Exploratory Data Analysis

Import packages

In [1]:
```
# Data analysis and wrangling
import pandas as pd
import numpy as np

# Data visualisation
import seaborn as sns
import matplotlib.pyplot as plt

# Shows plots in jupyter notebook
%matplotlib inline

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

# Remove warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading data & Viewing it with Pandas

In [2]:
```
## Importing the DataSet
client_df = pd.read_csv(r"C:\Users\lenovo\Desktop\4.1 Client Data.csv")
price_df = pd.read_csv(r"C:\Users\lenovo\Desktop\4.2 Price Data.csv")
```

You can view the first 5 rows of a dataframe using the `head` method. Similarly, if you wanted to see the last 5, you can use `tail(3)`

In [3]:
```
## Viewing the Client Data
client_df.head(5)
```

Out[3]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526

5 rows × 26 columns

In [4]:
```
## Viewing the Price Data
price_df.head(5)
```

Out[4]:

	id	price_date	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.0	44
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.0	44
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.0	44
3	038af19179925da21a25619c5a24b745	2015-04-01	0.149626	0.0	0.0	44
4	038af19179925da21a25619c5a24b745	2015-05-01	0.149626	0.0	0.0	44

Data Description for

client_data.csv

- id = client company identifier
- activity_new = category of the company's activity
- channel_sales = code of the sales channel
- cons_12m = electricity consumption of the past 12 months
- cons_gas_12m = gas consumption of the past 12 months
- cons_last_month = electricity consumption of the last month
- date_activ = date of activation of the contract
- date_end = registered date of the end of the contract
- date_modif_prod = date of the last modification of the product
- date_renewal = date of the next contract renewal
- forecast_cons_12m = forecasted electricity consumption for next 12 months
- forecast_cons_year = forecasted electricity consumption for the next calendar year
- forecast_discount_energy = forecasted value of current discount
- forecast_meter_rent_12m = forecasted bill of meter rental for the next 2 months
- forecast_price_energy_off_peak = forecasted energy price for 1st period (off peak)
- forecast_price_energy_peak = forecasted energy price for 2nd period (peak)
- forecast_price_pow_off_peak = forecasted power price for 1st period (off peak)
- has_gas = indicated if client is also a gas client
- imp_cons = current paid consumption
- margin_gross_pow_ele = gross margin on power subscription
- margin_net_pow_ele = net margin on power subscription
- nb_prod_act = number of active products and services
- net_margin = total net margin
- num_years_antig = antiquity of the client (in number of years)
- origin_up = code of the electricity campaign the customer first subscribed to
- pow_max = subscribed power
- churn = has the client churned over the next 3 months

price_data.csv

- id = client company identifier
- price_date = reference date
- price_off_peak_var = price of energy for the 1st period (off peak)
- price_peak_var = price of energy for the 2nd period (peak)
- price_mid_peak_var = price of energy for the 3rd period (mid peak)

- price_off_peak_fix = price of power for the 1st period (off peak)
- price_peak_fix = price of power for the 2nd period (peak)
- price_mid_peak_fix = price of power for the 3rd period (mid peak)

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.

In [5]:
```
## Checking the info of client 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
```

The datatype of date_activ, date_end, date_modif_prod and date_renewal is object, so we need to convert them to datetime.

In [6]:
```
## Converting colomns to datetime
date=["date_activ", "date_end", "date_modif_prod", "date_renewal"]
```

In [7]:
```
for i in date:
    client_df[i] = pd.to_datetime(client_df[i])
```

In [8]:
```
## Checking the info of client data after converting
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  datetime64[ns]
 6   date_end                        14606 non-null  datetime64[ns]
 7   date_modif_prod                 14606 non-null  datetime64[ns]
 8   date_renewal                    14606 non-null  datetime64[ns]
 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: datetime64[ns](4), float64(11), int64(7), object(4)
memory usage: 2.9+ MB
```

In [9]:
```
## Checking the info of price data
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
```

In [10]:
```
price_df["price_date"] = pd.to_datetime(price_df["price_date"])
```

In [11]:
```
## see the data shape of dataset
client_df.shape
```

Out[11]: (14606, 26)

In [12]:
```
price_df.shape
```

Out[12]: (193002, 8)

Statistics

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

In [13]:
```
## Checking Statistics for Client Data
client_df.describe()
```

Out[13]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_ene
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	14606.000
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.762906	0.966
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.786255	5.108
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.000000	0.000
50%	1.411550e+04	0.000000e+00	792.500000	1112.875000	314.000000	0.000
75%	4.076375e+04	0.000000e+00	3383.000000	2401.790000	1745.750000	0.000
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	175375.000000	30.000

In [14]:
```
## Checking Statistics for Price Data
price_df.describe()
```

Out[14]:

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_mid_peak_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

Missing Data

Missing values can also be checked with { data.isnull.sum() } but here we are finding the % of missing values

In [15]:
```
## Check how much of our data is missing in client data
pd.DataFrame({"Missing value (%)": client_df.isnull().sum()/len(client_df.index)*100})
```

Out[15]:

	Missing value (%)
id	0.0
channel_sales	0.0
cons_12m	0.0
cons_gas_12m	0.0
cons_last_month	0.0
date_activ	0.0
date_end	0.0
date_modif_prod	0.0
date_renewal	0.0
forecast_cons_12m	0.0
forecast_cons_year	0.0
forecast_discount_energy	0.0
forecast_meter_rent_12m	0.0
forecast_price_energy_off_peak	0.0
forecast_price_energy_peak	0.0
forecast_price_pow_off_peak	0.0
has_gas	0.0
imp_cons	0.0
margin_gross_pow_ele	0.0
margin_net_pow_ele	0.0
nb_prod_act	0.0
net_margin	0.0
num_years_antig	0.0
origin_up	0.0
pow_max	0.0
churn	0.0

In [16]:
```
## ## Check how much of our data is missing in price data
pd.DataFrame({"Missing value (%)": price_df.isnull().sum()/len(price_df.index)*100})
```

Out[16]:

	Missing value (%)
id	0.0
price_date	0.0
price_off_peak_var	0.0
price_peak_var	0.0
price_mid_peak_var	0.0
price_off_peak_fix	0.0
price_peak_fix	0.0
price_mid_peak_fix	0.0

Their are no missing values in both i.e. Client & Price dataset.

Data visualization

Client Data

Churn Percentage

In [17]:
```
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 [18]:
```
churn_percentage
```

Out[18]:

churn	Companies
0	90.284814
1	9.715186

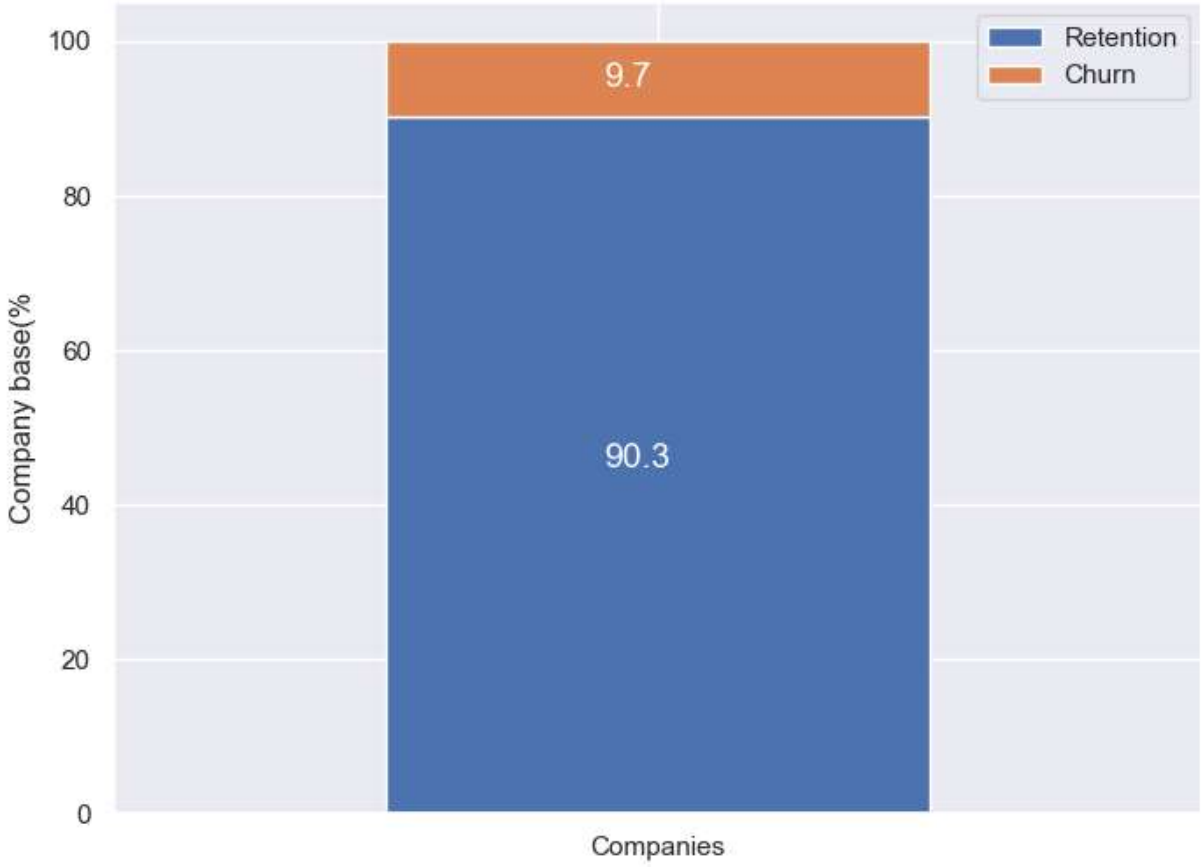
In [19]:
```
def annotate_stacked_bars(ax, pad=0.99, colour="white", textsize=13):
    """
    Add value annotations to the bars
    """

    # 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
        )
```

In [20]:
```
ax=churn_percentage.transpose().plot(kind='bar',stacked=True,figsize=(8,6),rot=0)
annotate_stacked_bars(ax, textsize=14)
plt.legend(['Retention','Churn'],loc='upper right')
plt.ylabel('Company base(%')
```

Out[20]:
```
Text(0, 0.5, 'Company base(%')
```



About 10% of total customers have churned, it is imbalanced dataset, so in the modeling process, we need to handle this problem.

Sale Channels

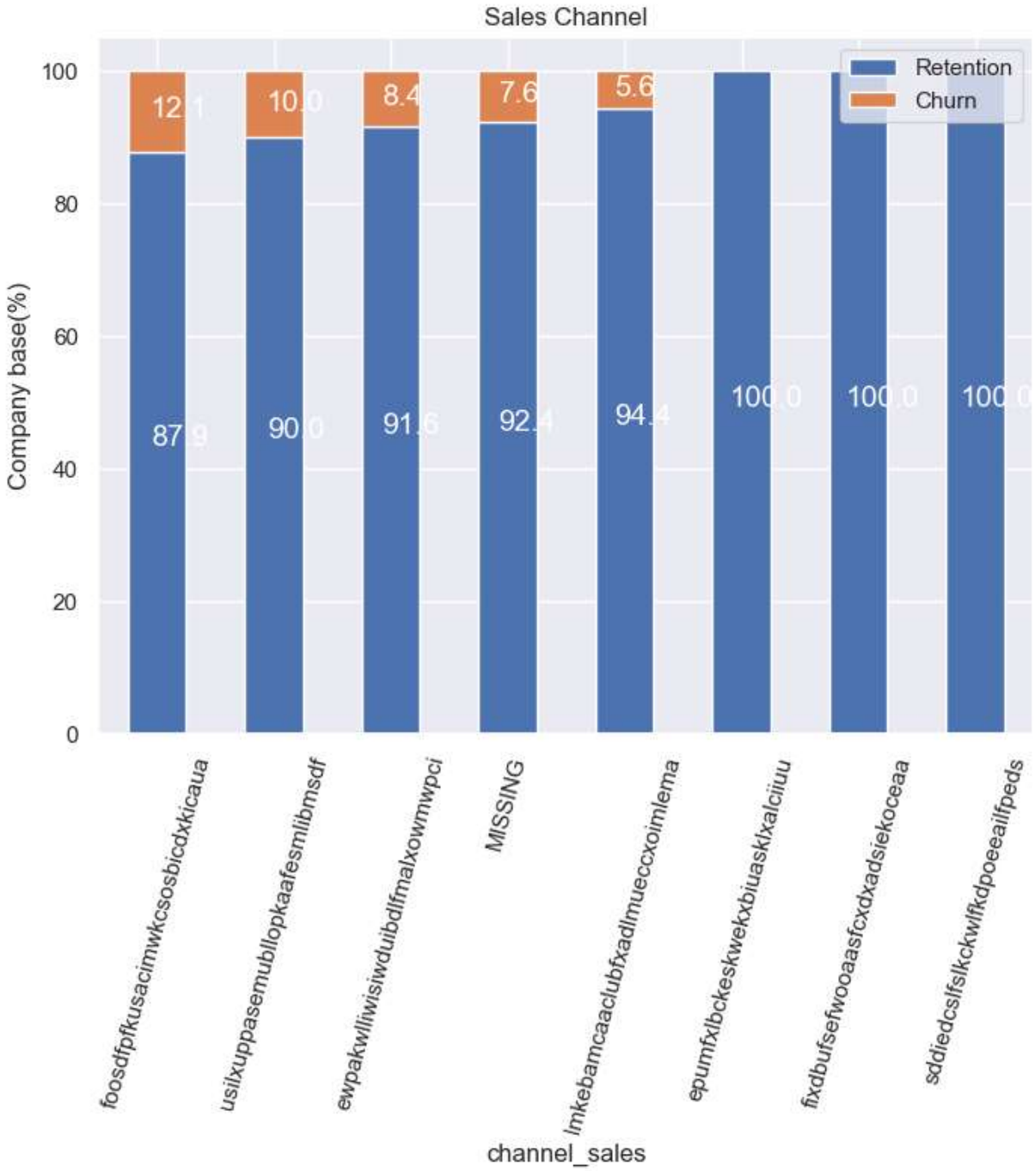
Sale channels can play an important role in customer's rentation or churn. Some strong sale channels can create quality and loyal customers.

In [21]:
```
channel=client_df[['id', 'channel_sales', 'churn']]
channel = channel.groupby([channel['channel_sales'], channel['churn']])['id'].count().unstack(
```

In [22]:
```
channel_churn=(channel.div(channel.sum(axis=1),axis=0)*100).sort_values(by=[1],ascending=False
```

In [23]:
```
ax=channel_churn.plot(kind='bar',stacked=True,figsize=(8,6),rot=75)
annotate_stacked_bars(ax, textsize=14)
plt.title('Sales Channel')
plt.legend(['Retention','Churn'],loc='upper right')
plt.ylabel('Company base(%)')
```

Out[23]:
```
Text(0, 0.5, 'Company base(%)')
```



There are 5 Sale channels having churn rate.

Consumption

Now, we will see the consumption of customers during the last month and 12 months againt the churn. Because this is a numeric - categorical comparison therefore we can use histogram distribution or box plot

visualization

In [24]:
```
consumption = client_df[["cons_12m", "cons_gas_12m", "cons_last_month", "imp_cons"]]
```

In [25]:
```
for col in consumption:
    client_df.boxplot(column=col, by='churn', figsize=(6,6))
    plt.title(col)
plt.show()
```

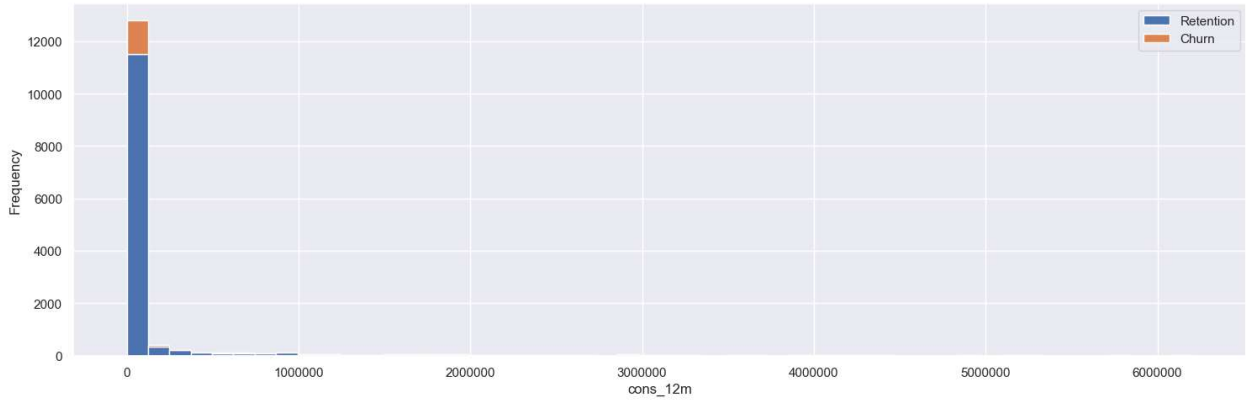
We will create a plot_distribution function to see the spread and skewness of the data.

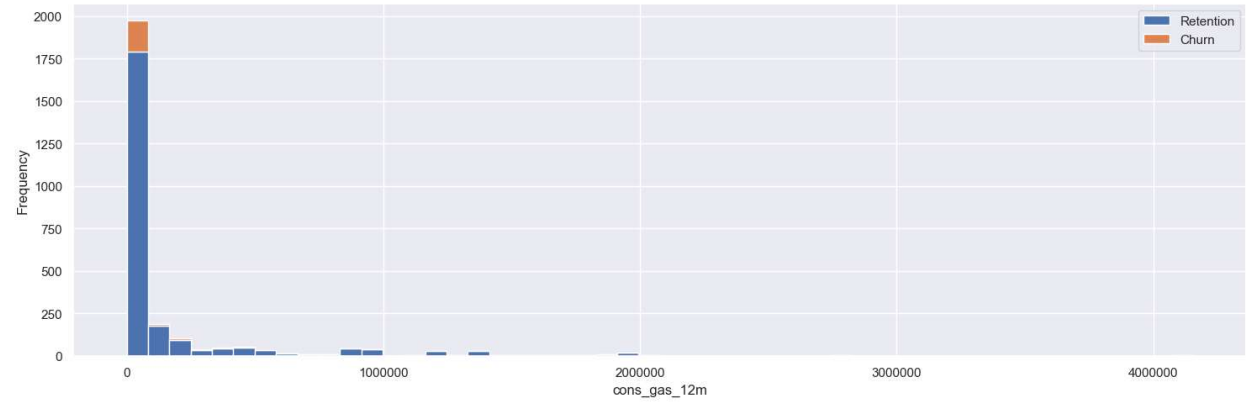
In [26]:
```
consumption = client_df[["cons_12m", "cons_gas_12m", "cons_last_month", "imp_cons", "has_gas",
```

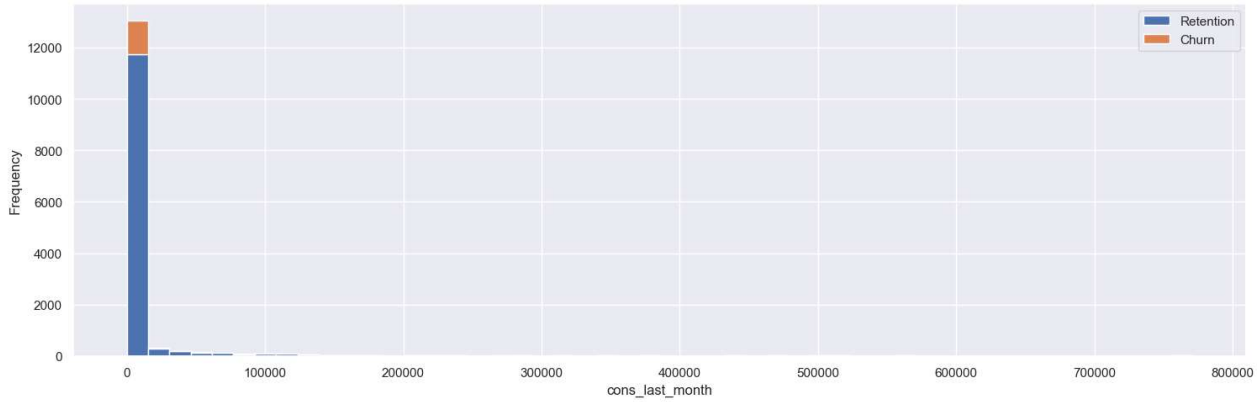
In [27]:
```
def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distirbution in a stacked histogram of churned or retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "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')
```

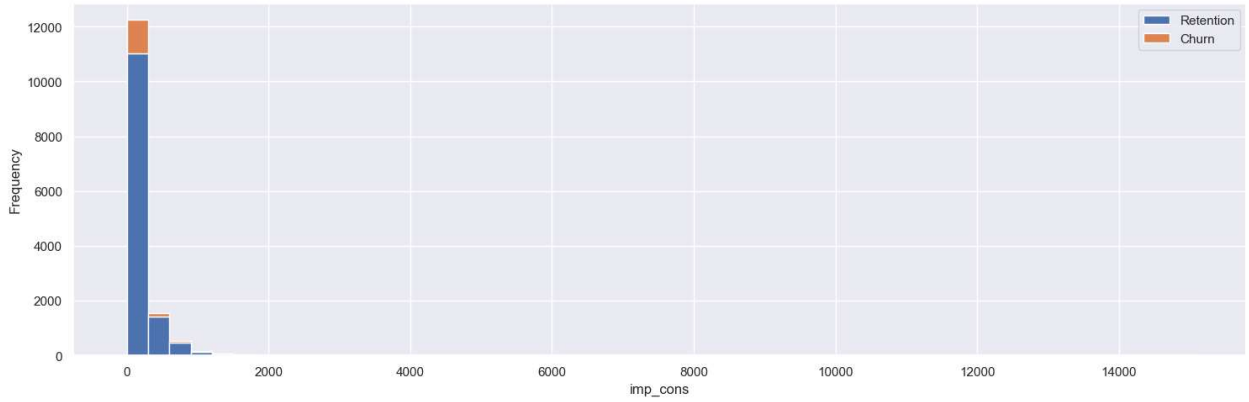
In [28]:
```
fig, axs = plt.subplots(nrows=4, figsize=(18, 25))

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])
```







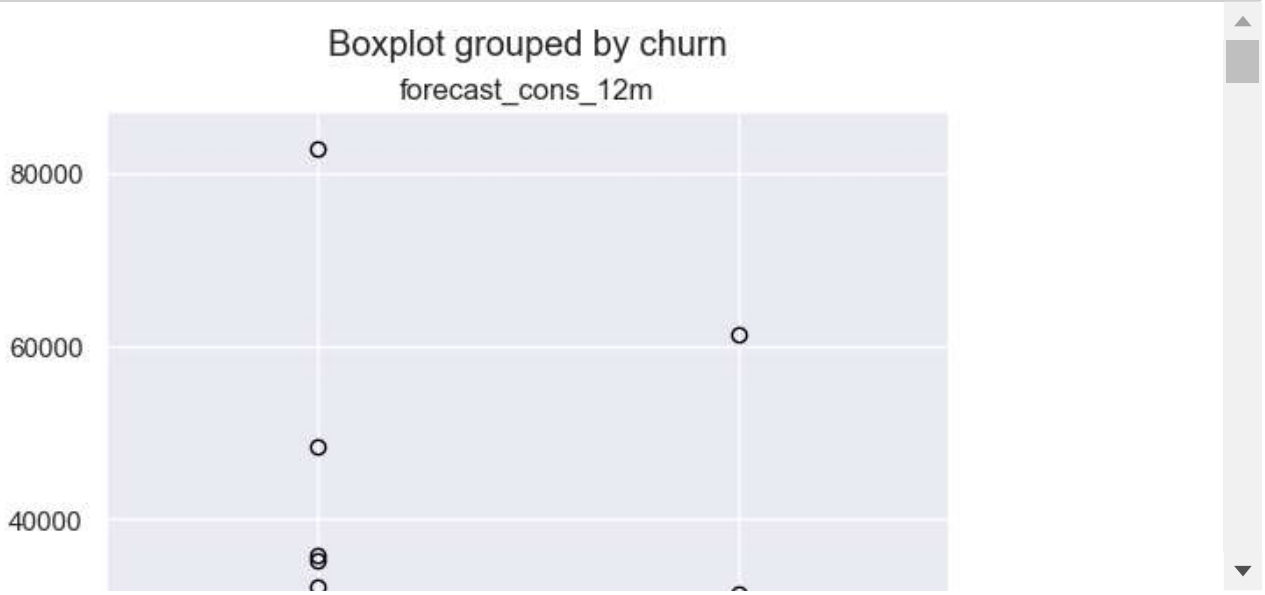


There are a lot of outliers at the upper in the dataset and it is highly positive skewed. we'll address skewness and outliers in the feature engineering.

Forecast

In [29]:
```
forecast = client_df[["forecast_cons_12m", "forecast_cons_year","forecast_discount_energy","fo
    "forecast_price_energy_off_peak","forecast_price_energy_peak", "forecast_price_pow_off_peal
```

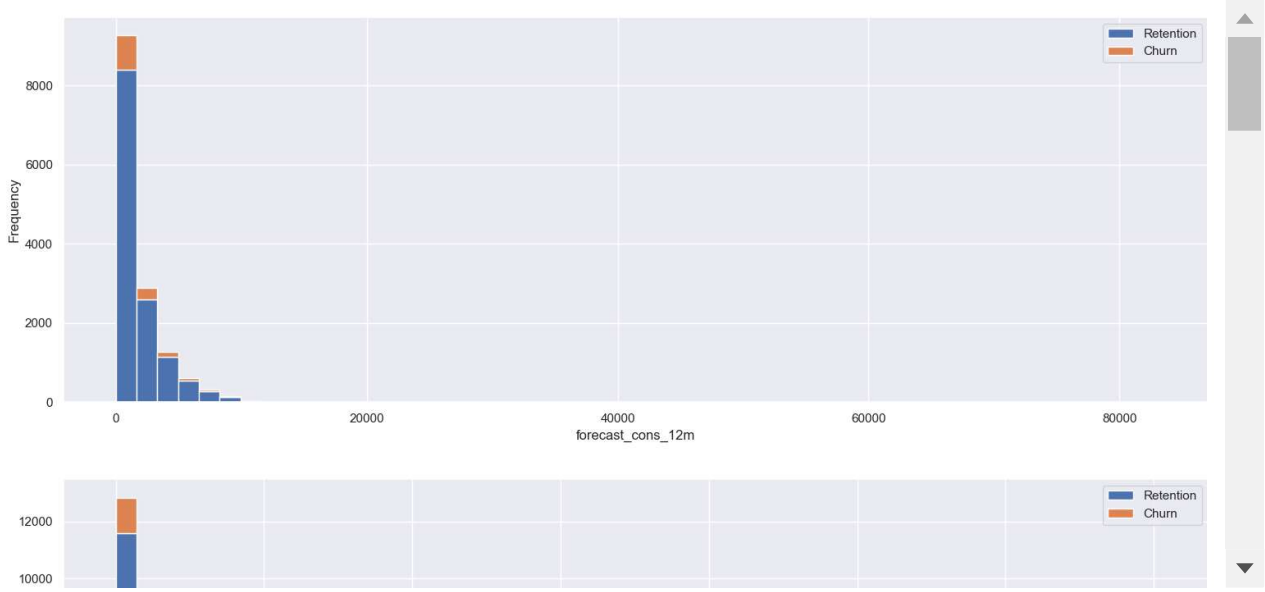
In [30]:
```
for col in forecast:
    client_df.boxplot(column=col, by='churn', figsize=(6,6))
    plt.title(col)
plt.show()
```



Similar to consumption, forecast variables also show postive skewed.

In [31]:
```
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])
```

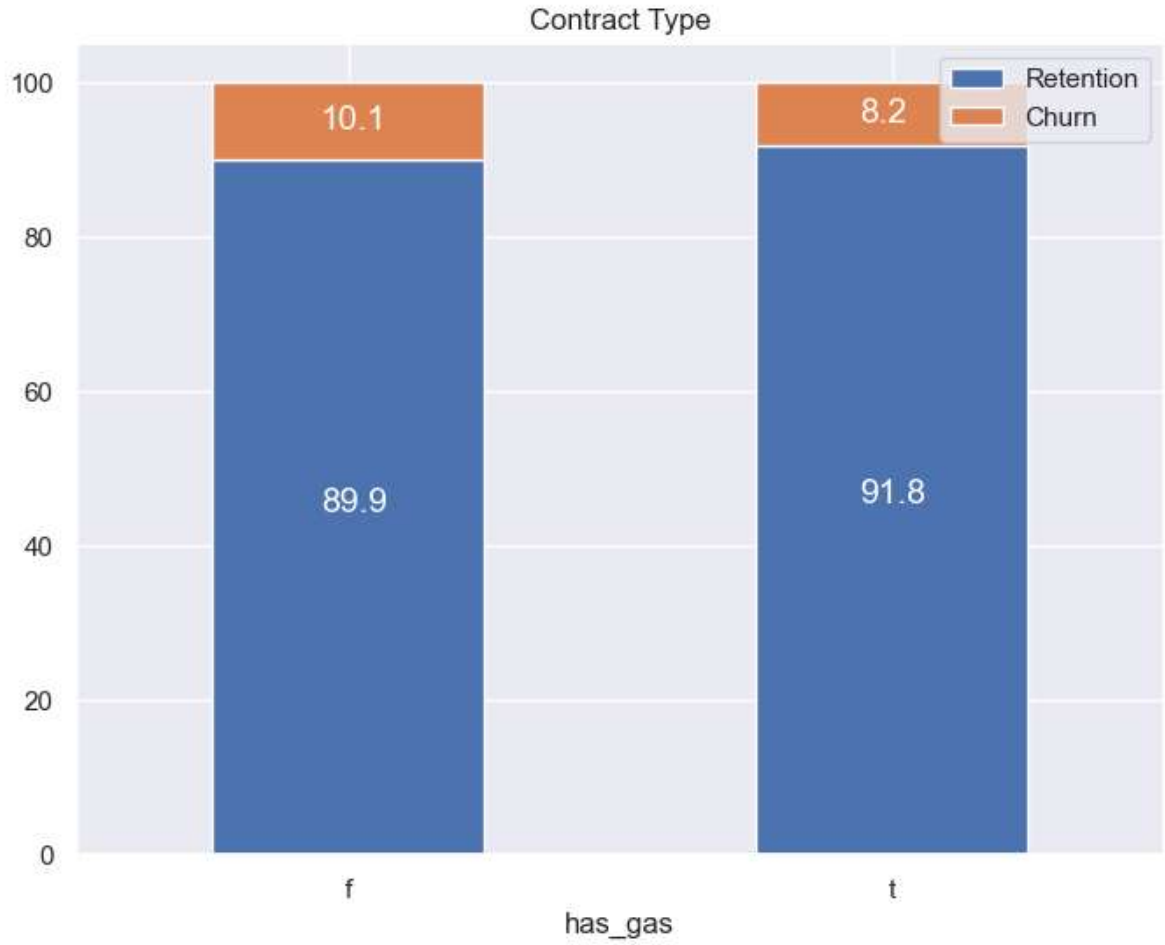


Contract type

In [32]:
```
contract_type = client_df[["id","has_gas","churn"]]
contract = contract_type.groupby([contract_type["churn"], contract_type["has_gas"]])["id"].cou
contract_percentage = (contract.div(contract.sum(axis=1), axis=0) * 100).sort_values(by=[1], a
```

In [33]:
```
ax=contract_percentage.plot(kind='bar',stacked=True,figsize=(8,6),rot=0)
annotate_stacked_bars(ax, textsize=14)
plt.legend(['Retention','Churn'],loc="upper right")
plt.title("Contract Type")
```

Out[33]:
```
Text(0.5, 1.0, 'Contract Type')
```

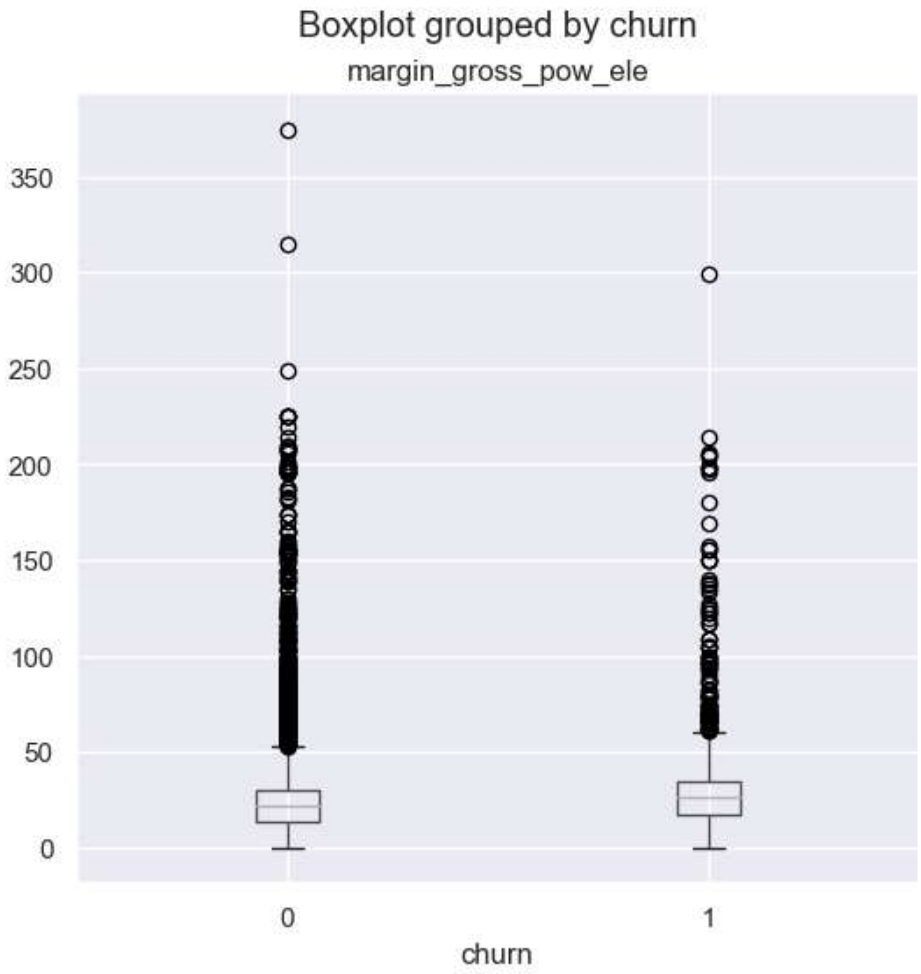


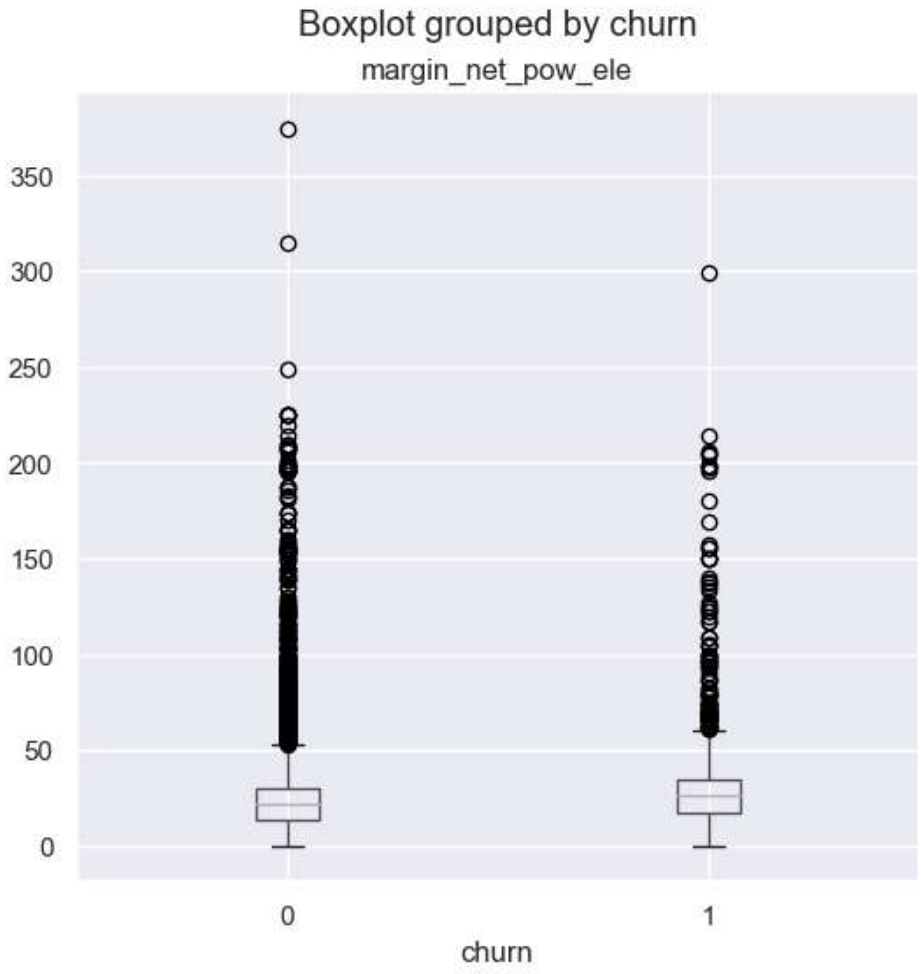
The churn rate for customer without contract is a little bit higher than customers with contract.

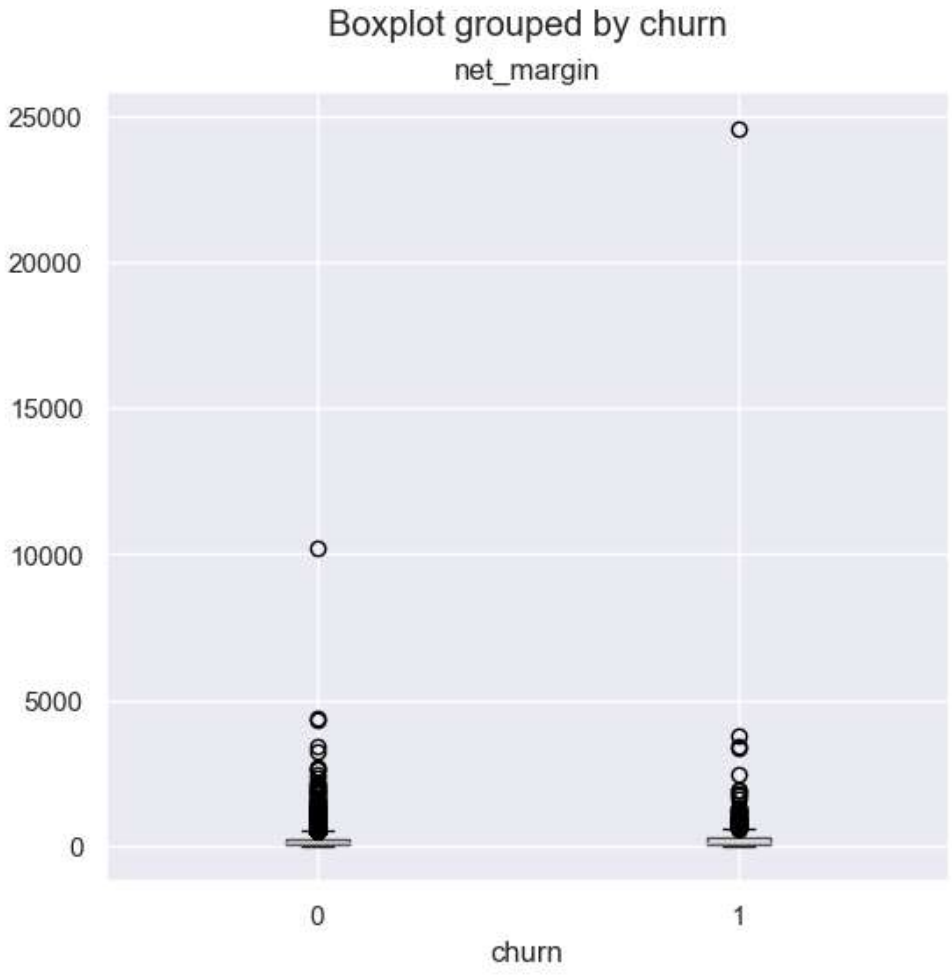
Margins

In [34]:
```
margin = client_df[["margin_gross_pow_ele", "margin_net_pow_ele", "net_margin"]]
```

In [35]:
```
for col in margin:
    client_df.boxplot(column=col, by='churn', figsize=(6,6))
    plt.title(col)
plt.show()
```



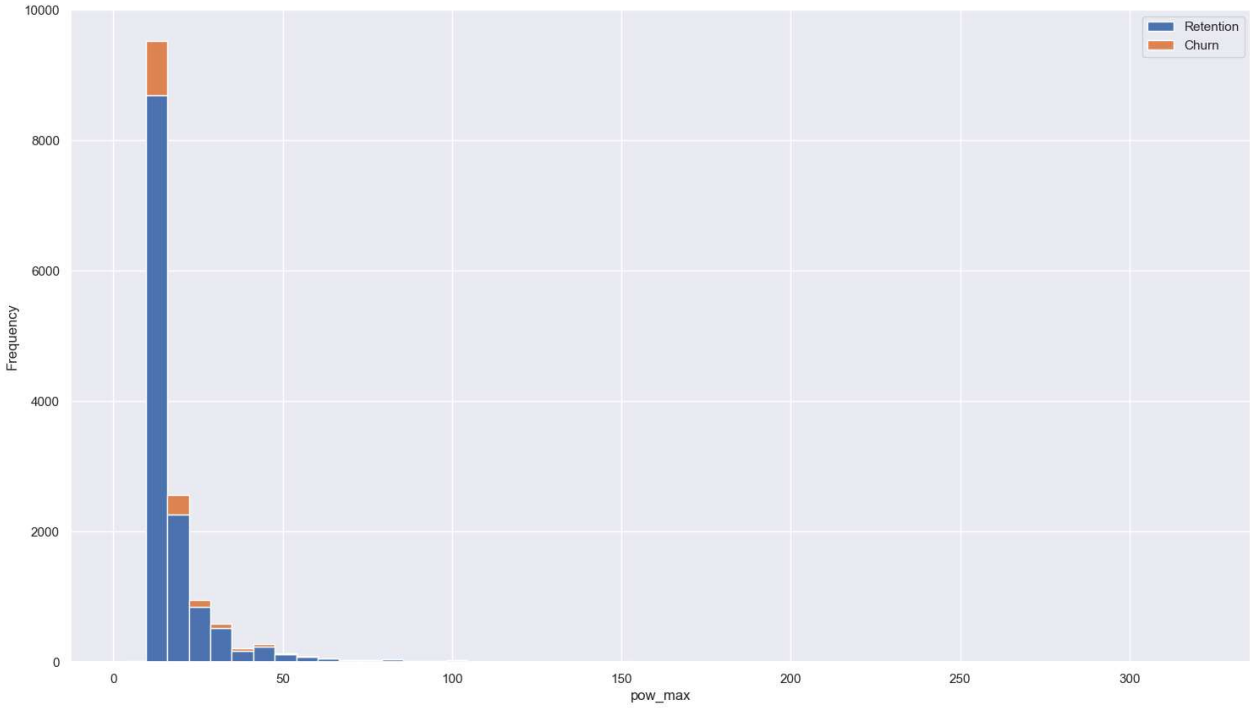

Boxplot grouped by churn
margin_net_pow_ele
350
300
250
200
150
100
50
0
0
1
churn



Subscribed power

In [36]:
```
power = client_df[["id", "pow_max", "churn"]]
```

In [37]:
```
fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)
```

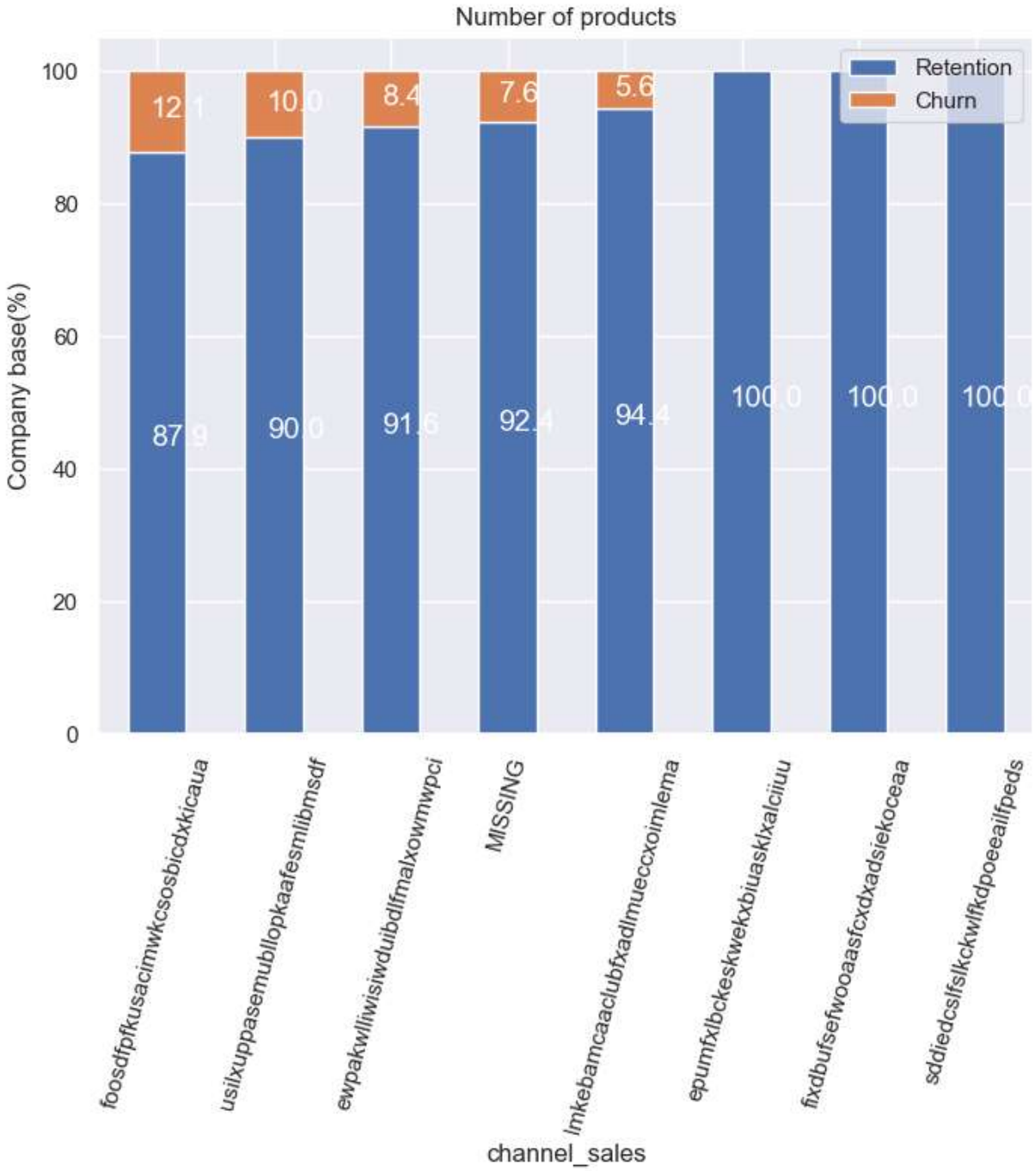


Other Columns

In [38]:
```
other_cols = client_df[['id', 'nb_prod_act', 'num_years_antig', 'origin_up', 'churn']]
products = other_cols.groupby([other_cols["nb_prod_act"], other_cols["churn"]])["id"].count().
products_percentage = (products.div(products.sum(axis=1), axis=0)*100).sort_values(by=[1], asc
```

In [39]:
```
ax=channel_churn.plot(kind='bar',stacked=True,figsize=(8,6),rot=75)
annotate_stacked_bars(ax, textsize=14)
plt.title('Number of products')
plt.legend(['Retention','Churn'],loc='upper right')
plt.ylabel('Company base(%)')
```

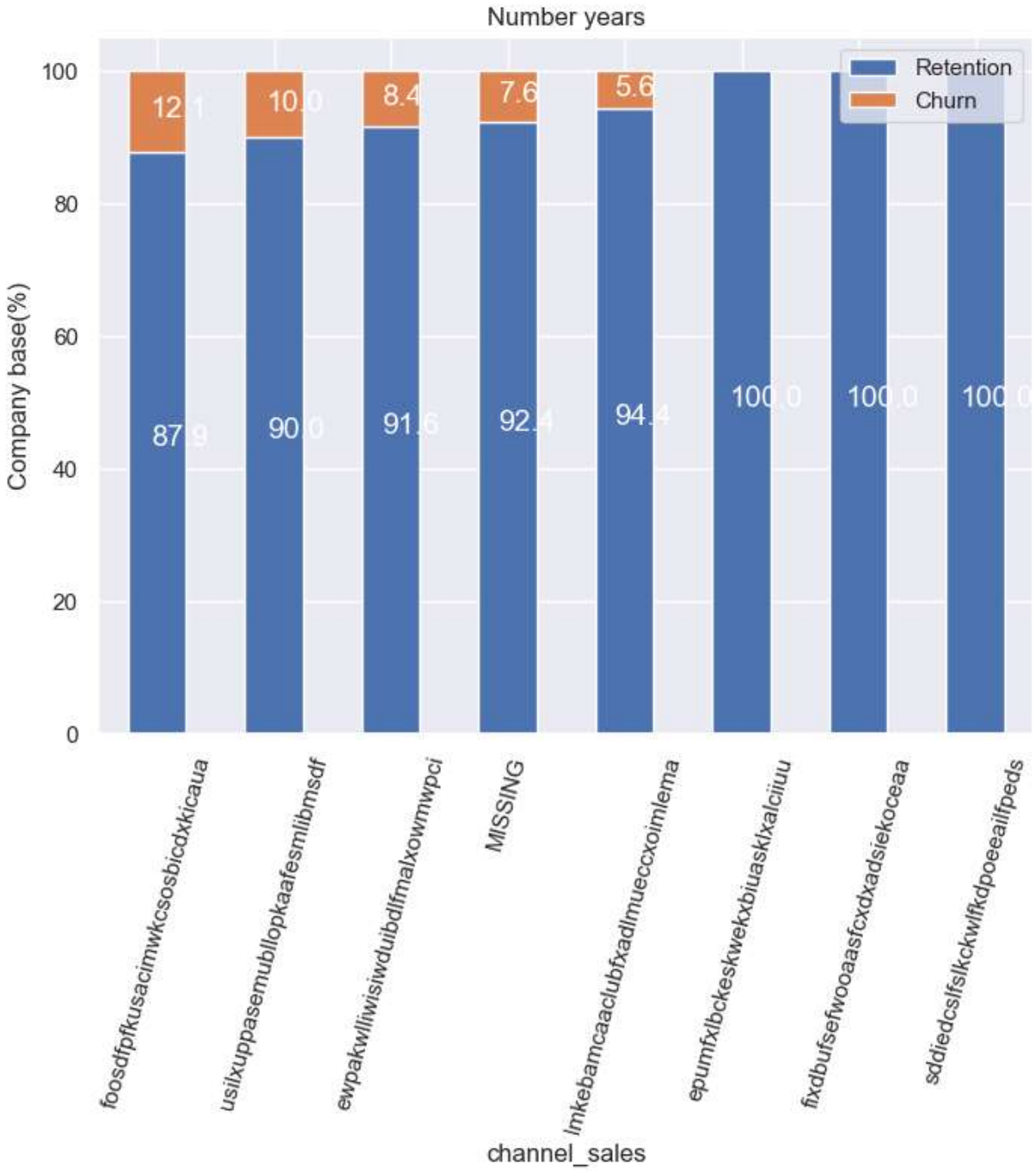
Out[39]: Text(0, 0.5, 'Company base(%)')



In [40]:
```
years_antig = other_cols.groupby([other_cols["num_years_antig"], other_cols["churn"]])["id"].c
years_antig_percentage = (years_antig.div(years_antig.sum(axis=1), axis=0)*100)
```

In [41]:
```
ax=channel_churn.plot(kind='bar',stacked=True,figsize=(8,6),rot=75)
annotate_stacked_bars(ax, textsize=14)
plt.title('Number years')
plt.legend(['Retention','Churn'],loc='upper right')
plt.ylabel('Company base(%)')
```

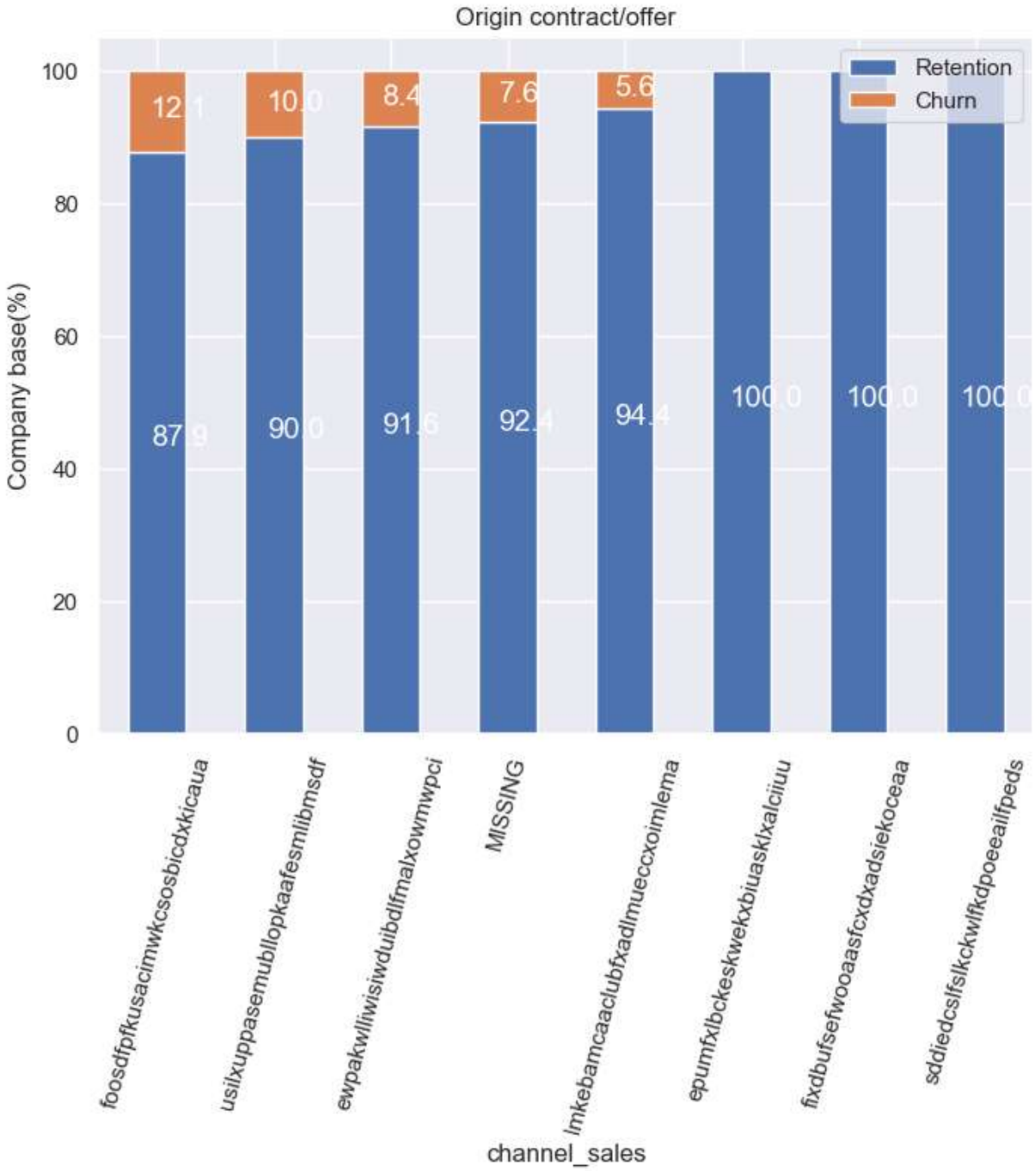
Out[41]:
```
Text(0, 0.5, 'Company base(%)')
```



In [42]:
```
origin = other_cols.groupby([other_cols["origin_up"], other_cols["churn"]])["id"].count().unst
origin_percentage = (origin.div(origin.sum(axis=1), axis=0)*100)
```

In [43]:
```
ax=channel_churn.plot(kind='bar',stacked=True,figsize=(8,6),rot=75)
annotate_stacked_bars(ax, textsize=14)
plt.title('Origin contract/offer')
plt.legend(['Retention','Churn'],loc='upper right')
plt.ylabel('Company base(%)')
```

Out[43]:
```
Text(0, 0.5, 'Company base(%)')
```



Let's check Price Dataset

In [44]:
```
price_df['id'].value_counts().value_counts()
```

Out[44]:
```
12    15990
11       83
10       11
9         6
8         3
7         3
Name: id, dtype: int64
```

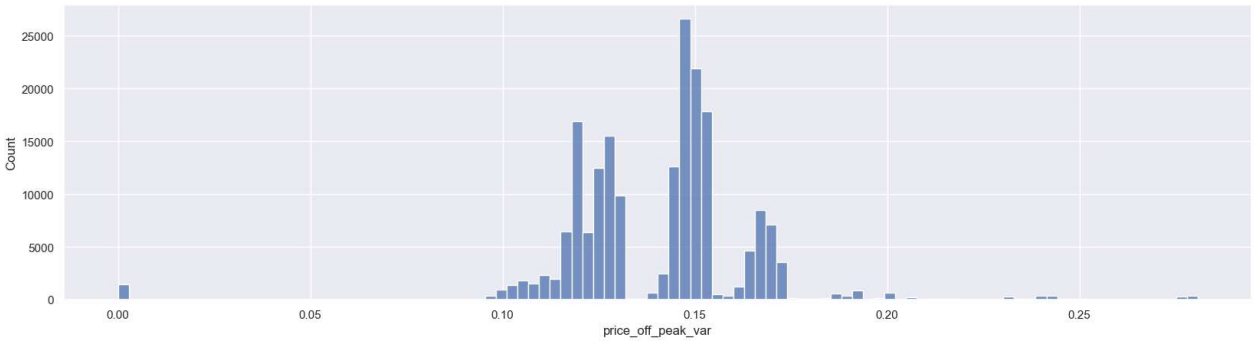
Most customers have a 12-month price records each. Luckily, every customer has the last-month's reocrd (2015-12-01), this will be helpful in identifying price sensitivity.

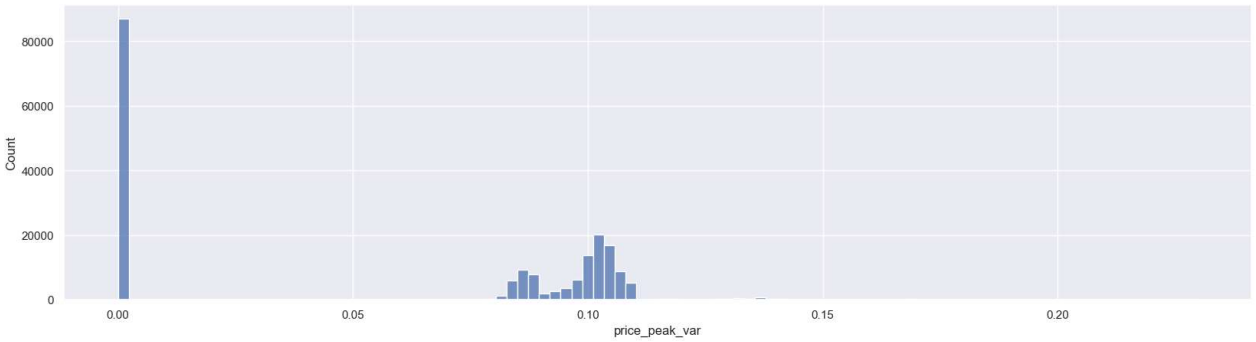
In [45]:
```
price_df['price_date'].value_counts()
```

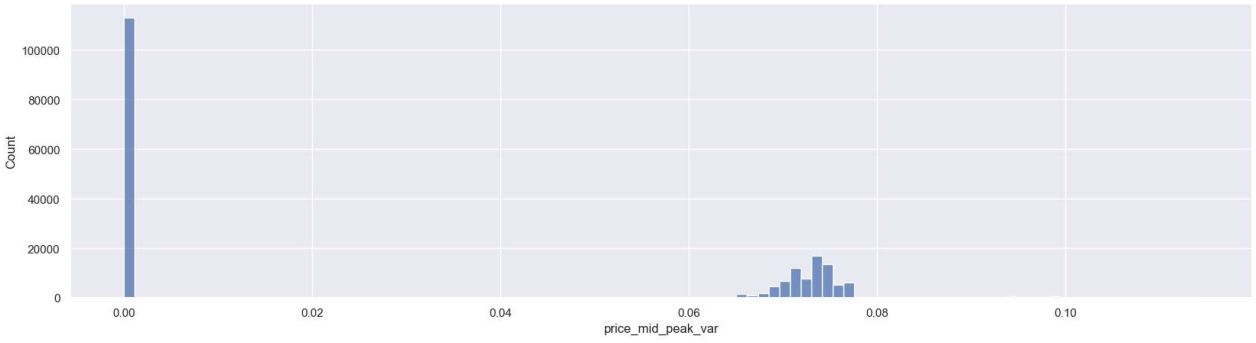
Out[45]:
```
2015-08-01    16094
2015-12-01    16094
2015-07-01    16090
2015-11-01    16087
2015-06-01    16085
2015-10-01    16085
2015-02-01    16082
2015-09-01    16082
2015-05-01    16080
2015-04-01    16079
2015-03-01    16074
2015-01-01    16070
Name: price_date, dtype: int64
```

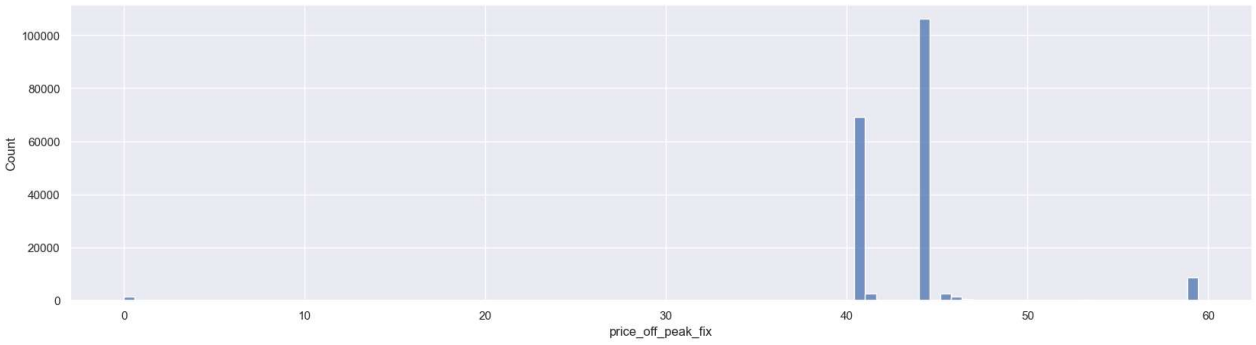
In [46]:
```
# plot histogram
def plot_histogram_by_churn(df, target_variable, figsize=(20,5), bins=100, if_churn=True):
    fig = plt.figure(figsize=figsize)
    if if_churn:
        ax = sns.histplot(data=df, x=target_variable, bins=bins, hue='churn')
    else:
        ax = sns.histplot(data=df, x=target_variable, bins=bins)
```

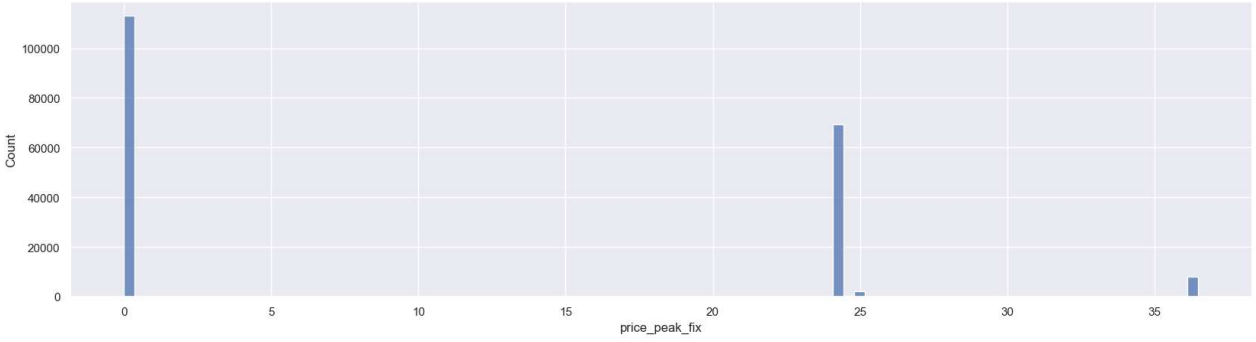
In [47]:
```
for attr in ['price_off_peak_var', 'price_peak_var', 'price_mid_peak_var', 'price_off_peak_fix
    plot_histogram_by_churn(df=price_df, target_variable=attr, if_churn=False)
```

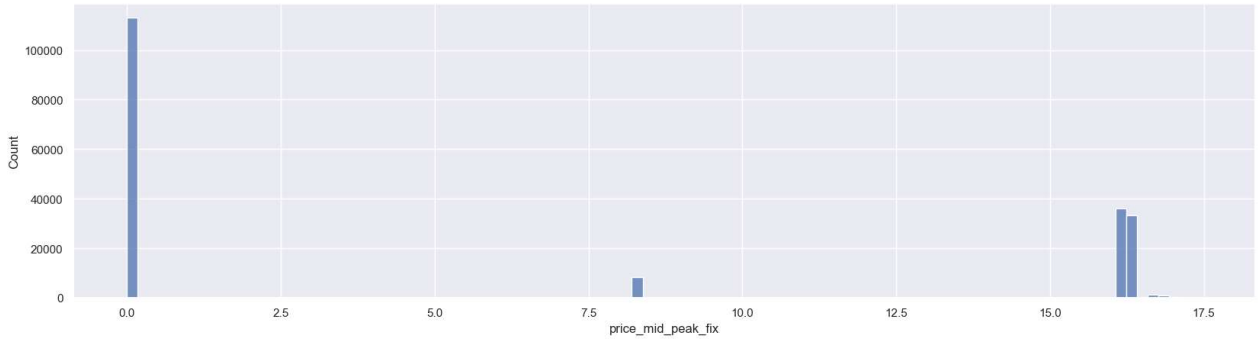












Plot histograms of price features. The values of each feature are centred around some points.

There are lots of 0 peak prices and mid-peak prices, but very few 0 off peak prices. So, I will analyse price sensivity only based on off-peak prices

Hypothesis Investigation

Now, let's check customers sensitivity to price. When the price increases and some cutomers leave, then we can say these customers are sensitive to price as the increase in prices lead the customer churn. In case there is a decrease or no change in the price and cutomers still switch, it is hard to say these cutomers are sensitive to price.

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 [48]:
```
# 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()
```

In [49]:
```
mean_year.head()
```

Out[49]:

	id	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	pr
0	0002203ffbb812588b632b9e628cc38d	0.124338	0.103794	0.073160	40.701732	
1	0004351ebdd665e6ee664792efc4fd13	0.146426	0.000000	0.000000	44.385450	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.181558	0.000000	0.000000	45.319710	
3	0010ee3855fdea87602a5b7aba8e42de	0.118757	0.098292	0.069032	40.647427	
4	00114d74e963e47177db89bc70108537	0.147926	0.000000	0.000000	44.266930	

In [50]:
```
#rename the columns of mean year
mean_year = mean_year.rename(

    columns={
        "price_off_peak_var": "mean_year_price_off_peak_var",
        "price_peak_var": "mean_year_price_peak_var",
        "price_mid_peak_var": "mean_year_price_mid_peak_var",
        "price_off_peak_fix": "mean_year_price_off_peak_fix",
        "price_peak_fix": "mean_year_price_peak_fix",
        "price_mid_peak_fix": "mean_year_price_mid_peak_fix"
    }
)
```

In [51]:
```
mean_year["mean_year_price_off_peak"] = mean_year["mean_year_price_off_peak_var"] + mean_year[
mean_year["mean_year_price_peak"] = mean_year["mean_year_price_peak_var"] + mean_year["mean_ye
mean_year["mean_year_price_med_peak"] = mean_year["mean_year_price_mid_peak_var"] + mean_year[
```

In [52]:
```
#rename the columns of mean 6 month
mean_6m = mean_6m.rename(

    columns={
        "price_off_peak_var": "mean_6m_price_off_peak_var",
        "price_peak_var": "mean_6m_price_peak_var",
        "price_mid_peak_var": "mean_6m_price_mid_peak_var",
        "price_off_peak_fix": "mean_6m_price_off_peak_fix",
        "price_peak_fix": "mean_6m_price_peak_fix",
        "price_mid_peak_fix": "mean_6m_price_mid_peak_fix"
    }
)

mean_6m["mean_6m_price_off_peak"] = mean_6m["mean_6m_price_off_peak_var"] + mean_6m["mean_6m_p
mean_6m["mean_6m_price_peak"] = mean_6m["mean_6m_price_peak_var"] + mean_6m["mean_6m_price_peal
mean_6m["mean_6m_price_med_peak"] = mean_6m["mean_6m_price_mid_peak_var"] + mean_6m["mean_6m_p
```

In [53]:
```
#rename the columns of mean 3 month
mean_3m = mean_3m.rename(

    columns={
        "price_off_peak_var": "mean_3m_price_off_peak_var",
        "price_peak_var": "mean_3m_price_peak_var",
        "price_mid_peak_var": "mean_3m_price_mid_peak_var",
        "price_off_peak_fix": "mean_3m_price_off_peak_fix",
        "price_peak_fix": "mean_3m_price_peak_fix",
        "price_mid_peak_fix": "mean_3m_price_mid_peak_fix"
    }
)

mean_3m["mean_3m_price_off_peak"] = mean_3m["mean_3m_price_off_peak_var"] + mean_3m["mean_3m_p
mean_3m["mean_3m_price_peak"] = mean_3m["mean_3m_price_peak_var"] + mean_3m["mean_3m_price_peal
mean_3m["mean_3m_price_med_peak"] = mean_3m["mean_3m_price_mid_peak_var"] + mean_3m["mean_3m_p
```

In [54]:
```
# 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 [55]:
```
price_features.head()
```

Out[55]:

	id	mean_year_price_off_peak_var	mean_year_price_peak_var	mean_year_price_mi
0	0002203ffbb812588b632b9e628cc38d	0.124338	0.103794	
1	0004351ebdd665e6ee664792efc4fd13	0.146426	0.000000	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.181558	0.000000	
3	0010ee3855fdea87602a5b7aba8e42de	0.118757	0.098292	
4	00114d74e963e47177db89bc70108537	0.147926	0.000000	

5 rows × 28 columns

Let's merge the price feature dataset with churn variable to check whether price sensitivity has any correlation with churn.

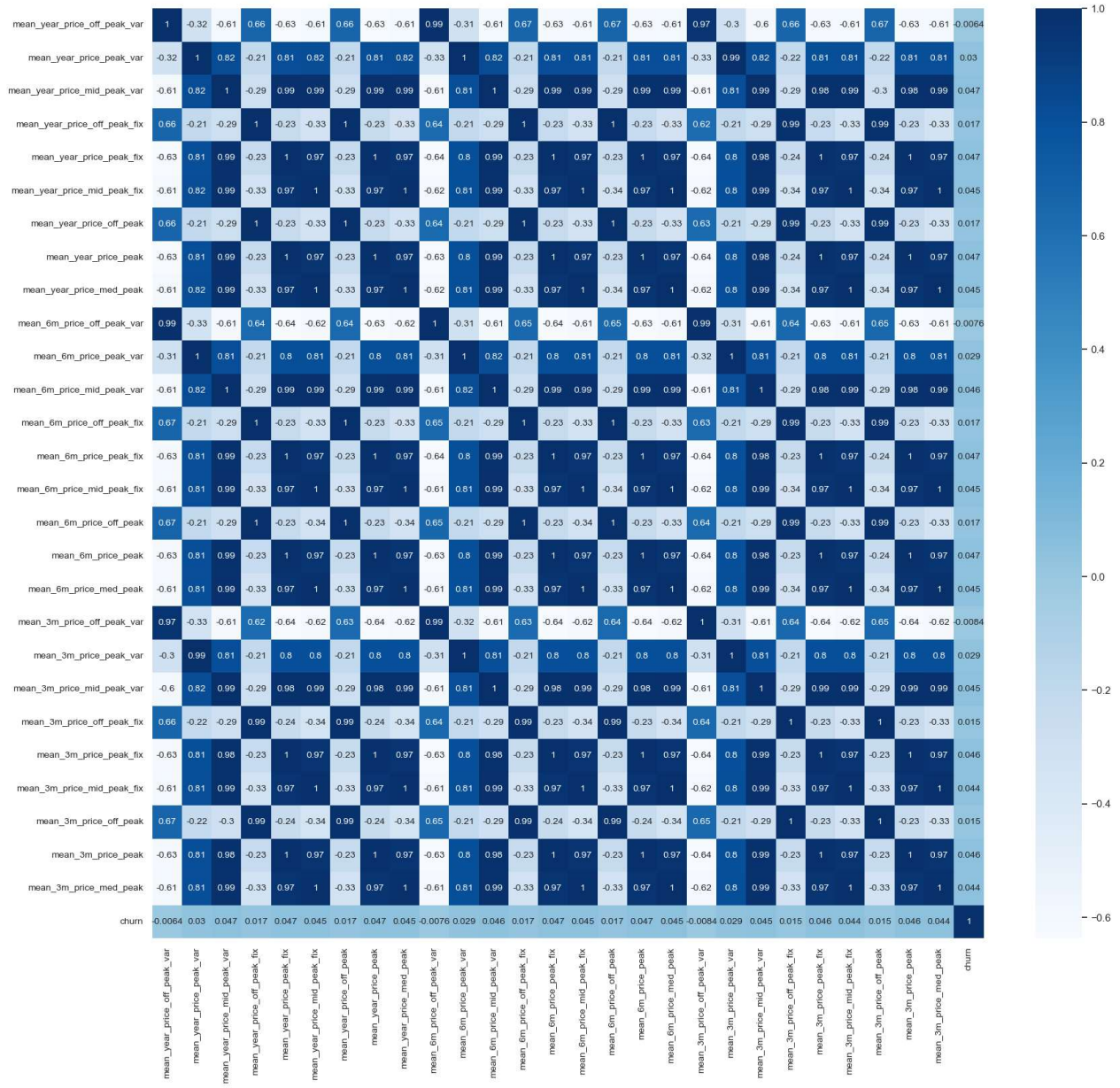
In [56]:
```
price_churn = pd.merge(price_features, client_df[['id', 'churn']], on='id')
price_churn.head()
```

Out[56]:

	id	mean_year_price_off_peak_var	mean_year_price_peak_var	mean_year_price_mi
0	0002203ffbb812588b632b9e628cc38d	0.124338	0.103794	
1	0004351ebdd665e6ee664792efc4fd13	0.146426	0.000000	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.181558	0.000000	
3	00114d74e963e47177db89bc70108537	0.147926	0.000000	
4	0013f326a839a2f6ad87a1859952d227	0.126076	0.105542	

5 rows × 29 columns

In [57]:
```
corr = price_churn.corr()
# Plot correlation
plt.figure(figsize=(20,18))
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, cmap='Blue
# Axis ticks size
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



The above plot shows the correlation betweeen price variables with each other and with churn, however, the correlation between churn and prices variables is very low, which mean customers' churn is not sensitive to price change.

Now, we will merge the client data with price churn data for modeling in the next move.

In [58]:
```
churn_data = pd.merge(client_df.drop(columns=['churn']), price_churn, on='id')
```

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

Out[59]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526

5 rows × 53 columns

In [60]:
```
churn_data.to_csv('clean_data_modeling.csv')
```

- - - - - - - - X X X X X X X X - - - - - - - -