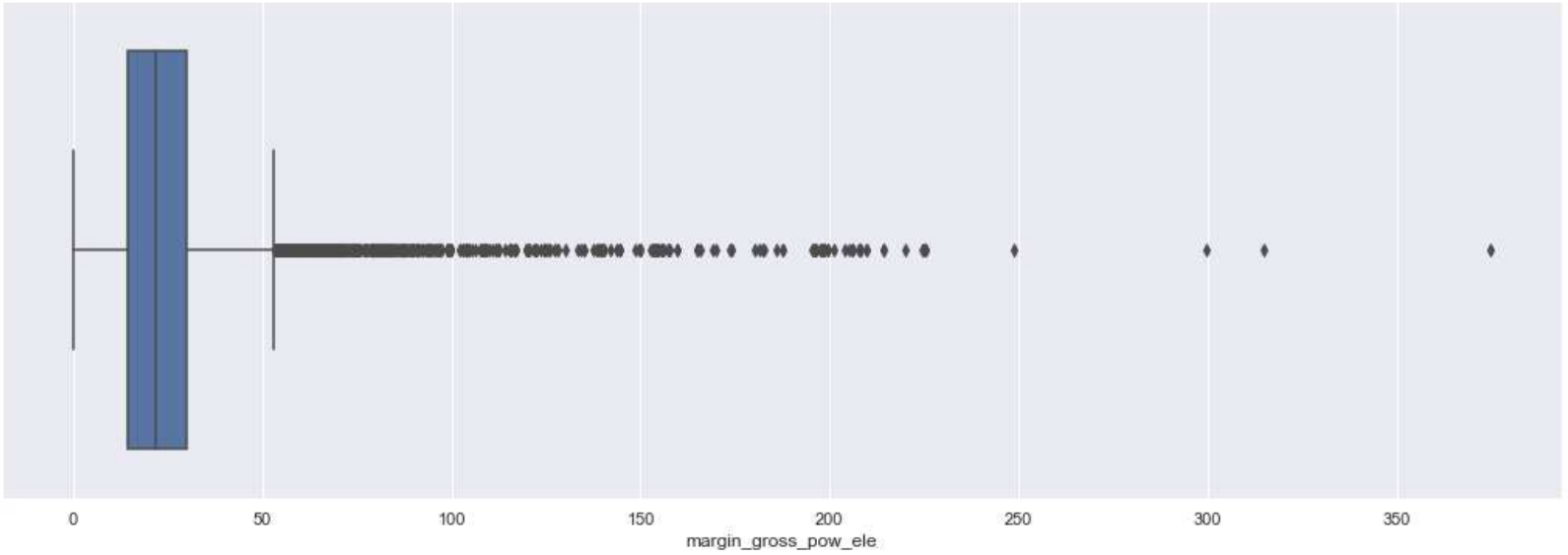
Margins

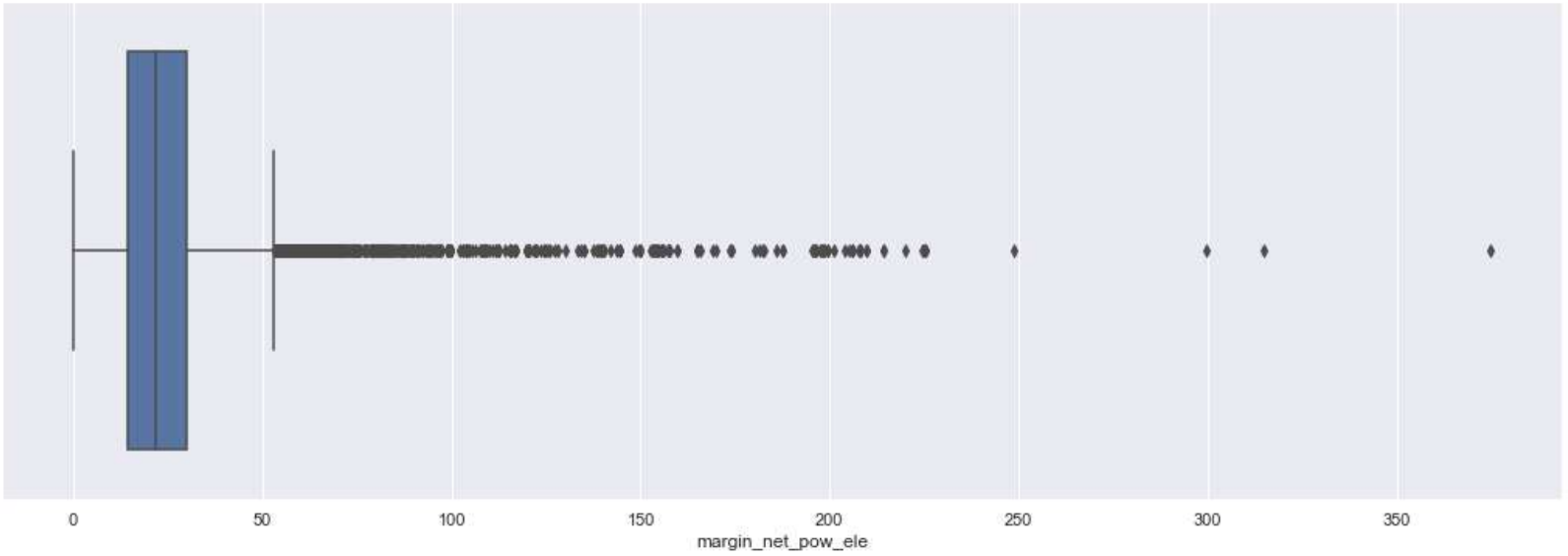
In [22]:
```
margin = client_df[['id', 'margin_gross_pow_ele', 'margin_net_pow_ele', 'net_margin']]
```

In [23]:

```
fig, axs = plt.subplots(nrows=3, figsize=(18,20))
# Plot histogram
sns.boxplot(margin["margin_gross_pow_ele"], ax=axs[0])
sns.boxplot(margin["margin_net_pow_ele"],ax=axs[1])
sns.boxplot(margin["net_margin"], ax=axs[2])
# Remove scientific notation
axs[0].ticklabel_format(style='plain', axis='x')
axs[1].ticklabel_format(style='plain', axis='x')
axs[2].ticklabel_format(style='plain', axis='x')
plt.show()
```

```
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
c:\Users\Arun\OneDrive\Documents\Halo\Forage\BCG Gamma Data Science\data\venv\lib\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keywor
d will result in an error or misinterpretation.
  FutureWarning
```


0
50
100
150
200
250
300
350
margin_gross_pow_ele

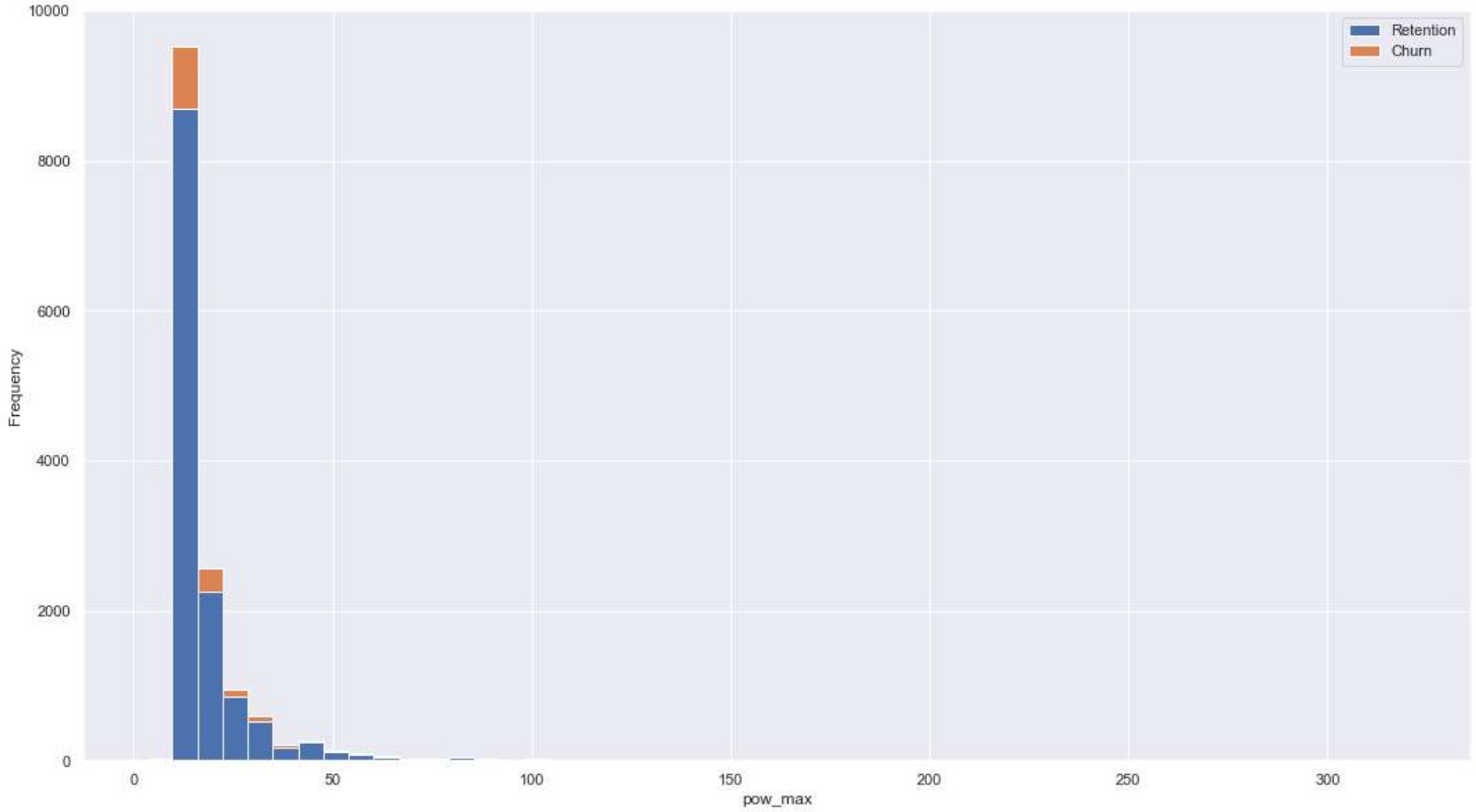

0
50
100
150
200
250
300
350
margin_net_pow_ele

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

Subscribed power

In [24]:
```
power = client_df[['id', 'pow_max', 'churn']]
```

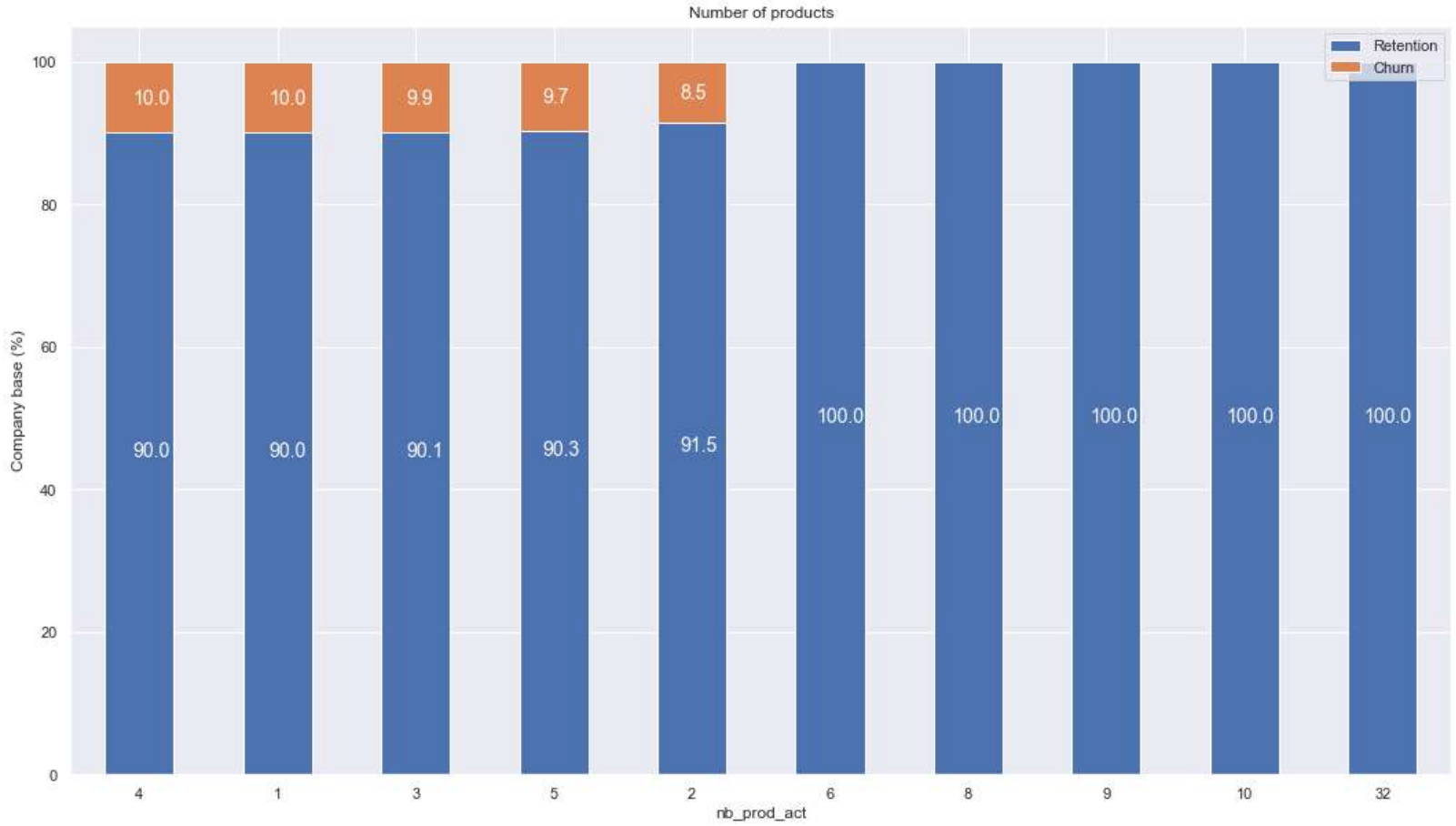
In [25]:
```
fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)
```



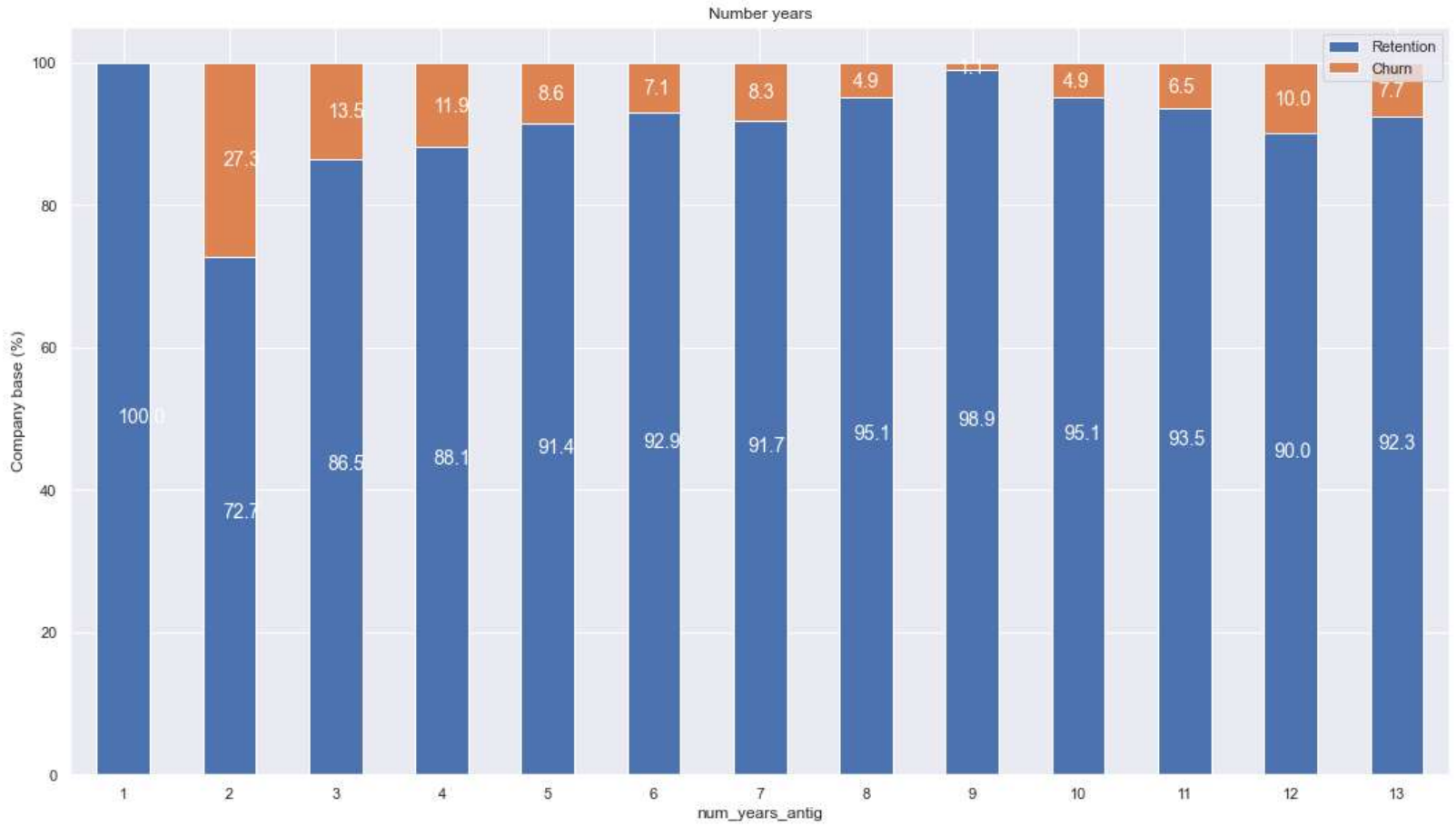
Other columns

In [26]:
```
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=Fals
```

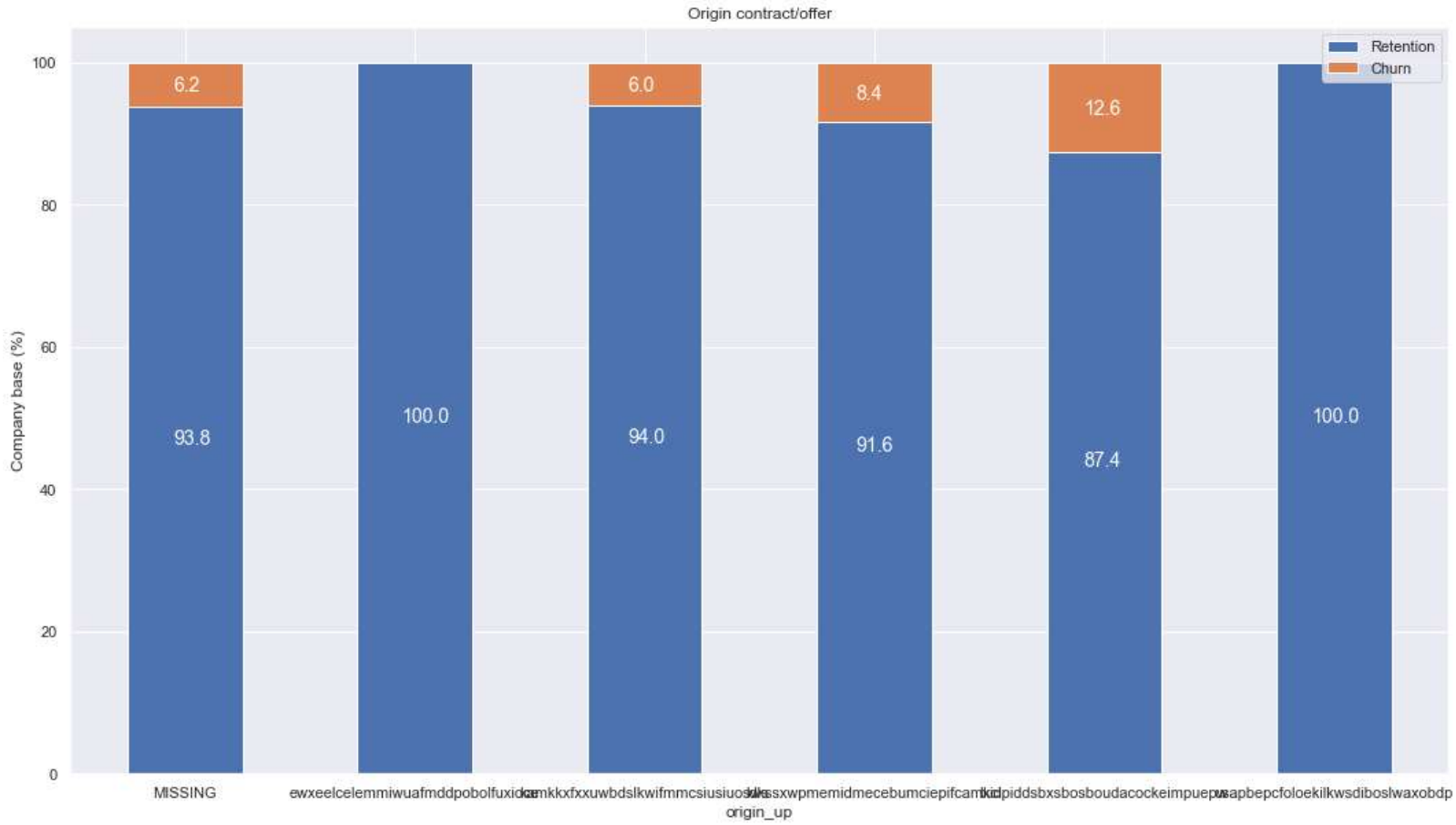
In [27]:
```
plot_stacked_bars(products_percentage, "Number of products")
```



In [28]:
```
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")
```



In [29]:
```
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")
```



5. Hypothesis investigation

Now that we have explored the data, it's time to investigate whether price sensitivity has some influence on churn. First we need to define exactly what is price sensitivity.

> Since we have the consumption data for each of the companies for the year of 2015, we will create new features to measure "price sensitivity" using the average of the year, the last 6 months and the last 3 months

In [30]:

```
# Transform date columns to datetime type
client_df["date_activ"] = pd.to_datetime(client_df["date_activ"], format='%Y-%m-%d')
client_df["date_end"] = pd.to_datetime(client_df["date_end"], format='%Y-%m-%d')
client_df["date_modif_prod"] = pd.to_datetime(client_df["date_modif_prod"], format='%Y-%m-%d')
client_df["date_renewal"] = pd.to_datetime(client_df["date_renewal"], format='%Y-%m-%d')
price_df['price_date'] = pd.to_datetime(price_df['price_date'], format='%Y-%m-%d')

# Create mean average data
mean_year = price_df.groupby(['id']).mean().reset_index()
mean_6m = price_df[price_df['price_date'] > '2015-06-01'].groupby(['id']).mean().reset_index()
mean_3m = price_df[price_df['price_date'] > '2015-10-01'].groupby(['id']).mean().reset_index()

# Comnbine into single dataframe
mean_year = mean_year.rename(
    index=str,
    columns={
        "price_p1_var": "mean_year_price_p1_var",
        "price_p2_var": "mean_year_price_p2_var",
        "price_p3_var": "mean_year_price_p3_var",
        "price_p1_fix": "mean_year_price_p1_fix",
        "price_p2_fix": "mean_year_price_p2_fix",
        "price_p3_fix": "mean_year_price_p3_fix"
    }
)

mean_year["mean_year_price_p1"] = mean_year["mean_year_price_p1_var"] + mean_year["mean_year_price_p1_fix
mean_year["mean_year_price_p2"] = mean_year["mean_year_price_p2_var"] + mean_year["mean_year_price_p2_fix
mean_year["mean_year_price_p3"] = mean_year["mean_year_price_p3_var"] + mean_year["mean_year_price_p3_fix

mean_6m = mean_6m.rename(
    index=str,
    columns={
        "price_p1_var": "mean_6m_price_p1_var",
        "price_p2_var": "mean_6m_price_p2_var",
        "price_p3_var": "mean_6m_price_p3_var",
        "price_p1_fix": "mean_6m_price_p1_fix",
        "price_p2_fix": "mean_6m_price_p2_fix",
        "price_p3_fix": "mean_6m_price_p3_fix"
    }
)
mean_6m["mean_6m_price_p1"] = mean_6m["mean_6m_price_p1_var"] + mean_6m["mean_6m_price_p1_fix"]
mean_6m["mean_6m_price_p2"] = mean_6m["mean_6m_price_p2_var"] + mean_6m["mean_6m_price_p2_fix"]
mean_6m["mean_6m_price_p3"] = mean_6m["mean_6m_price_p3_var"] + mean_6m["mean_6m_price_p3_fix"]
```

```
mean_3m = mean_3m.rename(
    index=str,
    columns={
        "price_p1_var": "mean_3m_price_p1_var",
        "price_p2_var": "mean_3m_price_p2_var",
        "price_p3_var": "mean_3m_price_p3_var",
        "price_p1_fix": "mean_3m_price_p1_fix",
        "price_p2_fix": "mean_3m_price_p2_fix",
        "price_p3_fix": "mean_3m_price_p3_fix"
    }
)
mean_3m["mean_3m_price_p1"] = mean_3m["mean_3m_price_p1_var"] + mean_3m["mean_3m_price_p1_fix"]
mean_3m["mean_3m_price_p2"] = mean_3m["mean_3m_price_p2_var"] + mean_3m["mean_3m_price_p2_fix"]
mean_3m["mean_3m_price_p3"] = mean_3m["mean_3m_price_p3_var"] + mean_3m["mean_3m_price_p3_fix"]

# Merge into 1 dataframe
price_features = pd.merge(mean_year, mean_6m, on='id')
price_features = pd.merge(price_features, mean_3m, on='id')
```

In [31]:
```
price_features.head()
```

Out[31]:

	id	mean_year_price_p1_var	mean_year_price_p2_var	mean_year_price_p3_var	mean_year_pric
0	0002203ffbb812588b632b9e628cc38d	0.124338	0.103794	0.073160	4(
1	0004351ebdd665e6ee664792efc4fd13	0.146426	0.000000	0.000000	4-
2	0010bcc39e42b3c2131ed2ce55246e3c	0.181558	0.000000	0.000000	4!
3	0010ee3855fdea87602a5b7aba8e42de	0.118757	0.098292	0.069032	4(
4	00114d74e963e47177db89bc70108537	0.147926	0.000000	0.000000	4-

5 rows × 28 columns

Now lets merge in the churn data and see whether price sensitivity has any correlation with churn

In [32]:
```
price_analysis = pd.merge(price_features, client_df[['id', 'churn']], on='id')
price_analysis.head()
```

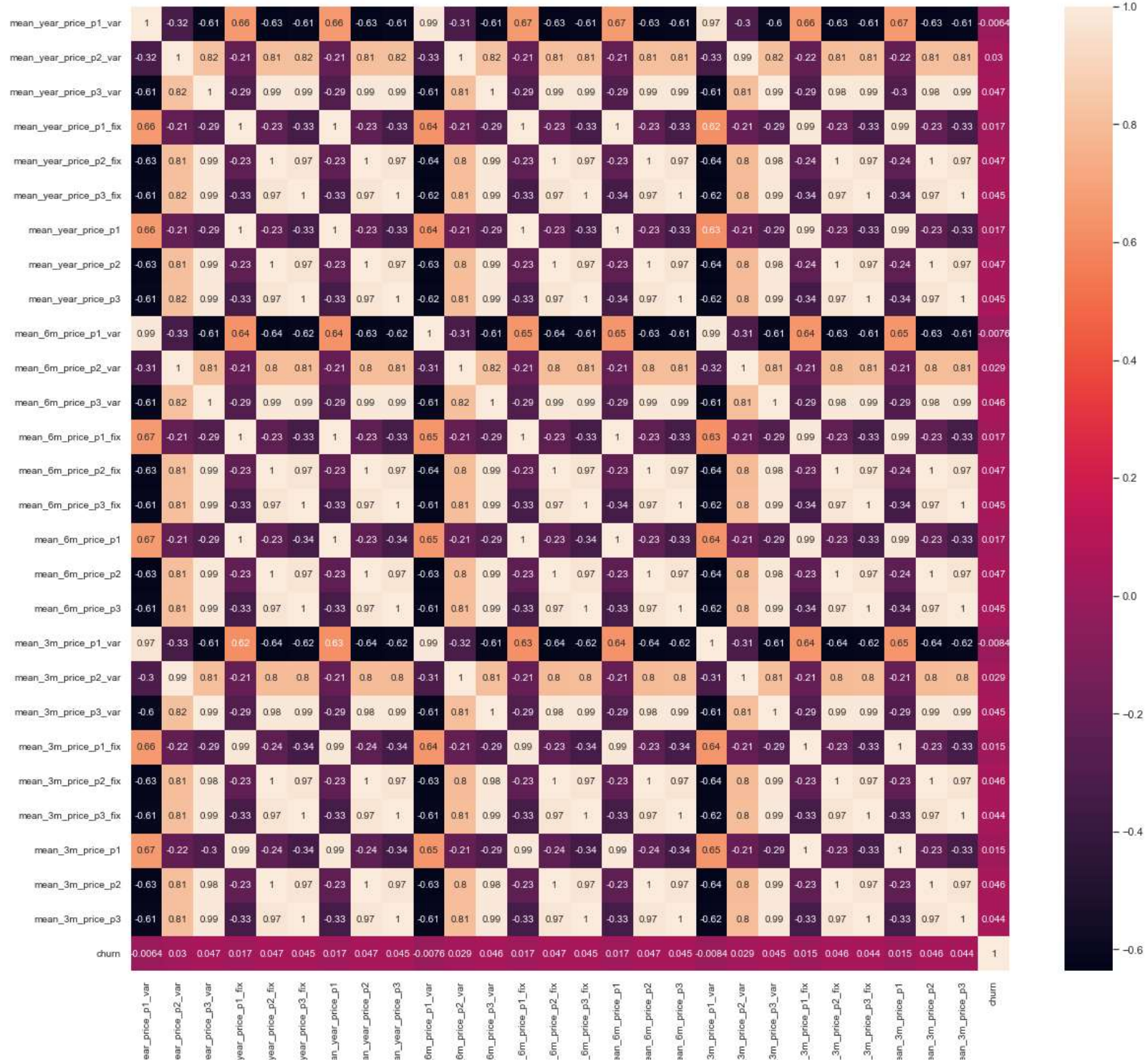
Out[32]:

	id	mean_year_price_p1_var	mean_year_price_p2_var	mean_year_price_p3_var	mean_year_pric
0	0002203ffbb812588b632b9e628cc38d	0.124338	0.103794	0.073160	4(
1	0004351ebdd665e6ee664792efc4fd13	0.146426	0.000000	0.000000	4
2	0010bcc39e42b3c2131ed2ce55246e3c	0.181558	0.000000	0.000000	4
3	00114d74e963e47177db89bc70108537	0.147926	0.000000	0.000000	4
4	0013f326a839a2f6ad87a1859952d227	0.126076	0.105542	0.074921	4(

5 rows × 29 columns

In [33]:

```
corr = price_analysis.corr()
# Plot correlation
plt.figure(figsize=(20,18))
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_k
# Axis ticks size
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```


mean_year_price_p1_var
mean_year_price_p2_var
mean_year_price_p3_var
mean_year_price_p1_fix
mean_year_price_p2_fix
mean_year_price_p3_fix
mean_year_price_p1
mean_year_price_p2
mean_year_price_p3
mean_6m_price_p1_var
mean_6m_price_p2_var
mean_6m_price_p3_var
mean_6m_price_p1_fix
mean_6m_price_p2_fix
mean_6m_price_p3_fix
mean_6m_price_p1
mean_6m_price_p2
mean_6m_price_p3
mean_3m_price_p1_var
mean_3m_price_p2_var
mean_3m_price_p3_var
mean_3m_price_p1_fix
mean_3m_price_p2_fix
mean_3m_price_p3_fix
mean_3m_price_p1
mean_3m_price_p2
mean_3m_price_p3
churn
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

From the correlation plot, it shows a higher magnitude of correlation between other price sensitivity variables, however overall the correlation with churn is very low. This indicates that there is a weak linear relationship between price sensitity and churn. This suggests that for price sensivity to be a major driver for predicting churn, we may need to engineer the feature differently.

In [34]:
```
merged_data = pd.merge(client_df.drop(columns=['churn']), price_analysis, on='id')
```

In [35]:
```
merged_data.head()
```

Out[35]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	2010-03-30
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526	2010-01-13

5 rows × 53 columns

In [36]:
```
merged_data.to_csv('clean_data_after_eda.csv')
```