Exploratory Data Analysis Starter

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)
```

Loading data with Pandas

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

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

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

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

Out[3]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_prod	date_renewal	forecast_cons_12m	...
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2016-06-15	2015-11-01	2015-06-23	0.00	...
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2016-08-30	2009-08-21	2015-08-31	189.95	...
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16	2016-04-16	2010-04-16	2015-04-17	47.96	...

3 rows × 26 columns

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

Out[4]:

	id	price_date	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_mid_peak_fix
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.0	44.266931	0.0	0.0
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.0	44.266931	0.0	0.0
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.0	44.266931	0.0	0.0

Descriptive statistics of data

Data types

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

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

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_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
```

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_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

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

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_meter_rent_12m	forecast_price_energy_off_peak	forecast_pric
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.762906	0.966726	63.086871	0.137283	
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.786255	5.108289	66.165783	0.024623	
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.000000	0.000000	16.180000	0.116340	
50%	1.411550e+04	0.000000e+00	792.500000	1112.875000	314.000000	0.000000	18.795000	0.143166	
75%	4.076375e+04	0.000000e+00	3383.000000	2401.790000	1745.750000	0.000000	131.030000	0.146348	
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	175375.000000	30.000000	599.310000	0.273963	

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

Out[8]:

	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

Data visualization

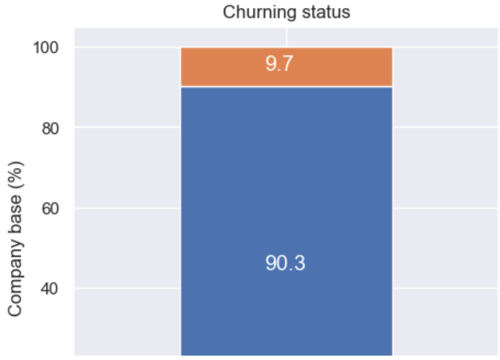
If you're working in Python, two of the most popular packages for visualization are `matplotlib` and `seaborn` . We highly recommend you use these, or at least be familiar with them because they are ubiquitous!

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

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
        )

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

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

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
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5), legend_="lower right")
```

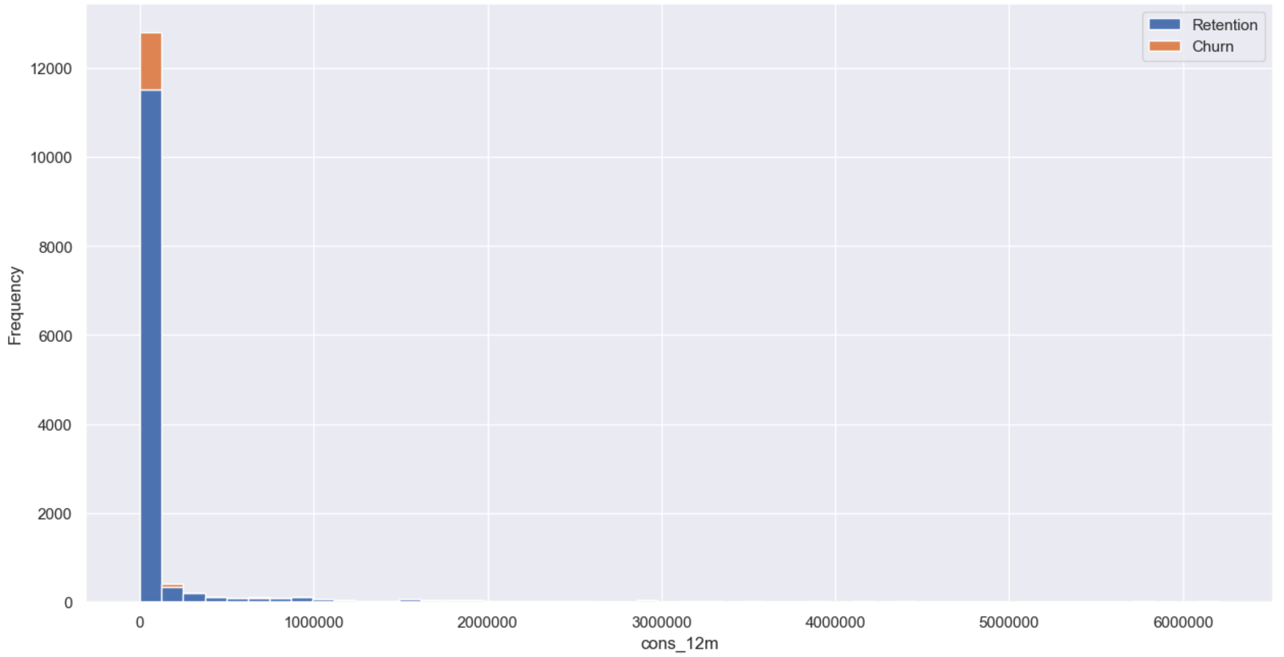


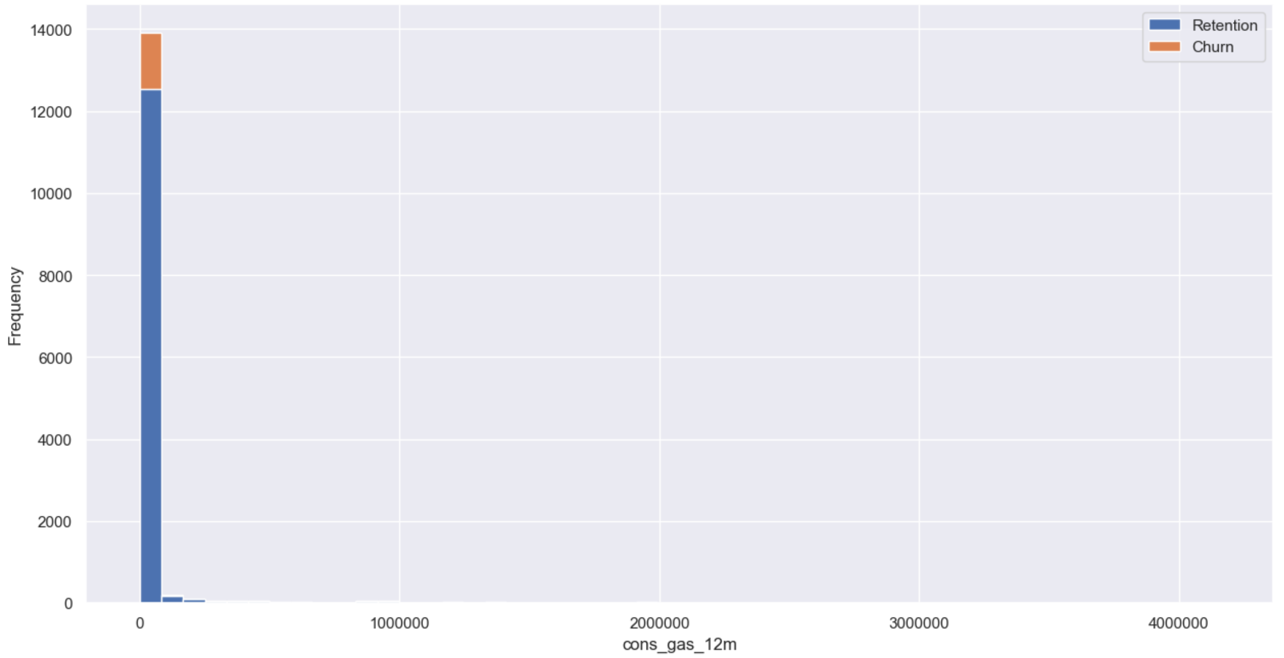
9.6% of customers have churned

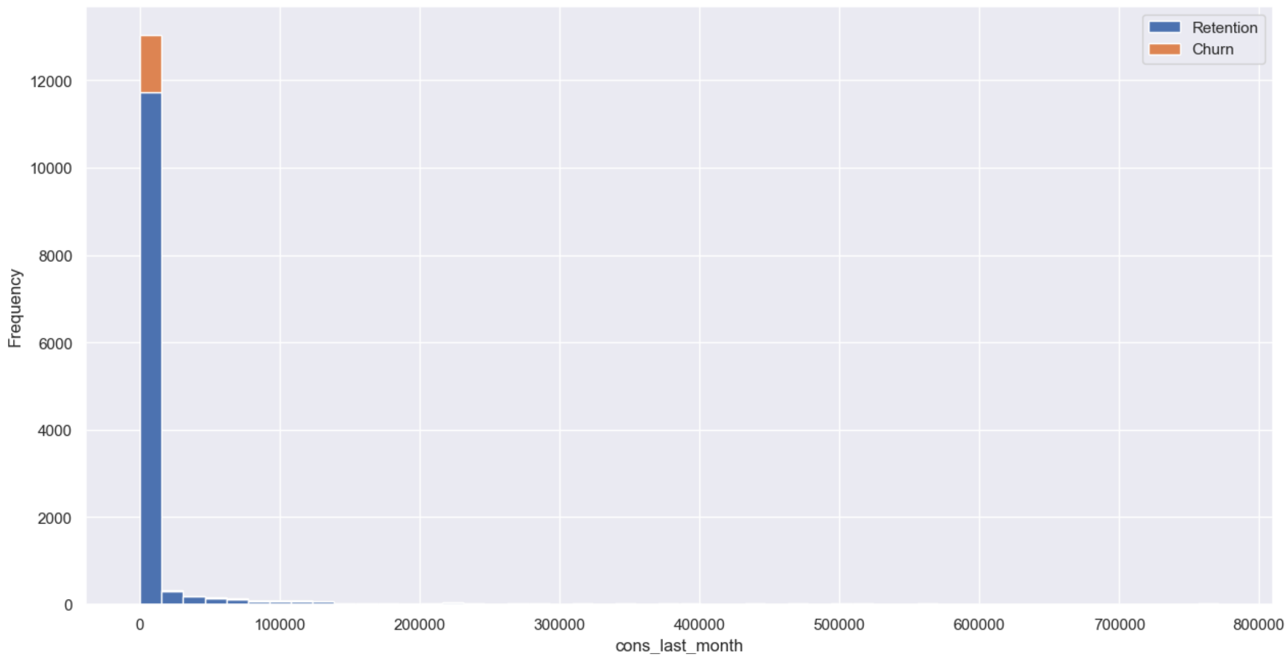
The second function `annotate_bars` is used by the first function, but the third function `plot_distribution` helps you to plot the distribution of a numeric column. An example of how it can be used is given below:

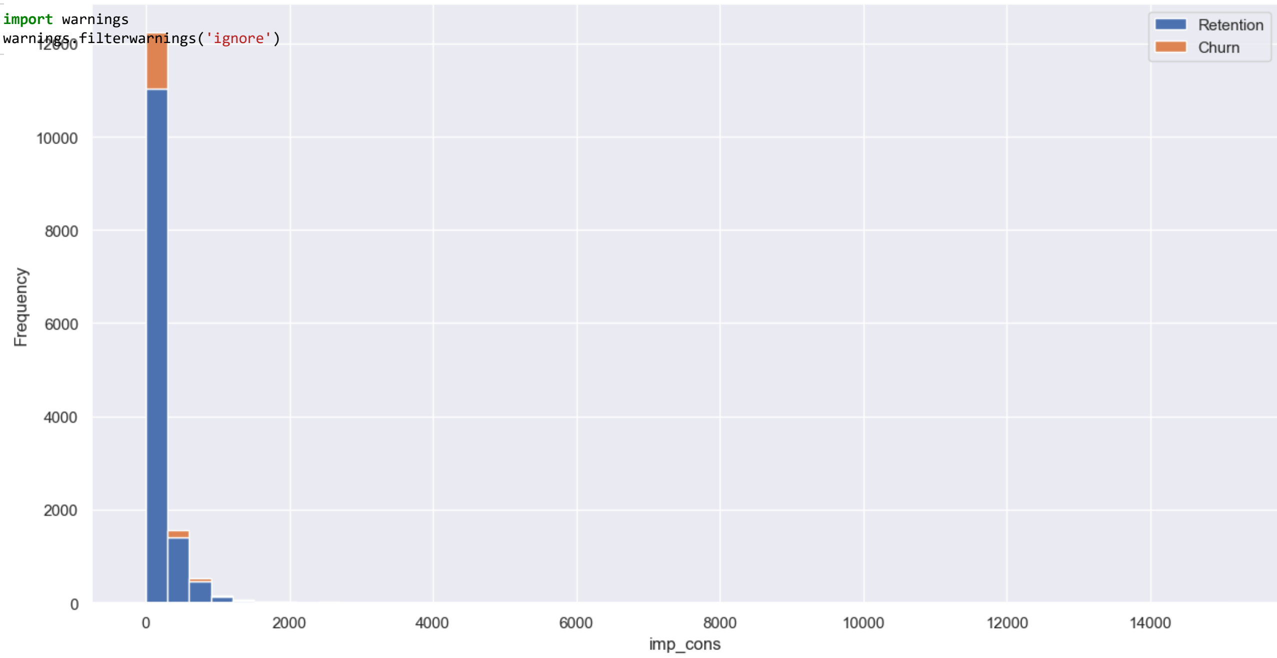
Consumption

In [19]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_cons', 'has_gas', 'churn']]

fig, axs = plt.subplots(nrows=4, figsize=(15, 35))

plot_distribution(consumption, 'cons_12m', axs[0])
plot_distribution(consumption, 'cons_gas_12m', axs[1])
plot_distribution(consumption, 'cons_last_month', axs[2])
plot_distribution(consumption, 'imp_cons', axs[3])

plt.show()
```


Retention
Churn
Frequency
cons_12m


Retention
Churn
Frequency
cons_gas_12m

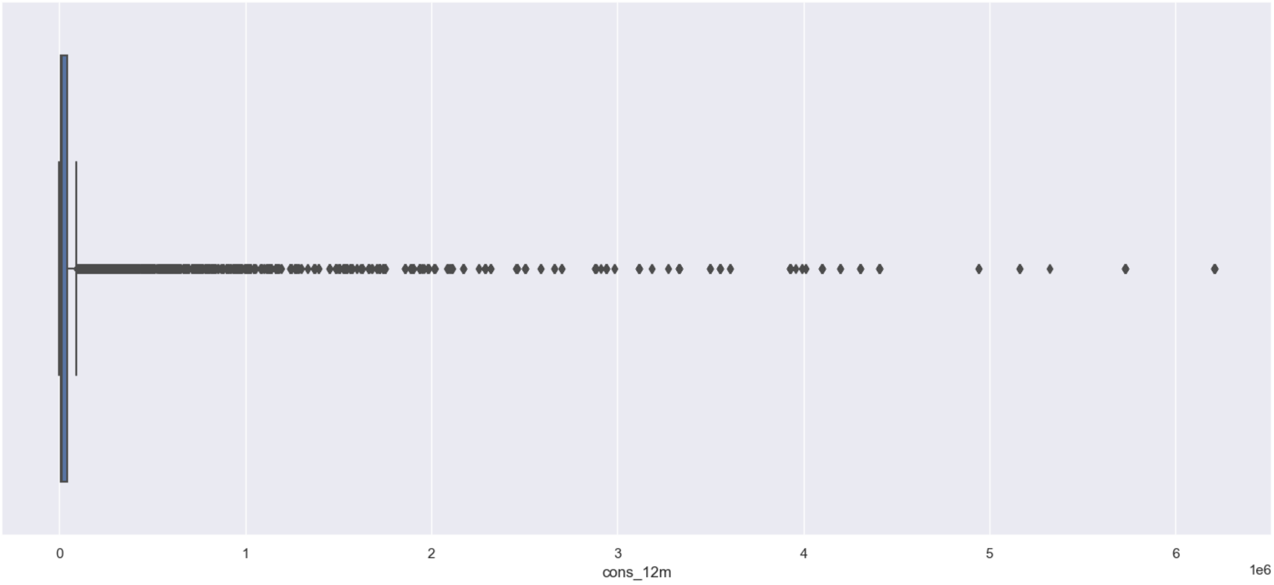

Retention
Churn
Frequency
cons_last_month



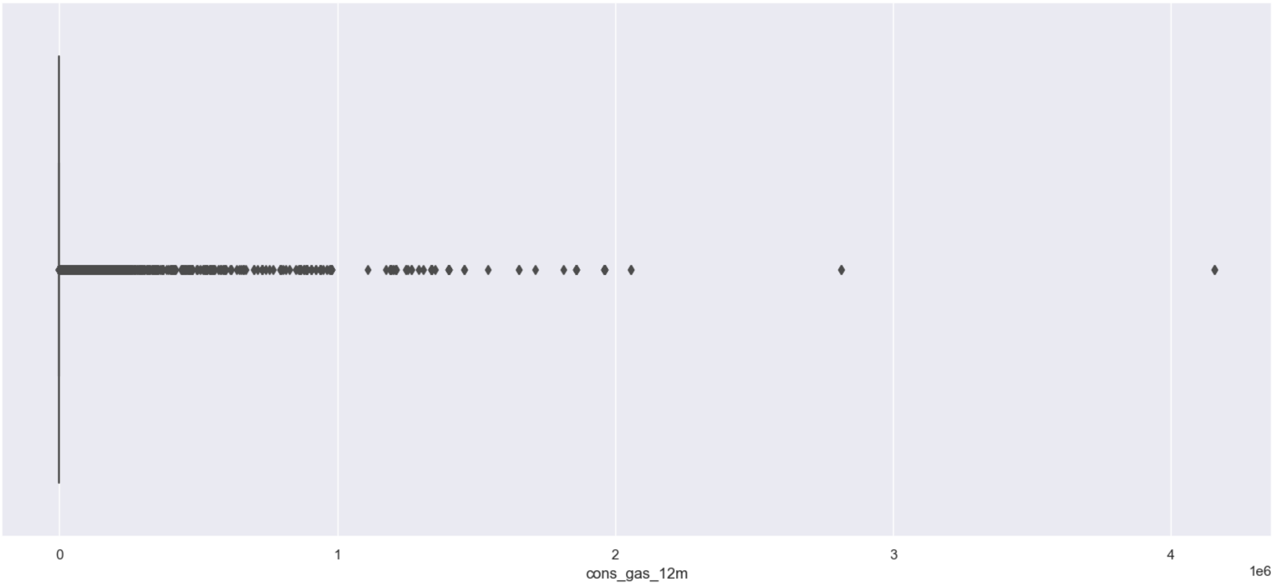
In [21]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [22]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_cons', 'has_gas', 'churn']]

fig, axs = plt.subplots(nrows=4, figsize=(18, 35))

sns.boxplot(consumption.cons_12m,ax= axs[0])
sns.boxplot(consumption.cons_gas_12m, ax=axs[1])
sns.boxplot(consumption.cons_last_month, ax=axs[2])
sns.boxplot(consumption.imp_cons, ax=axs[3])
```

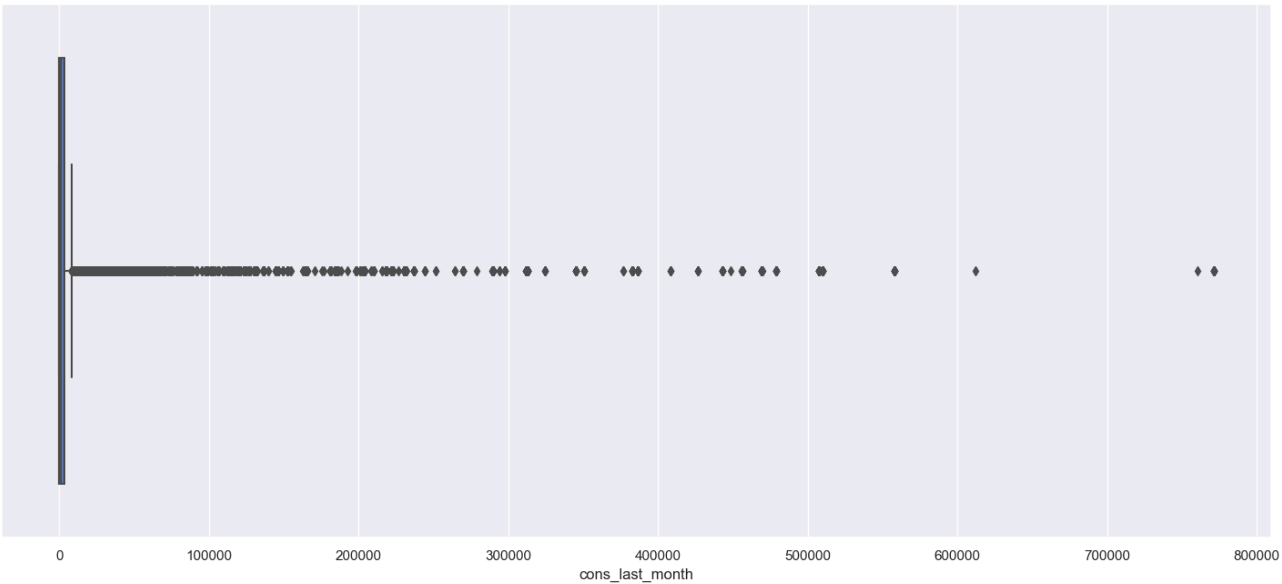
Out[22]:
```
<AxesSubplot:xlabel='imp_cons'>
```



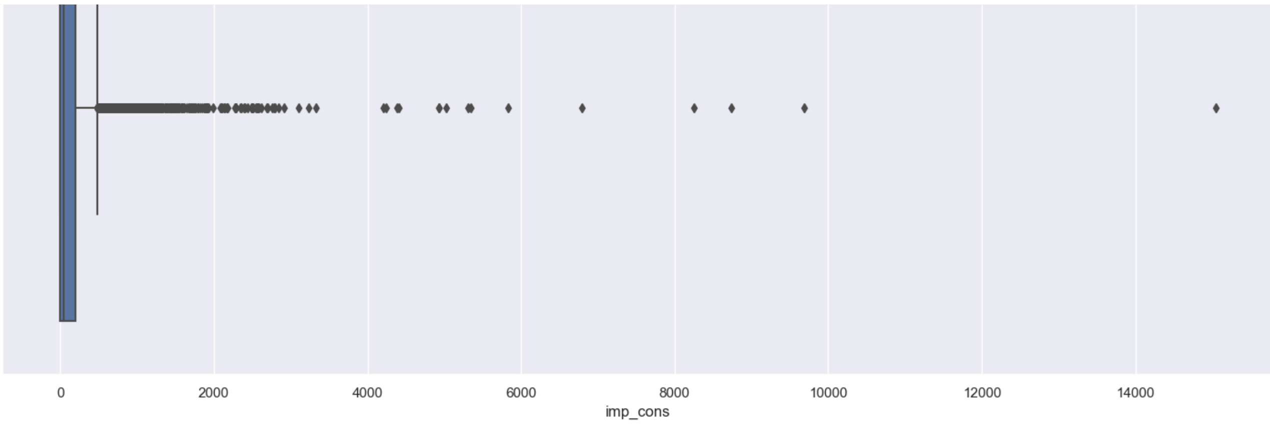
0
1
2
3
4
5
6
1e6
cons_12m



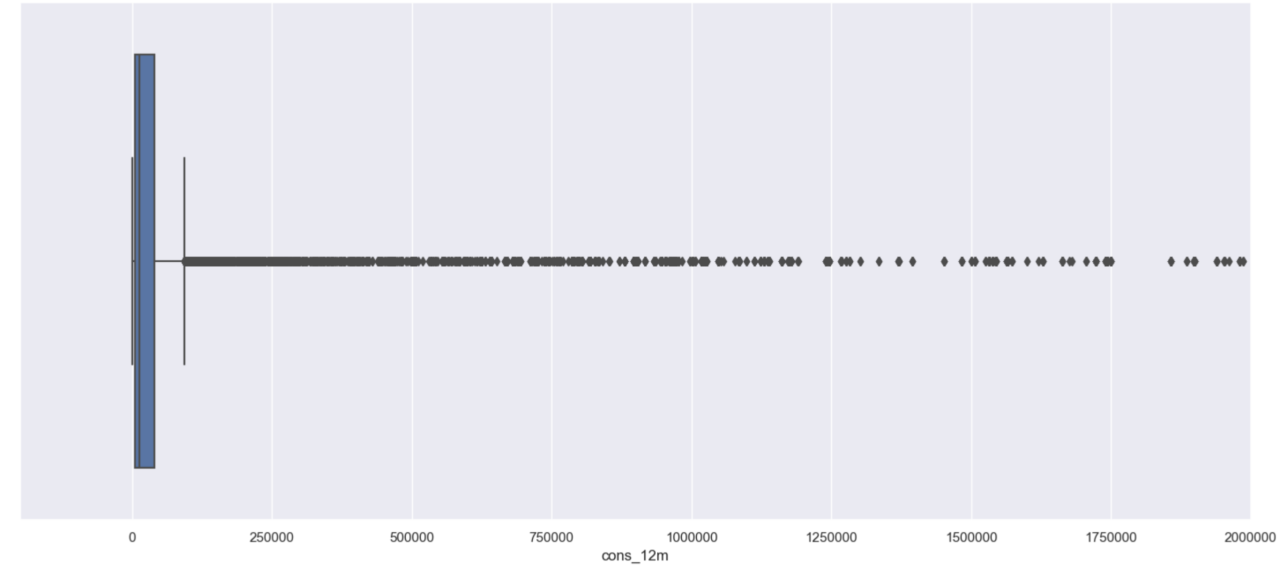
0
1
2
3
4
1e6
cons_gas_12m

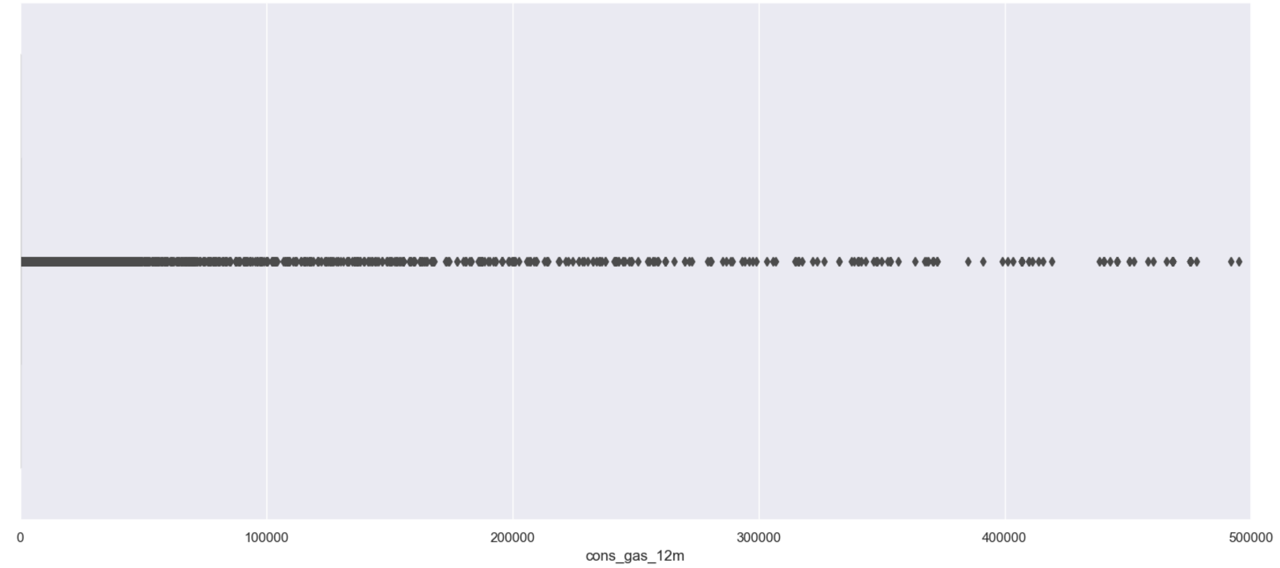


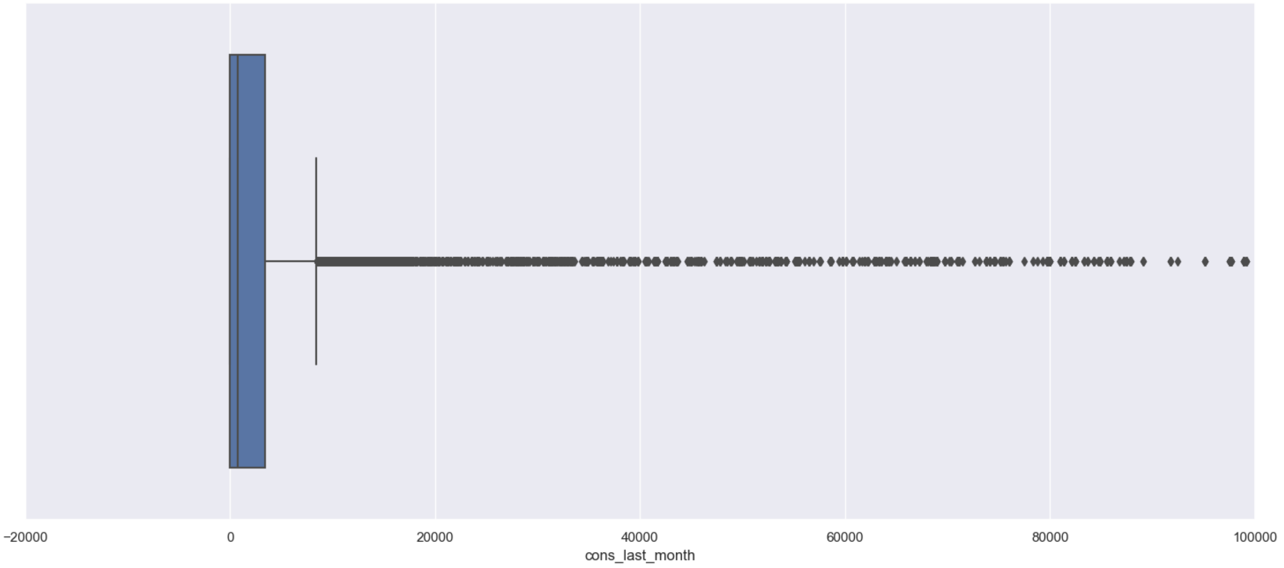
0
100000
200000
300000
400000
500000
600000
700000
800000
cons_last_month

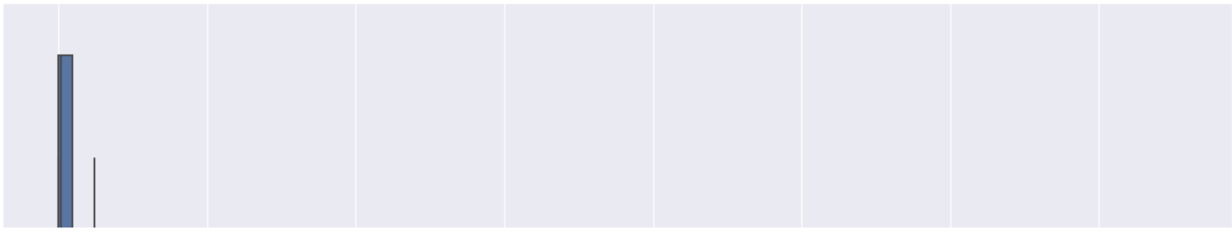

0
2000
4000
6000
8000
10000
12000
14000
imp_cons

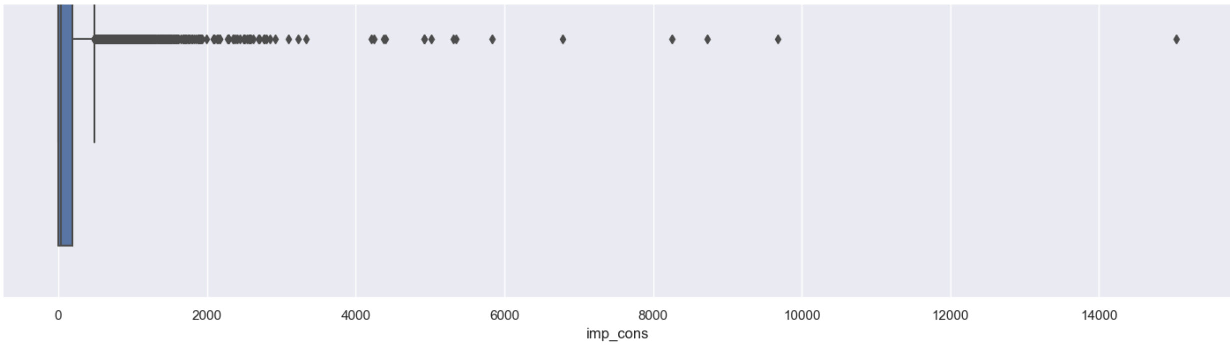
In [23]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_cons', 'has_gas', 'churn']]

fig, axs = plt.subplots(nrows=4, figsize=(18, 35))

sns.boxplot(consumption.cons_12m,ax= axs[0])
sns.boxplot(consumption.cons_gas_12m, ax=axs[1])
sns.boxplot(consumption.cons_last_month, ax=axs[2])
sns.boxplot(consumption.imp_cons, ax=axs[3])

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(0, 500000)
    axs[2].set_xlim(-20000, 100000)
    plt.show()
```


0
250000
500000
750000
1000000
1250000
1500000
1750000
2000000
cons_12m


0
100000
200000
300000
400000
500000
cons_gas_12m

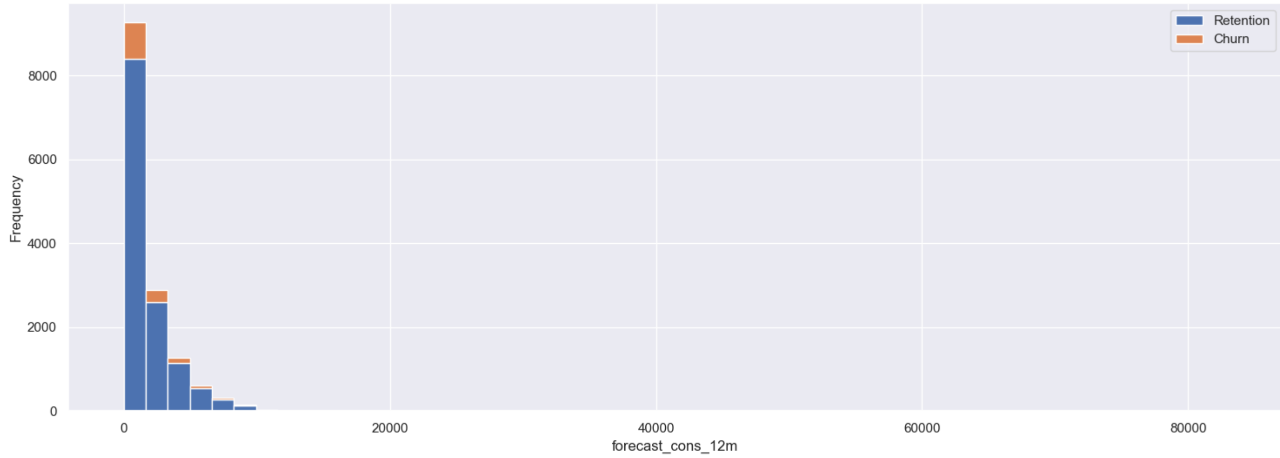

−20000
0
20000
40000
60000
80000
100000
cons_last_month

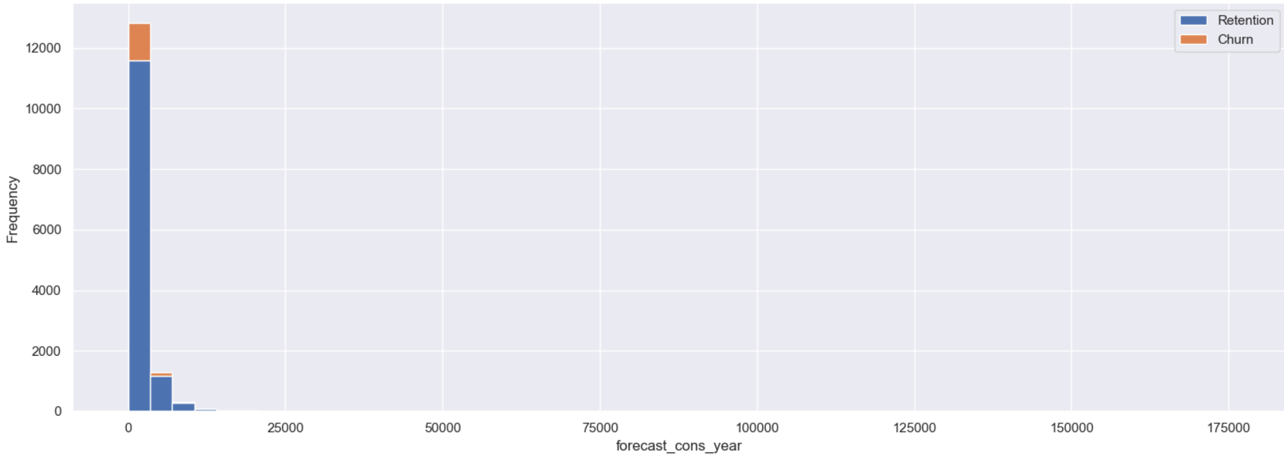


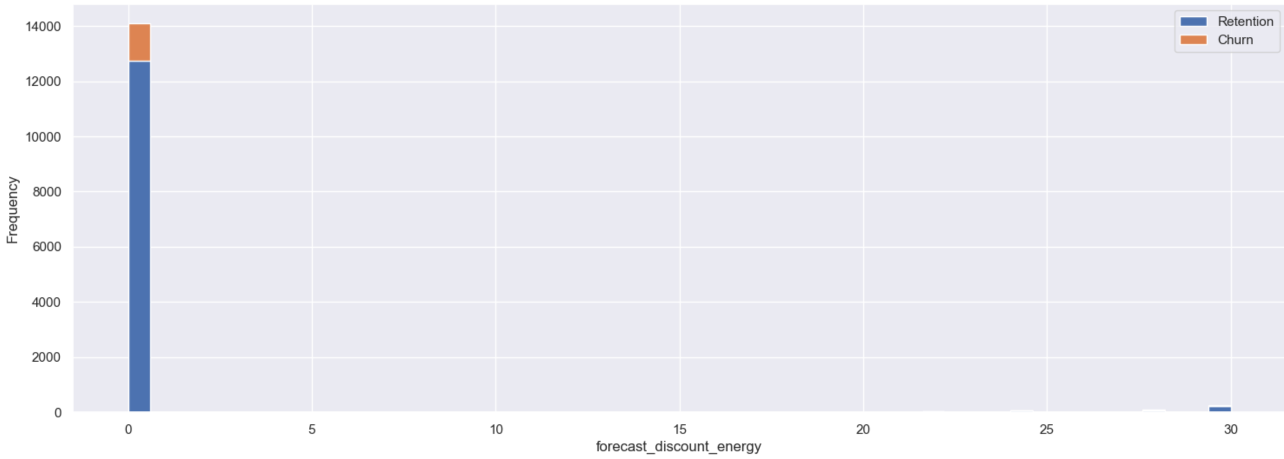

0
2000
4000
6000
8000
10000
12000
14000
imp_cons

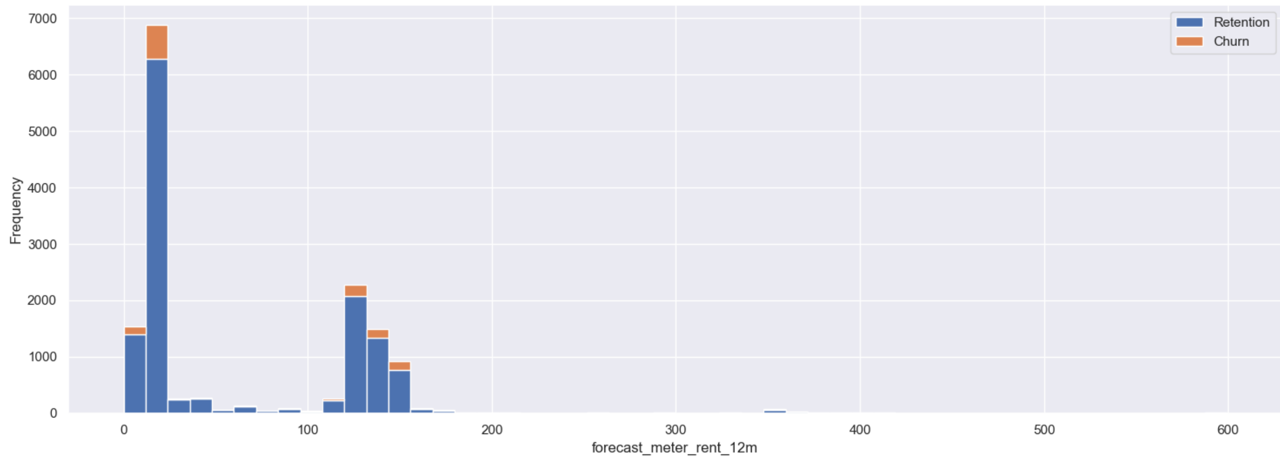
Forecast

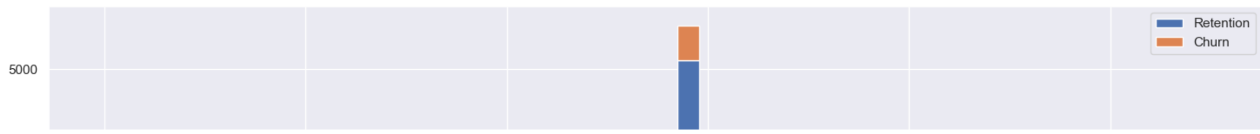
In [24]:
```
forecast = client_df[
    [ "forecast_cons_12m",
    "forecast_cons_year","forecast_discount_energy","forecast_meter_rent_12m",
    "forecast_price_energy_off_peak","forecast_price_energy_peak",
    "forecast_price_pow_off_peak","churn"
    ]
]

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])
```

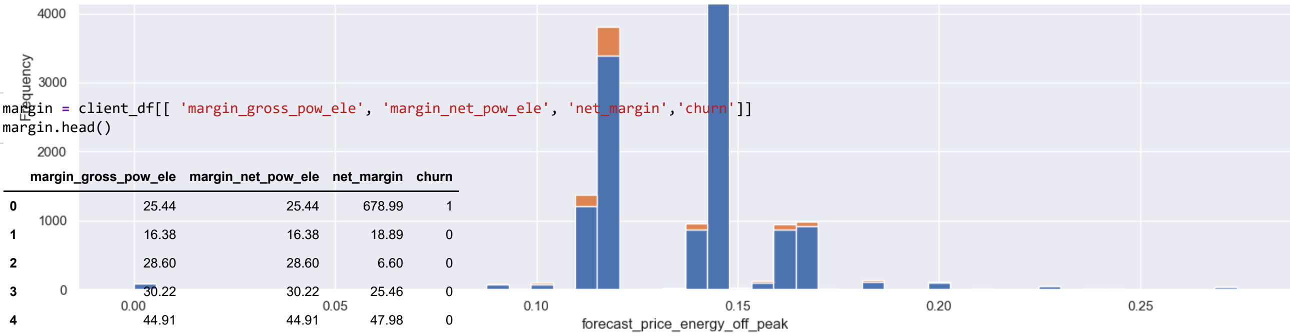

Retention
Churn
Frequency
0
2000
4000
6000
8000
0
20000
40000
60000
80000
forecast_cons_12m


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


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


Retention
Churn
Frequency
0
1000
2000
3000
4000
5000
6000
7000
0
100
200
300
400
500
600
forecast_meter_rent_12m

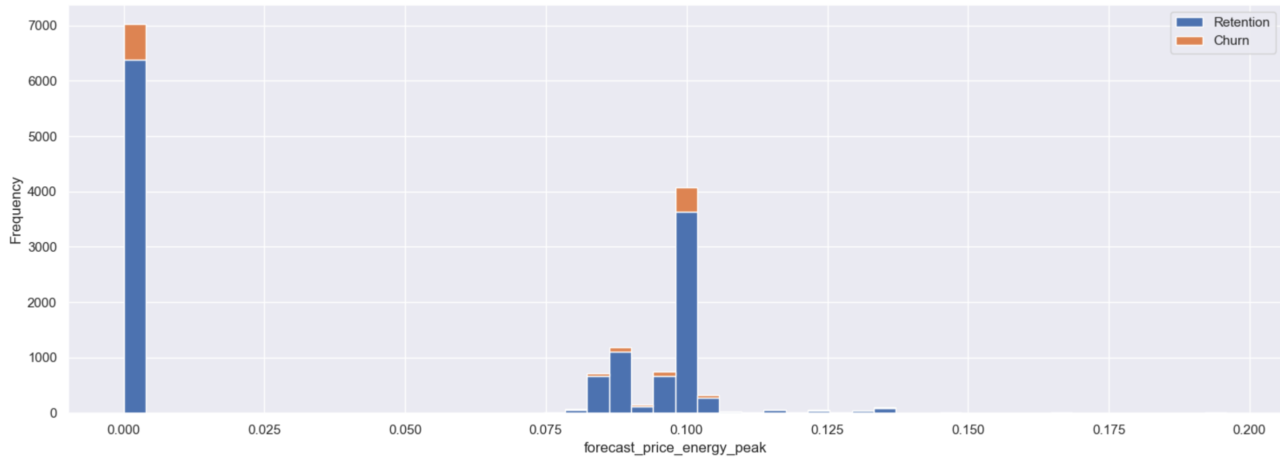

Retention
Churn
5000

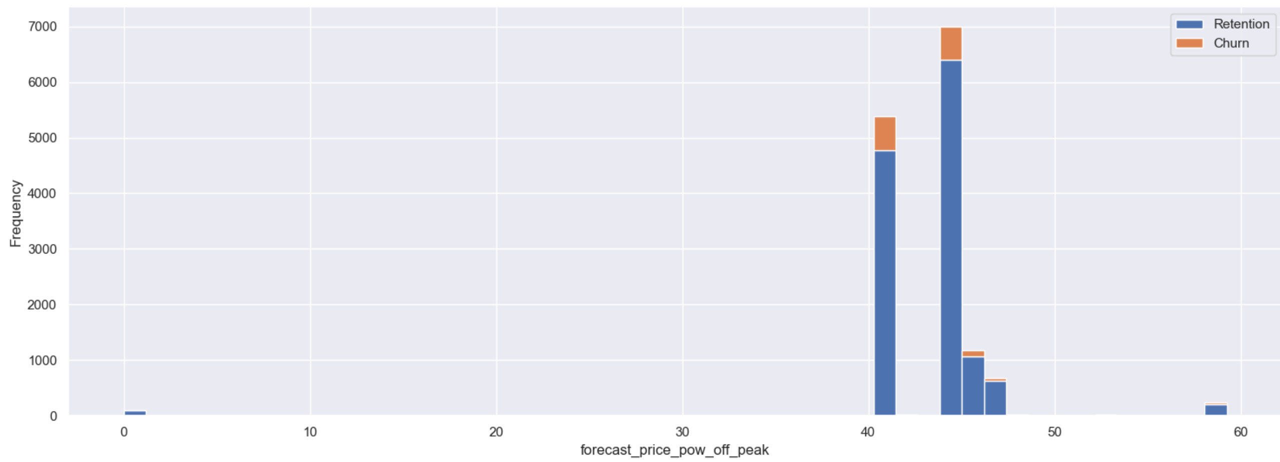


In [25]:
```
margin = client_df[[ 'margin_gross_pow_ele', 'margin_net_pow_ele', 'net_margin','churn']]
margin.head()
```

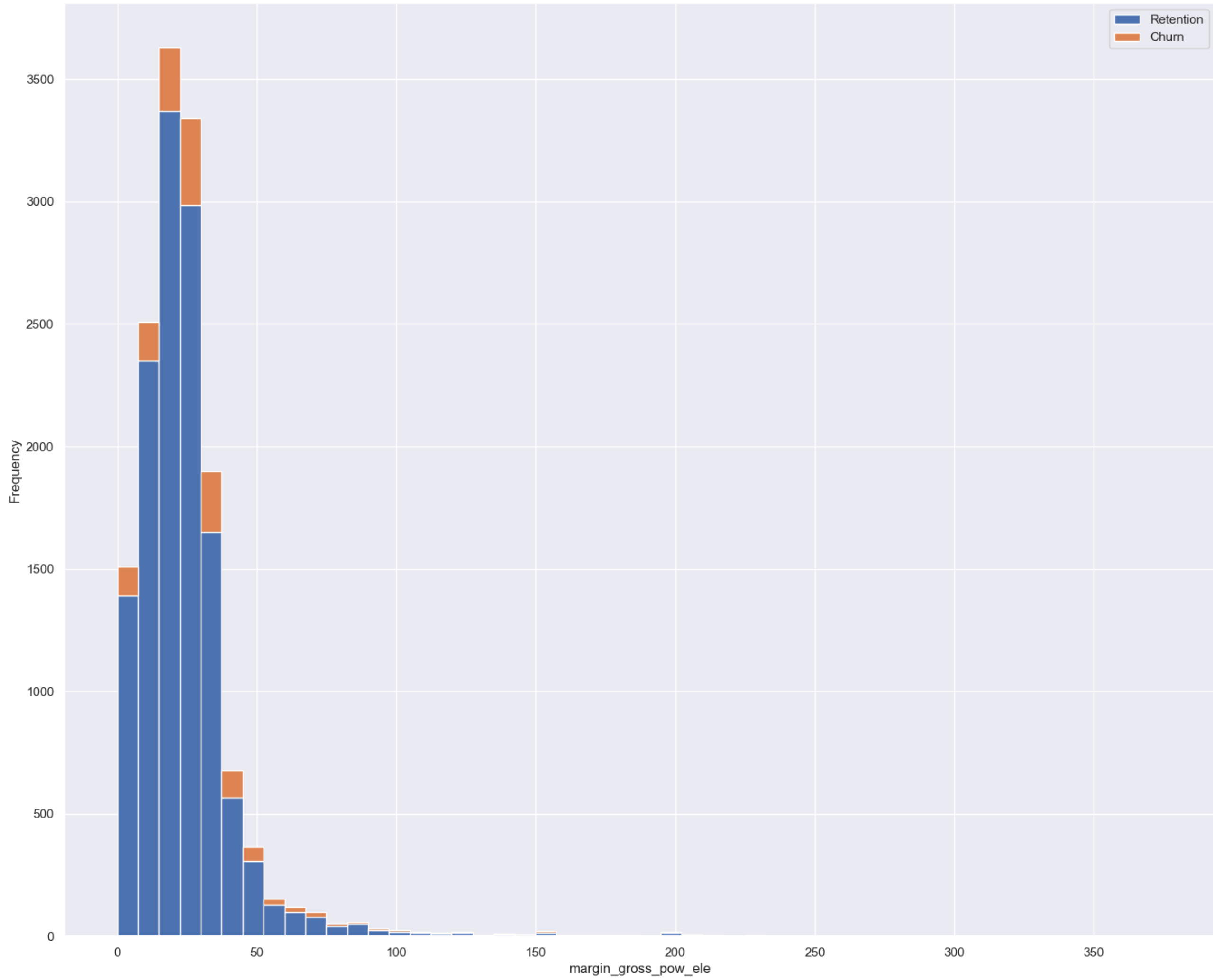
Out[25]:

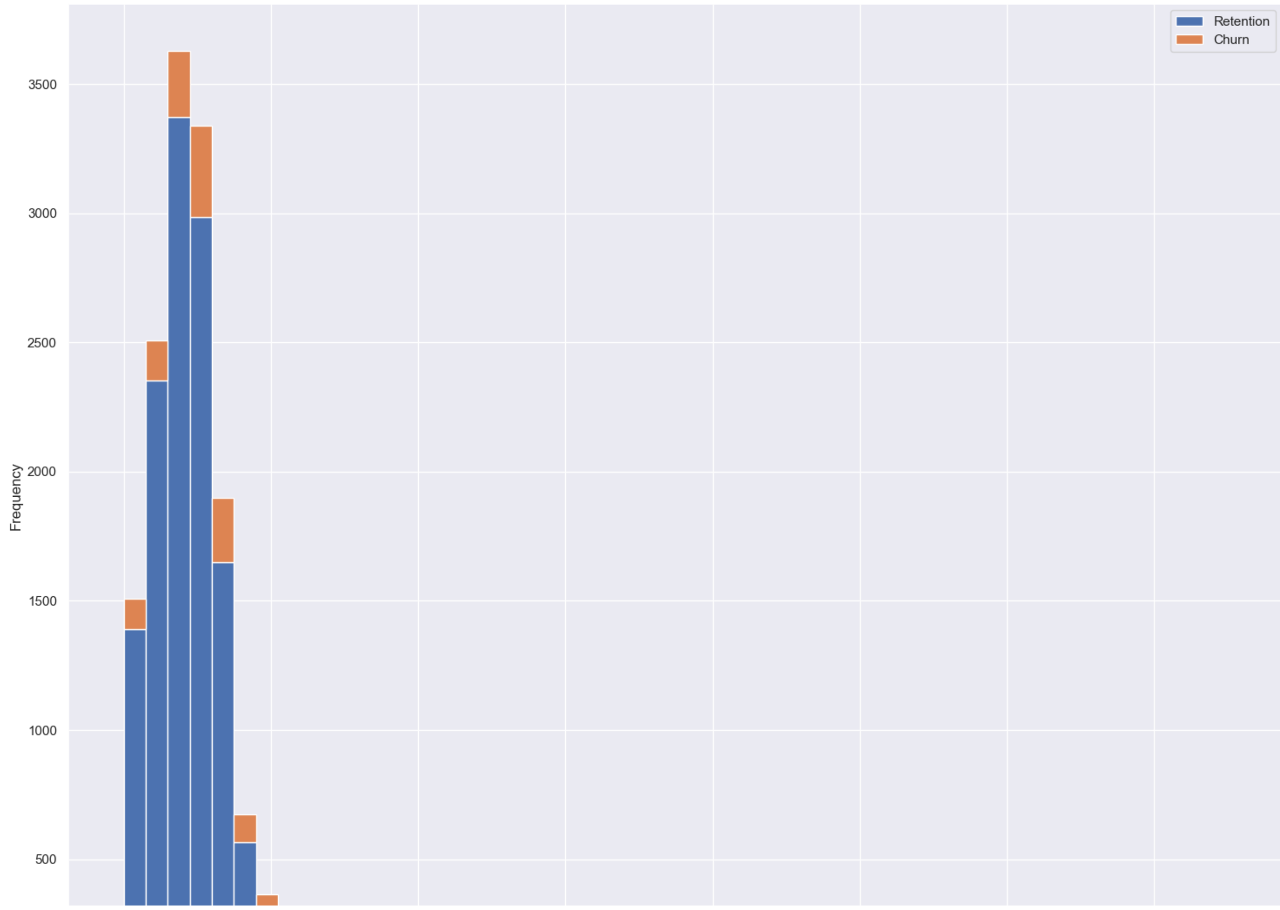
	margin_gross_pow_ele	margin_net_pow_ele	net_margin	churn
0	25.44	25.44	678.99	1
1	16.38	16.38	18.89	0
2	28.60	28.60	6.60	0
3	30.22	30.22	25.46	0
4	44.91	44.91	47.98	0

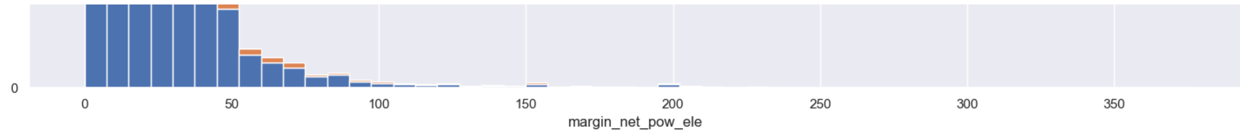


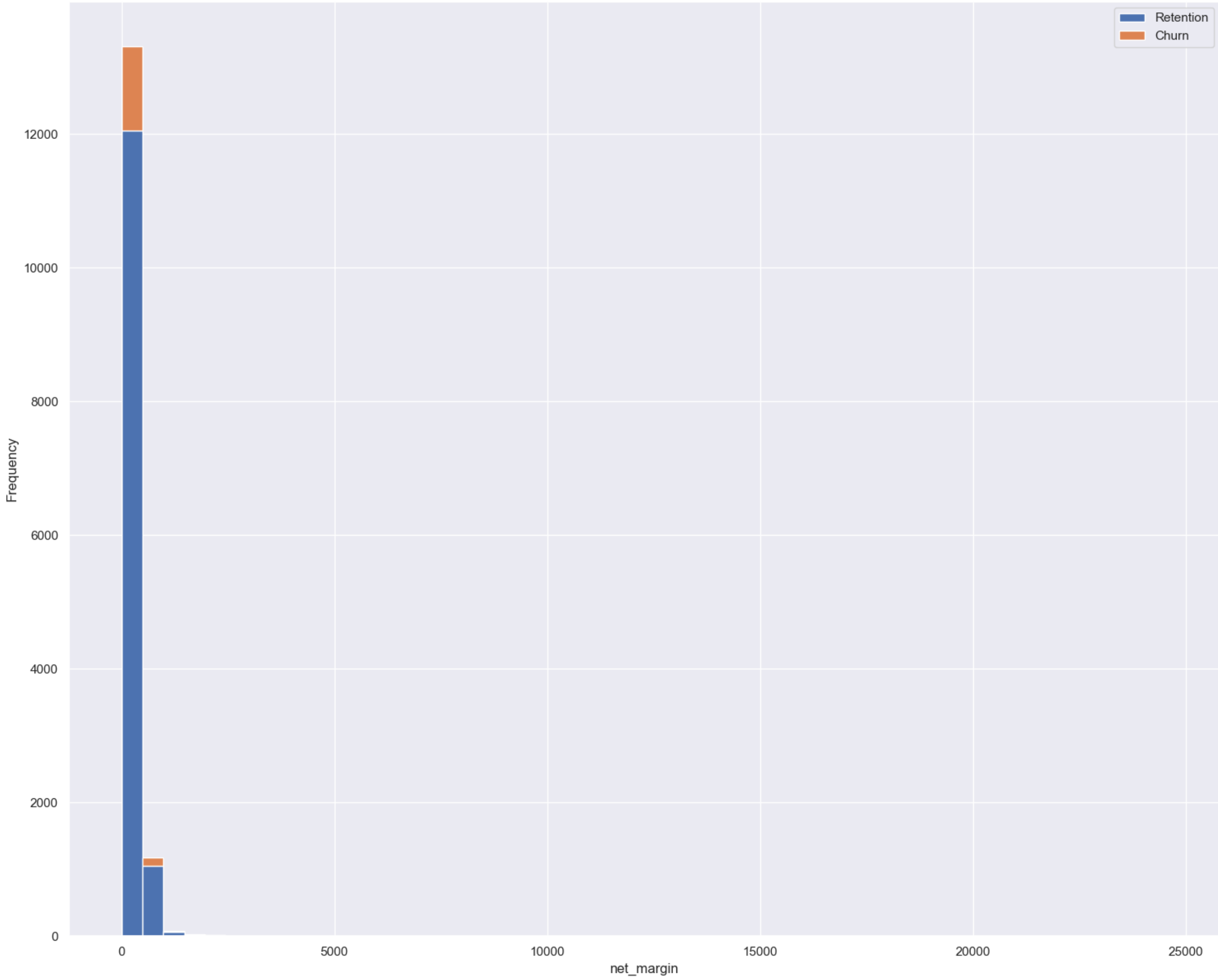


In [26]:
```
fig, axs = plt.subplots(nrows=3, figsize=(18,50))

# Plot histogram
plot_distribution(client_df, "margin_gross_pow_ele", axs[0])
plot_distribution(client_df, "margin_net_pow_ele", axs[1])
plot_distribution(client_df, "net_margin", axs[2])
```


Retention
Churn
Frequency
0
500
1000
1500
2000
2500
3000
3500
0
50
100
150
200
250
300
350
margin_gross_pow_ele

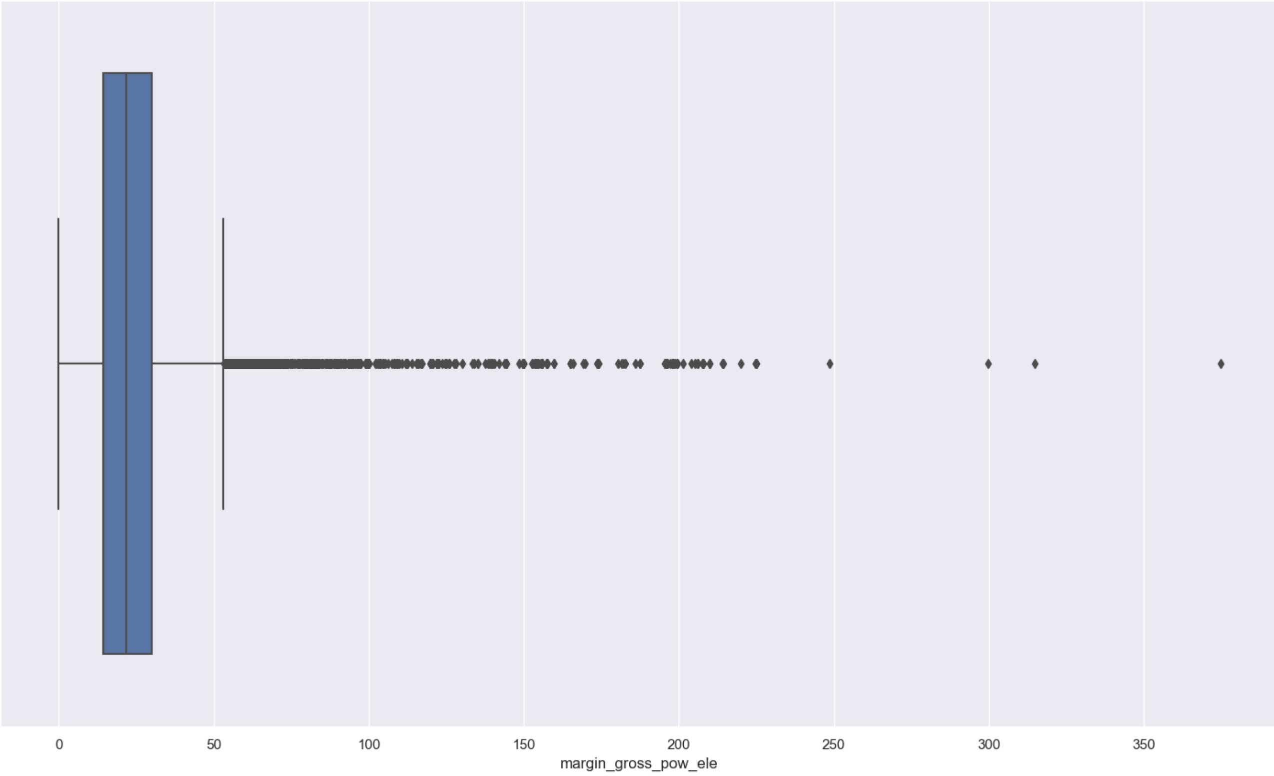

Retention
Churn
Frequency
500
1000
1500
2000
2500
3000
3500

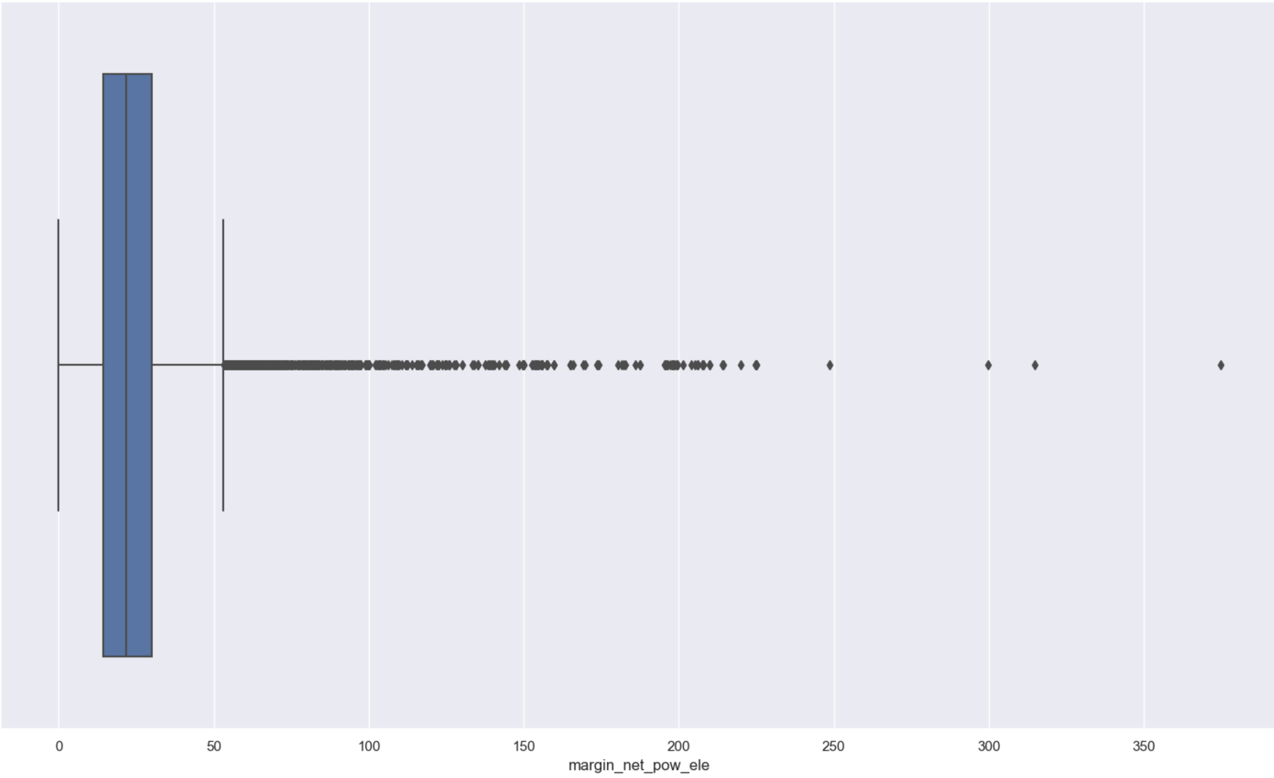

0
0
50
100
150
200
250
300
350
margin_net_pow_ele

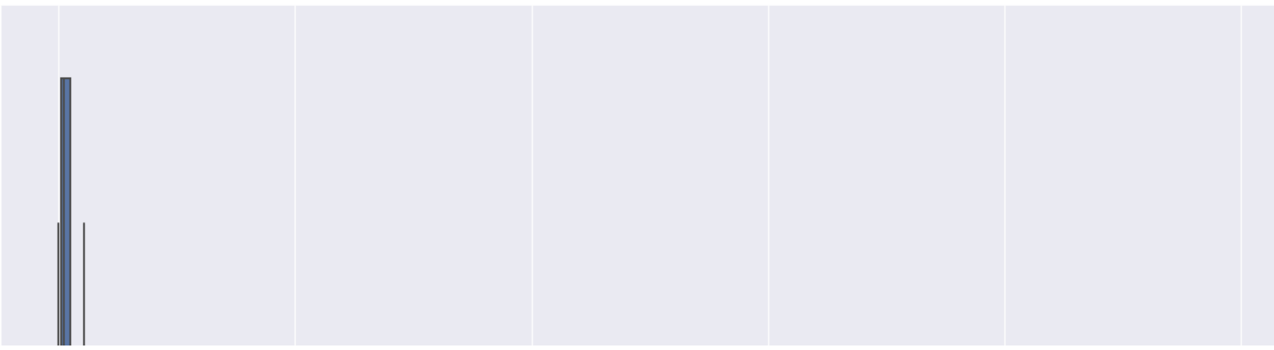

Retention
Churn
Frequency
12000
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
25000
net_margin

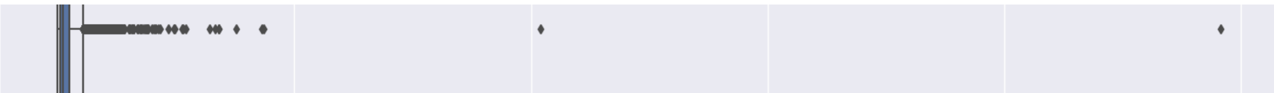
In [27]:
```
fig, axs = plt.subplots(nrows=3, figsize=(18, 35))
sns.boxplot(client_df["margin_gross_pow_ele"],ax= axs[0])
sns.boxplot(client_df["margin_net_pow_ele"], ax=axs[1])
sns.boxplot(client_df["net_margin"], ax=axs[2])
```

Out[27]: <AxesSubplot:xlabel='net_margin'>

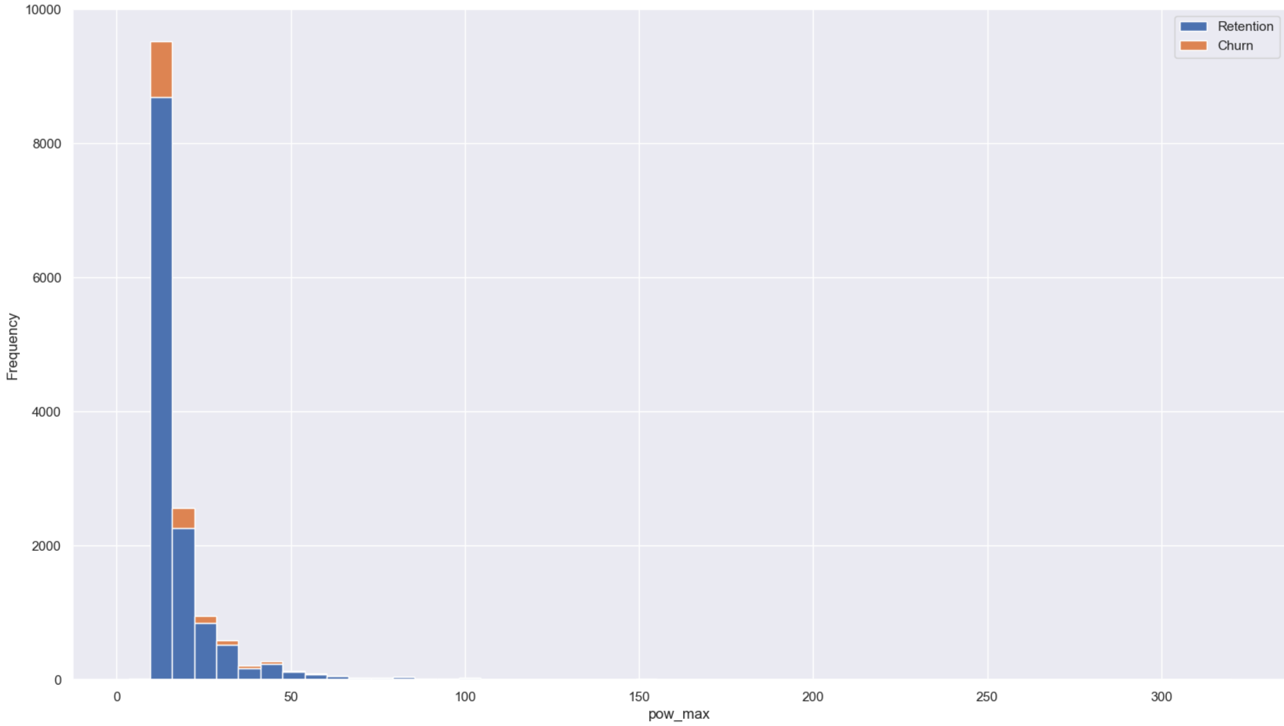

0
50
100
150
200
250
300
350
margin_gross_pow_ele


0
50
100
150
200
250
300
350
margin_net_pow_ele





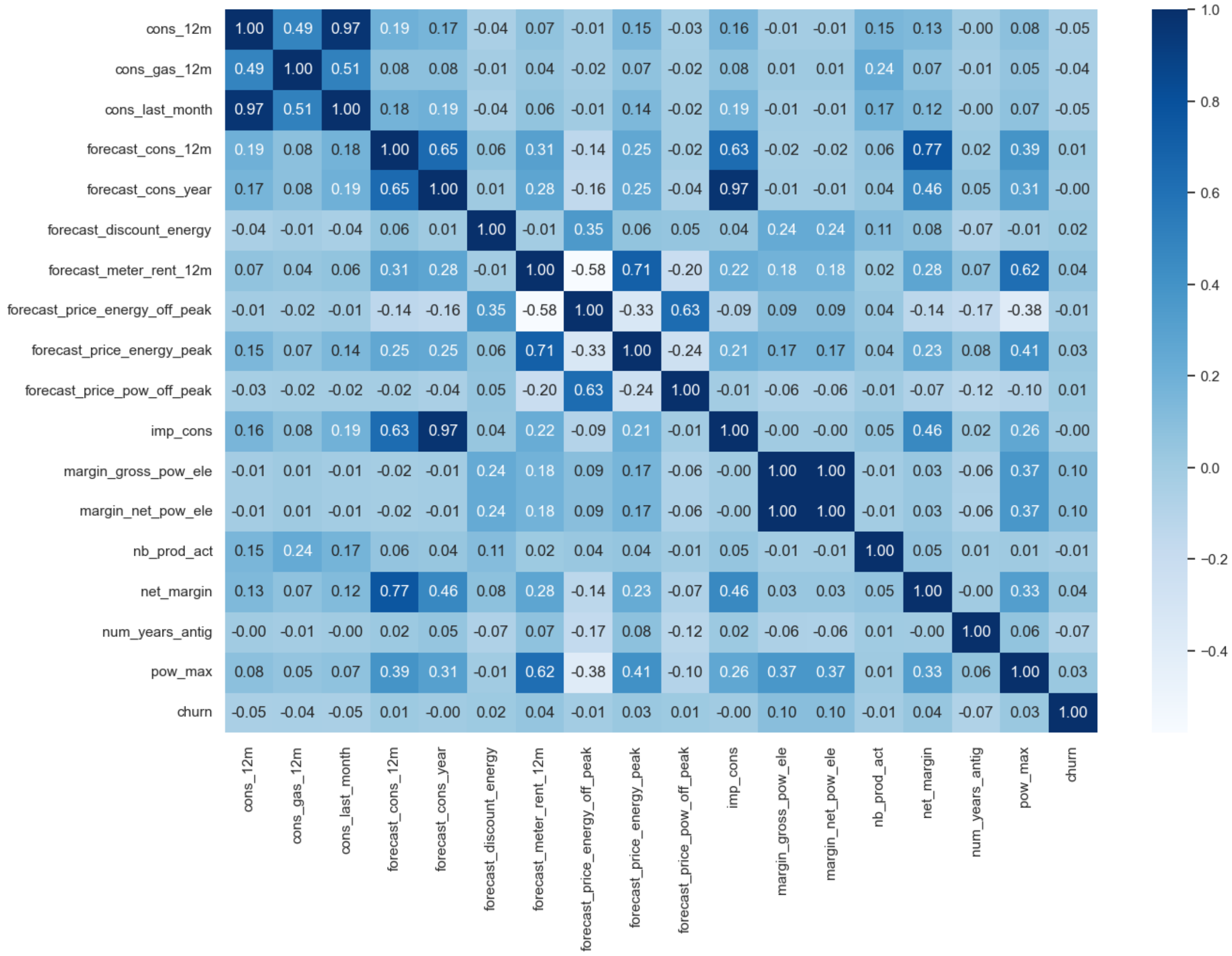
In [28]:
```
power = client_df[['id', 'pow_max', 'churn']]
fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)
```



plot the heatmap

In [33]:
```
corr = client_df.corr()
plt.figure(figsize=(15,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap='Blues',fmt=".2f" )
```

Out[33]: <AxesSubplot:>



- cons_last_month and cons_12m has high correlation
- forecast_cons_year and forecoast_cons_12m has high correlation
- imp_cons and forecast_cons_year have high correlation
- forecoast_cons_12m and net_margin have high correlation

Findings:

*** Analyzed consumption, forecast, margin and power related columns from client data**

*** We have highly positively skews data in almost all areas. This need to be properly handled before doing data modelling**

*** Analysis shows that we have around 9.7% of customers have churned**

Suggestions:

*** Churning is likely to happen when competitor has given good offers at the same price of here**

*** We can ask customer feedback, to check for the suggestions/complaints from their side**

*** We can provide extra benefits and offers for people who subscribed with the company for a long/specified period of time. This can help to reduce the churn percentage**