0.0 Import packages

In [1]:

```
import pandas as pd
pd.set_option ('display.max_columns', 50)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
import datetime
import pickle
import os

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

1.0 Load Data

In [2]:

```
train_data = pd.read_csv("/Users/adampillai/Documents/Forage BCG Internship/Task 2/ml_case
history_data = pd.read_csv ("/Users/adampillai/Documents/Forage BCG Internship/Task 2/ml_c
churn_data = pd.read_csv ("/Users/adampillai/Documents/Forage BCG Internship/Task 2/ml_cas
```

In [3]:

```
train_data.head()
```

Out[3]:

	id	activity_new	campaign_disc_ele	
0	48ada52261e7cf58715202705a0451c9	esoiiifxdlbkcsluxmfuacbdckommixw	NaN	lmkebamcaaclubfx
1	24011ae4ebbe3035111d65fa7c15bc57	NaN	NaN	foosdfpfkusacim
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	NaN	
3	764c75f661154dac3a6c254cd082ea7d	NaN	NaN	foosdfpfkusacim
4	bba03439a292a1e166f80264c16191cb	NaN	NaN	lmkebamcaaclubfx

In [4]:

```
history_data.head()
```

Out[4]:

	id	price_date	price_p1_var	price_p2_var	price_p3_var	price_p1_fix	p
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.0	44.266931	
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.0	44.266931	
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.0	44.266931	
3	038af19179925da21a25619c5a24b745	2015-04-01	0.149626	0.0	0.0	44.266931	
4	038af19179925da21a25619c5a24b745	2015-05-01	0.149626	0.0	0.0	44.266931	

In [5]:

```
churn_data.head()
```

Out[5]:

	id	churn
0	48ada52261e7cf58715202705a0451c9	0
1	24011ae4ebbe3035111d65fa7c15bc57	1
2	d29c2c54acc38ff3c0614d0a653813dd	0
3	764c75f661154dac3a6c254cd082ea7d	0
4	bba03439a292a1e166f80264c16191cb	0

1.1 Merge two dataset

In [6]:

```
train= pd.merge(train_data,churn_data, on="id")
```

In [7]:

```
train.head()
```

Out[7]:

	id	activity_new	campaign_disc_ele	
0	48ada52261e7cf58715202705a0451c9	esoiiifxdlbkcsluxmfuacbdckommixw	NaN	lmkebamcaaclubfx
1	24011ae4ebbe3035111d65fa7c15bc57	NaN	NaN	foosdfpfkusacin
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	NaN	
3	764c75f661154dac3a6c254cd082ea7d	NaN	NaN	foosdfpfkusacin
4	bba03439a292a1e166f80264c16191cb	NaN	NaN	lmkebamcaaclubfx

1.2 Missing Value in (%)

In [8]:

```
pd.DataFrame({"Missing Values (%)": train.isnull().sum()/len(train.index)*100})
```

Out[8]:

	Missing Values (%)
id	0.000000
activity_new	59.300447
campaign_disc_ele	100.000000
channel_sales	26.205268
cons_12m	0.000000
cons_gas_12m	0.000000
cons_last_month	0.000000
date_activ	0.000000
date_end	0.012425
date_first_activ	78.205765

	Missing Values (%)
date_modif_prod	0.975398
date_renewal	0.248509
forecast_base_bill_ele	78.205765
forecast_base_bill_year	78.205765
forecast_bill_12m	78.205765
forecast_cons	78.205765
forecast_cons_12m	0.000000
forecast_cons_year	0.000000
forecast_discount_energy	0.782803
forecast_meter_rent_12m	0.000000
forecast_price_energy_p1	0.782803
forecast_price_energy_p2	0.782803
forecast_price_pow_p1	0.782803
has_gas	0.000000
imp_cons	0.000000
margin_gross_pow_ele	0.080765
margin_net_pow_ele	0.080765
nb_prod_act	0.000000
net_margin	0.093191
num_years_antig	0.000000
origin_up	0.540507
pow_max	0.018638
churn	0.000000

In [9]:

```
pd.DataFrame({"Missing Values (%)": history_data.isnull().sum()/len(history_data.index)*1(
```

Out[9]:

	Missing Values (%)
id	0.000000
price_date	0.000000
price_p1_var	0.704138
price_p2_var	0.704138
price_p3_var	0.704138
price_p1_fix	0.704138
price_p2_fix	0.704138
price_p3_fix	0.704138

3.0 Data Visualization

3.1 Churning Rate

In [10]:

```
churn = train [["id", "churn"]]
churn.columns = ["Companies", "churn"]
```

In [11]:

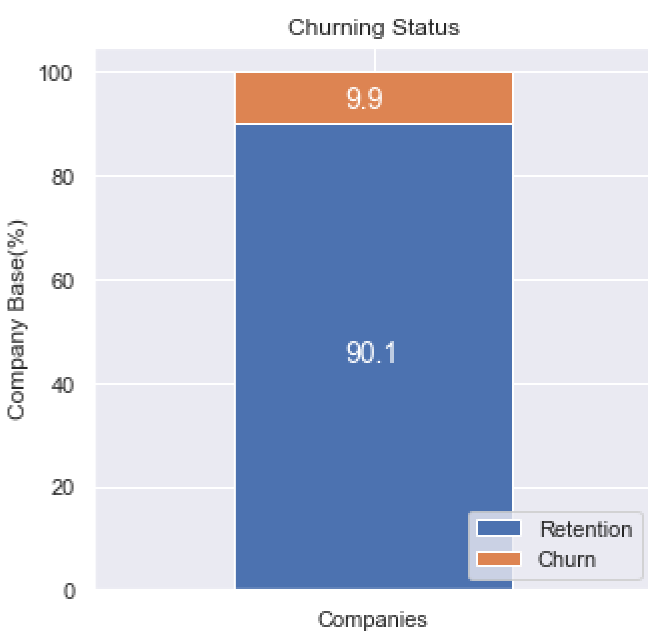
```
def plot_stacked_bars(dataframe, title_, size_=(18,10), rot_=0, legend_="upper right"):

    ax = dataframe.plot(kind= "bar",
                        stacked = True,
                        figsize=size_,
                        rot=rot_,
                        title=title_)

    annotate_stacked_bars(ax, textsize=14)
    plt.legend(["Retention", "Churn"], loc=legend_)

    plt.ylabel("Company Base(%)")
    plt.show

def annotate_stacked_bars(ax, pad=0.98, colour="white", textsize=13):

    for p in ax.patches:

        value= str(round(p.get_height(),1))

        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 [12]:

```
churn_total = churn.groupby(churn["churn"]).count()
churn_percentage = churn_total/churn_total.sum()*100
```

In [13]:

```
plot_stacked_bars(churn_percentage.transpose(), "Churning Status", (5,5), legend_="lower r
```

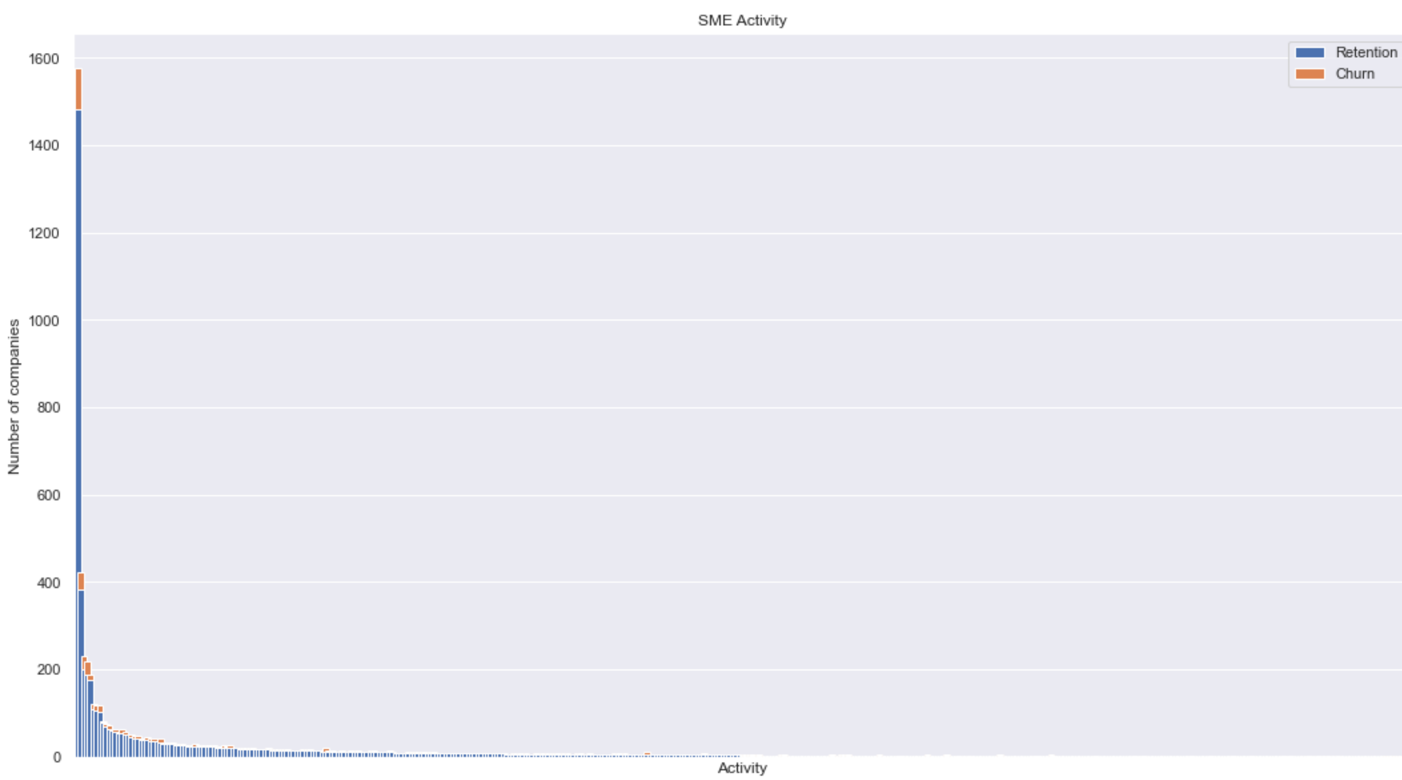


3.2 SME Activity

In [14]:
```
activity = train[["id","activity_new", "churn"]]
```

In [15]:
```
activity = activity.groupby([activity["activity_new"],
 activity["churn"]])["id"].count().unstack(level=1).sort_values(by=[0], ascending=False)
```

In [16]:
```
activity.plot(kind="bar",
 figsize=(18, 10),
 width=2,
 stacked=True,
 title="SME Activity")
# Labels
plt.ylabel("Number of companies")
plt.xlabel("Activity")
# Rename legend
plt.legend(["Retention", "Churn"], loc="upper right")
# Remove the label for the xticks as the categories are encoded and we can't draw any mea
plt.xticks([])
plt.show()
```



The distribution of the classes over the labeled data despite the lack of 60% of the entries.

We see churn is not specifically related to any SME cateogry in particular.

Note: Not showing the labels in the x-axis to facilitate visualization

If we take a look at the values percentage-wise

In [17]:
```
activity_total = activity.fillna(0)[0]+activity.fillna(0)[1]
activity_percentage = activity.fillna(0)[1]/(activity_total)*100
pd.DataFrame({"Percentage churn": activity_percentage,
 "Total companies": activity_total }).sort_values(by="Percentage churn",
 ascending=False).head(10)
```

Out[17]:

activity_new	Percentage churn	Total companies
xwkaesbkfsacseixxksofpddwfkbobki	100.000000	1.0
wkwdccuiboaeaalcaawlwmldiwmpewma	100.000000	1.0
ikiucmkuisupefxcxfxxulkpwssppfuo	100.000000	1.0
opoiuuwdmxdssidluooopfswlkkkcsxf	100.000000	1.0
pfcocskbxlmofswiflsbcefcpufbopuo	100.000000	2.0
oeacexidmflusdkwuuicmpiaklkxulxm	100.000000	1.0
wceaopxmdpccxfmcdpopulcaubcxibuw	100.000000	1.0
kmlwkmxoocpieebifumobckeafmidpxf	100.000000	1.0
cwouwoubfifoafkxifokoidcuoamebea	66.666667	3.0
wfiuolfffsekuoimxdsasfwcmwssewoi	50.000000	4.0

3.3 Sales Channel

In [18]:

```
channel = train[["id","channel_sales", "churn"]]
```

In [19]:

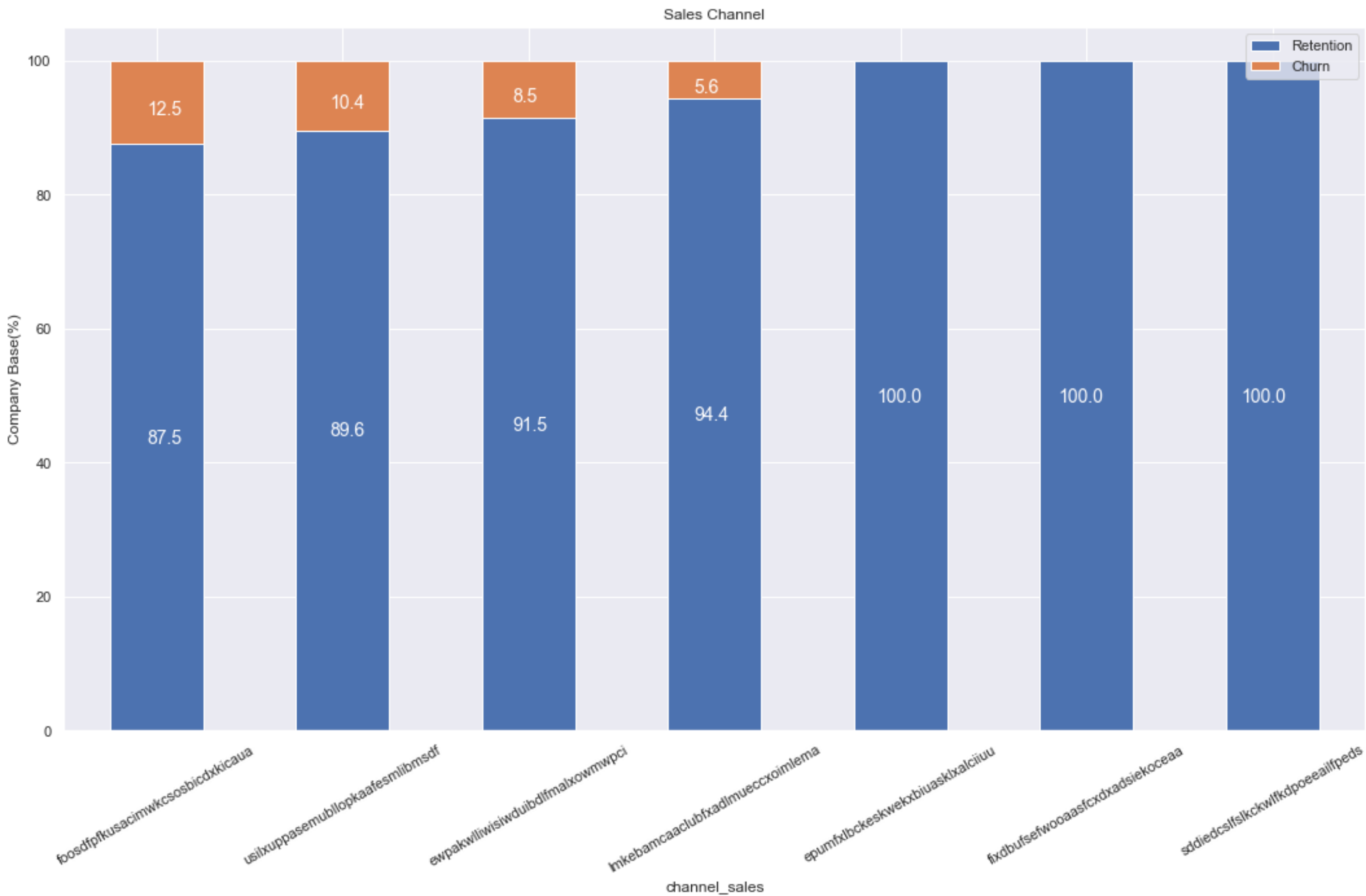
```
channel = channel.groupby([channel["channel_sales"],
 channel["churn"]])["id"].count().unstack(level=1).fillna(0)
```

In [20]:

```
channel_churn = (channel.div(channel.sum(axis=1), axis=0)*100).sort_values(by=[1], ascendi
```

In [21]:

```
plot_stacked_bars(channel_churn, "Sales Channel", rot_=30)
```



In [22]:

```
channel_total = channel.fillna(0)[0]+channel.fillna(0)[1]
channel_percentage = channel.fillna(0)[1]/(channel_total)*100
pd.DataFrame({"Churn percentage": channel_percentage,
 "Total companies": channel_total }).sort_values(by="Churn percentage",
 ascending=False).head(10)
```

Out[22]:

channel_sales	Churn percentage	Total companies
foosdfpfkusacimwkcsosbicdxkicaua	12.498306	7377.0
usilxuppasemubllopkaafesmlibmsdf	10.387812	1444.0
ewpakwlliwisiwduibdlfmalxowmwpci	8.488613	966.0
lmkebamcaaclubfxadlmueccxoimlema	5.595755	2073.0
epumfxlbckeskwekxbiuasklxalciiuu	0.000000	4.0
fixdbufsefwooaasfcxdxadsiekoceaa	0.000000	2.0
sddiedcslfslkckwlfkdpoeeailfpeds	0.000000	12.0

3.4 Annual and Monthly Consumption

In [23]:

```
consumption = train[["id","cons_12m", "cons_gas_12m","cons_last_month", "imp_cons", "has_g
```

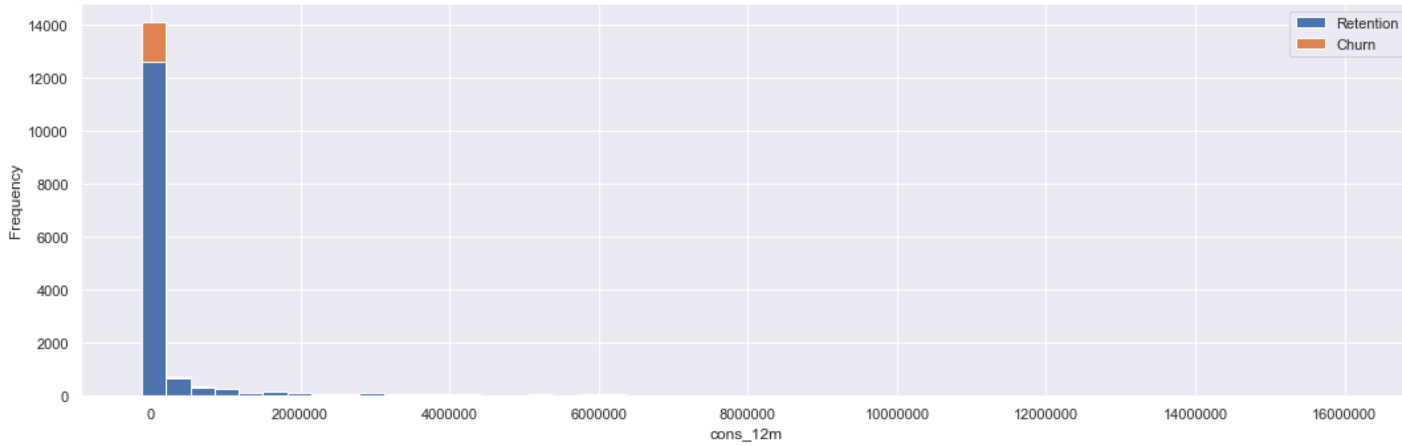
In [24]:

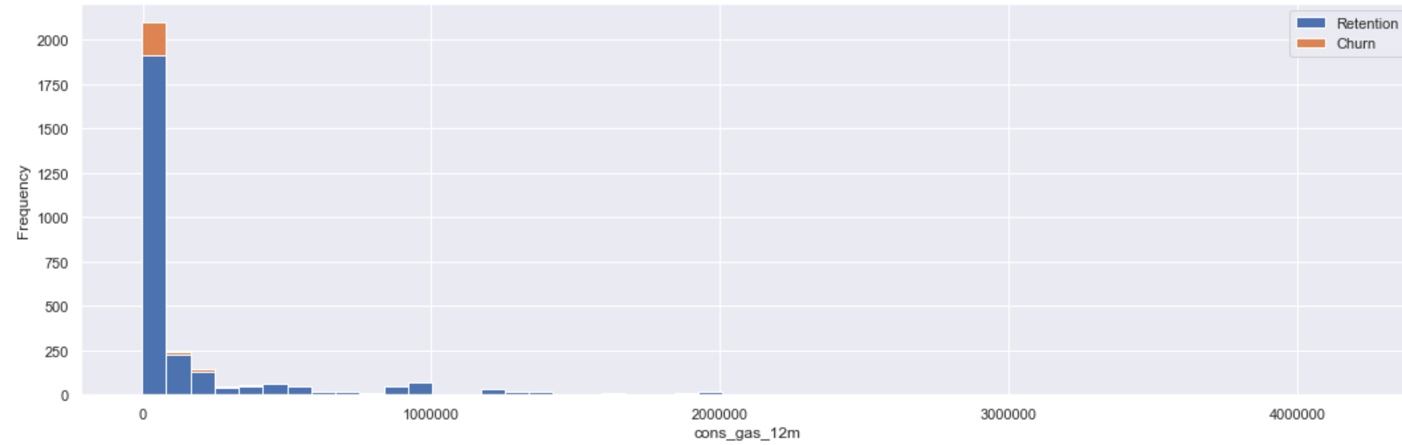
```
def plot_distribution(dataframe, column, ax, bins_=50):
 """
 Plot variable distirbution in a stacked histogram of churned or retained company
 """
 # Create a temporal dataframe with the data to be plot
```

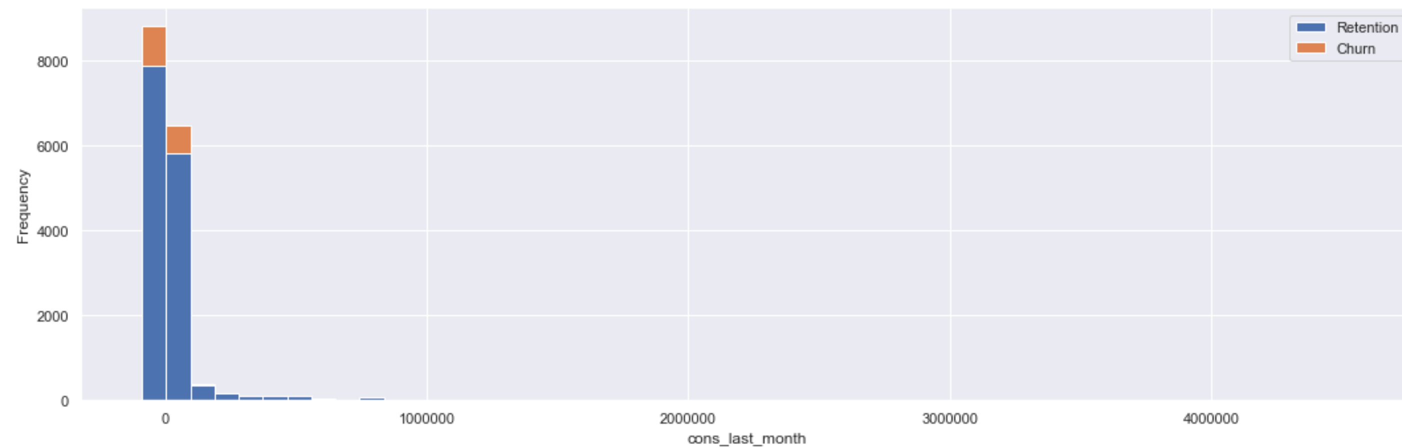
```python
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "Churn":dataframe[dataframe["churn"]==1][column]})
    # Plot the histogram
    temp[["Retention","Churn"]].plot(kind='hist', bins=bins_, ax=ax, stacked=True)
    # X-axis label
    ax.set_xlabel(column)
    # Change the x-axis to plain style
    ax.ticklabel_format(style='plain', axis='x')
```

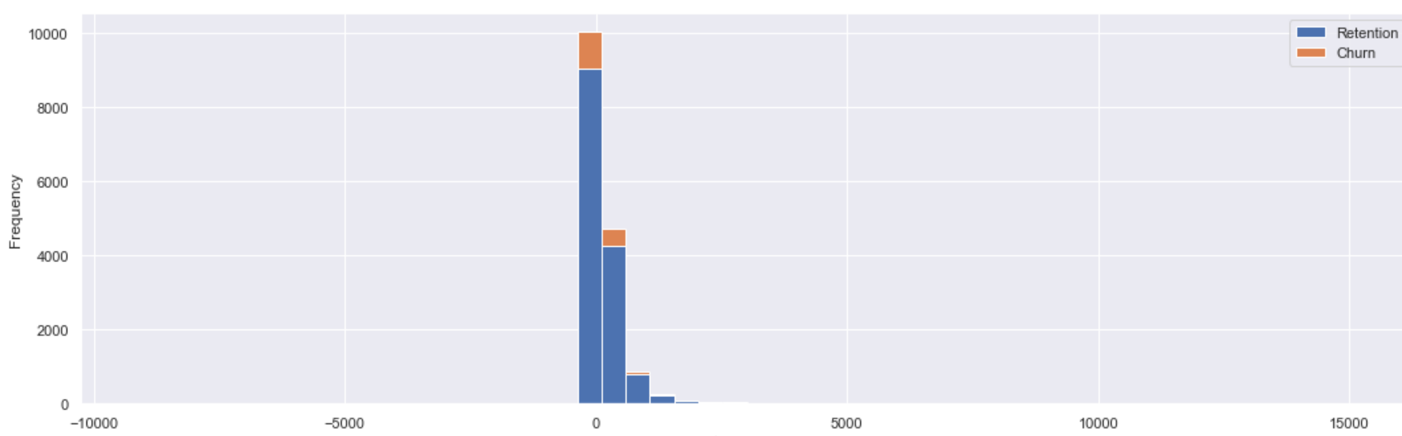
In [25]:

```python
 fig, axs = plt.subplots(nrows=4, figsize=(18,25))
# Plot histogram
plot_distribution(consumption, "cons_12m", axs[0])
# Note that the gas consumption must have gas contract
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])
```







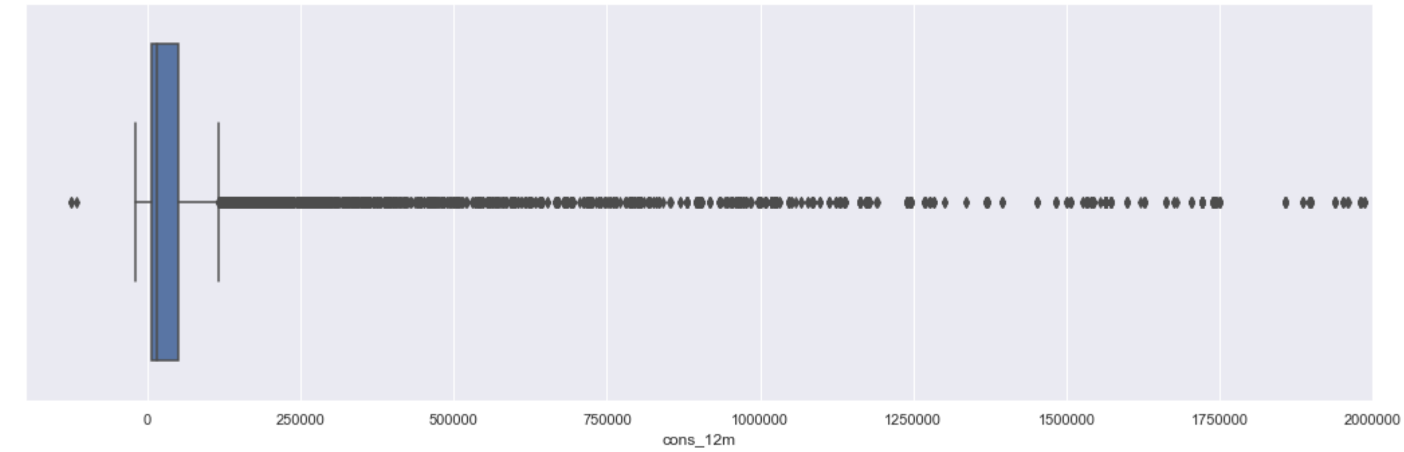


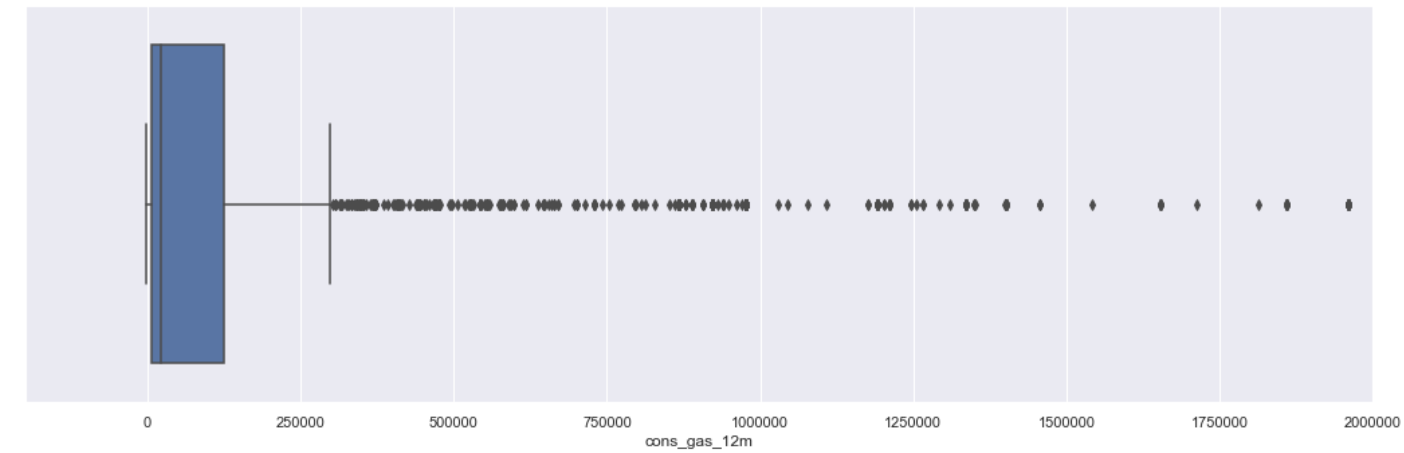
We can clearly see in here that the consumption data is highly skewed to the right, presenting a very long right-tail towards the higher values of the distribution.

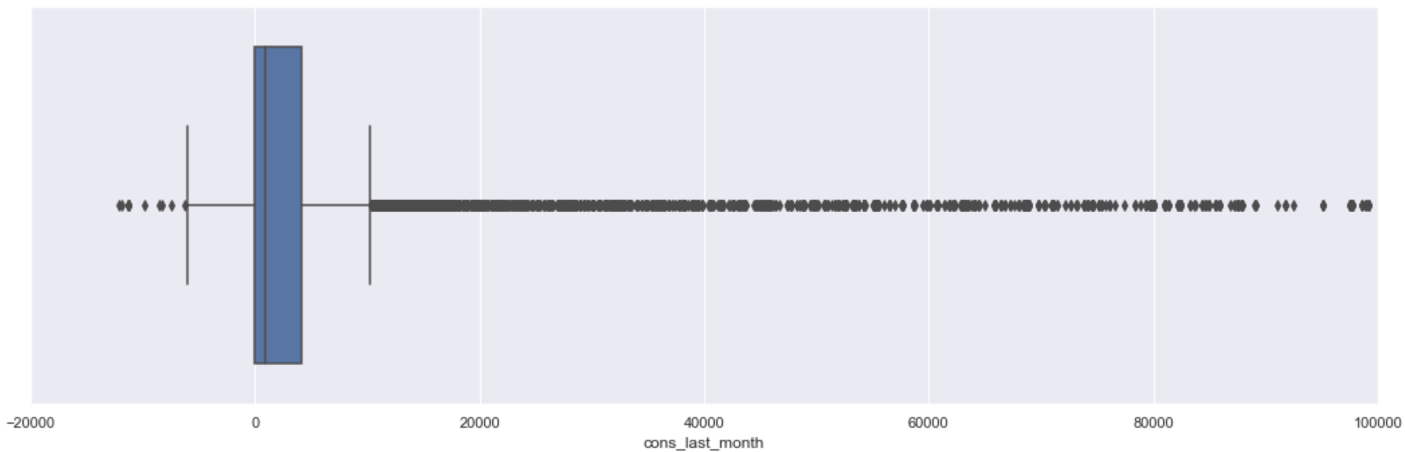
The values on the higher end and lower ends of the distribution are likely to be outliers. We can use a standard plot to visualise the outliers in more detail. A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell us about our outliers and what their values are. It can also tell us if our data is symmetrical, how tightly our data is grouped, and if and how our data is skewed.

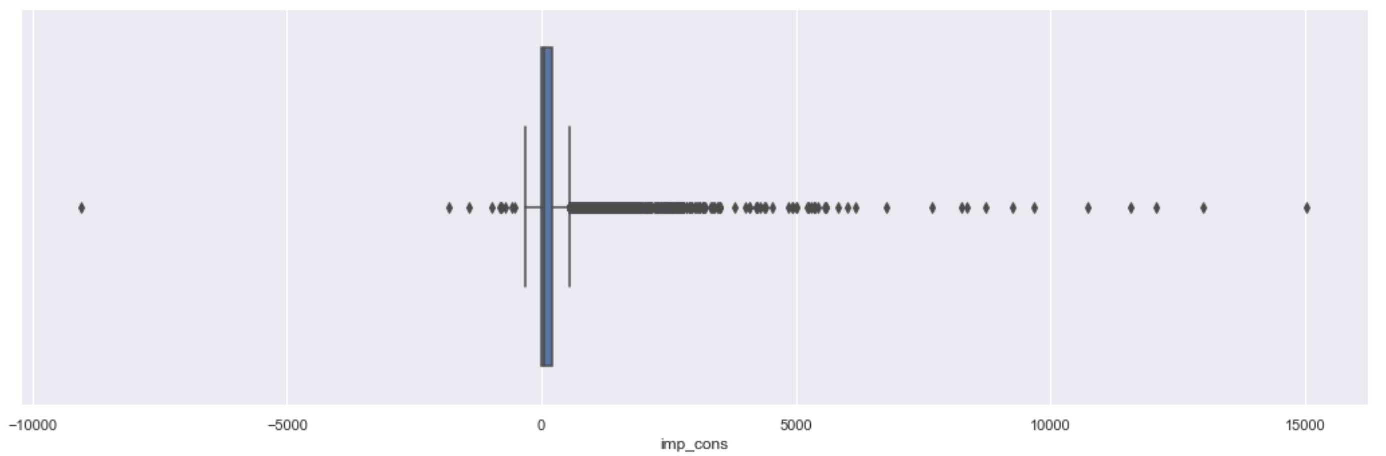
In [26]:

```
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()
```









Dates

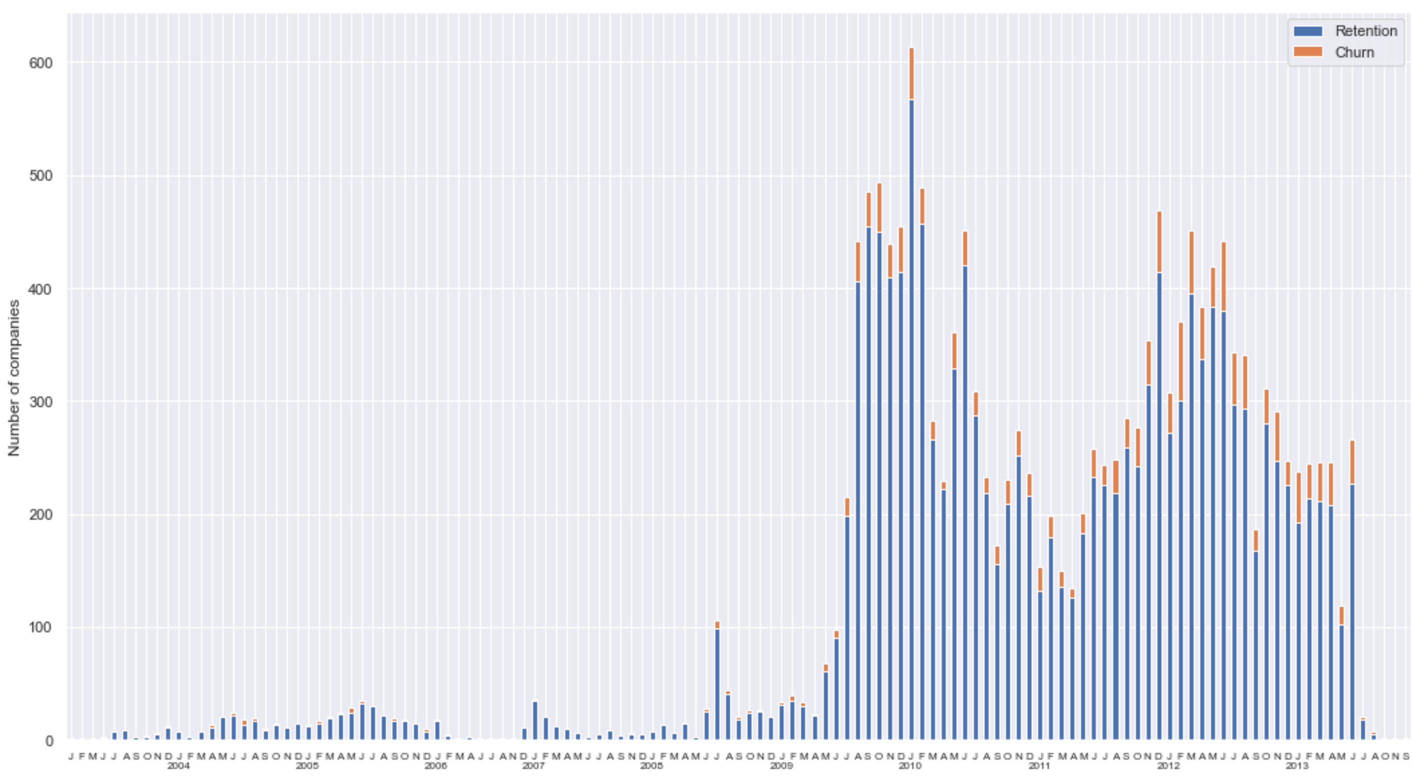
In [27]:

```
dates = train[["id","date_activ","date_end", "date_modif_prod","date_renewal","churn"]].c
```

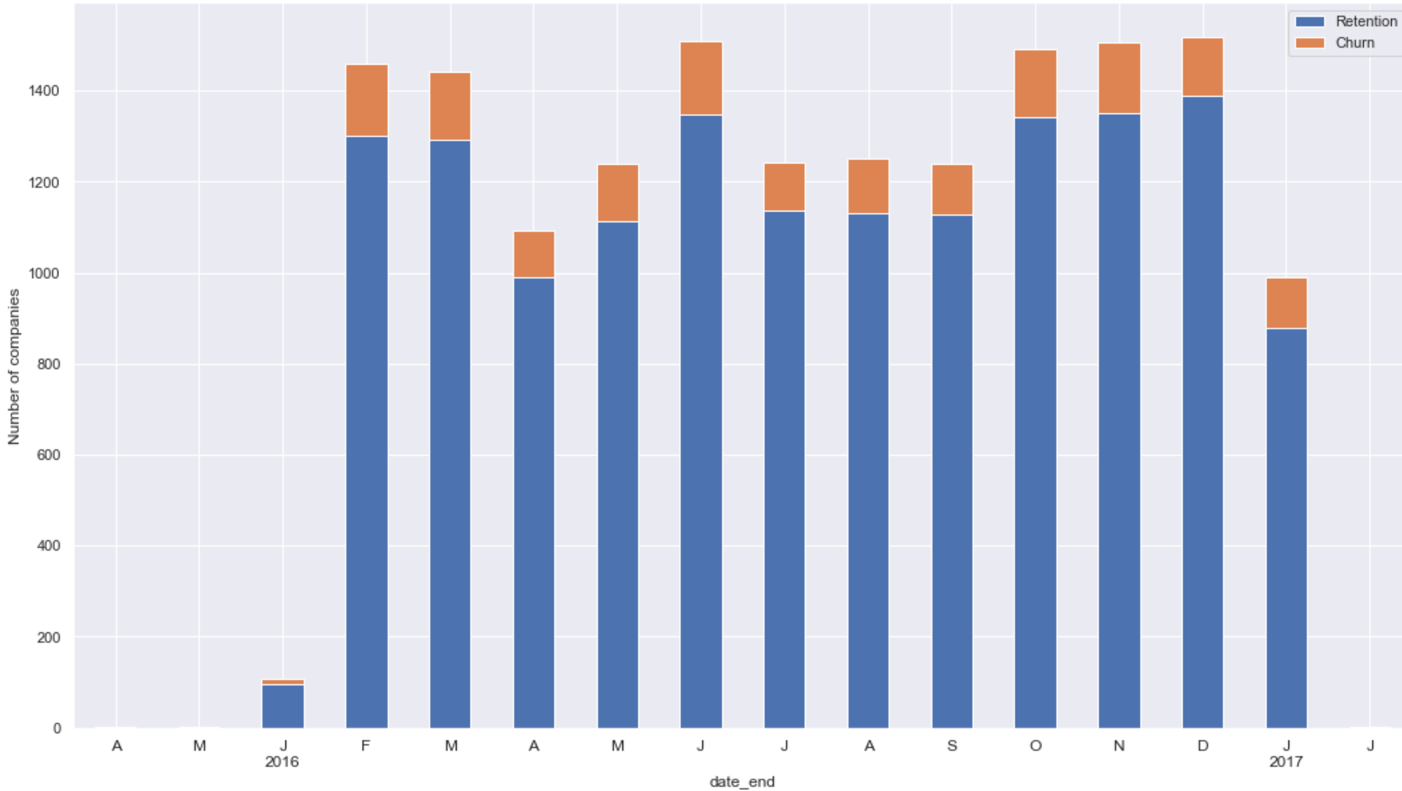
In [28]:
```python
# Transform date columns to datetime type
dates["date_activ"] = pd.to_datetime(dates["date_activ"], format='%Y-%m-%d')
dates["date_end"] = pd.to_datetime(dates["date_end"], format='%Y-%m-%d')
dates["date_modif_prod"] = pd.to_datetime(dates["date_modif_prod"], format='%Y-%m-%d')
dates["date_renewal"] = pd.to_datetime(dates["date_renewal"], format='%Y-%m-%d')
```

In [29]:
```python
def plot_dates(dataframe, column, fontsize_=12):
    """
    Plot monthly churn and retention distribution
    """
    # Group by month
    temp = dataframe[[column,
                      "churn",
                      "id"]].set_index(column).groupby([pd.Grouper(freq='M'), "churn"]).
    # Plot
    ax=temp.plot(kind="bar", stacked=True, figsize=(18,10), rot=0)
    # Change x-axis labels to months
    ax.set_xticklabels(map(lambda x: line_format(x), temp.index))
    # Change xlabel size
    plt.xticks(fontsize=fontsize_)
    # Rename y-axis
    plt.ylabel("Number of companies")
    # Rename legend
    plt.legend(["Retention", "Churn"], loc="upper right")
    plt.show()

def line_format(label):
    """
    Convert time label to the format of pandas line plot
    """
    month = label.month_name()[:1]
    if label.month_name() == "January":
        month += f'\n{label.year}'
    return month
```

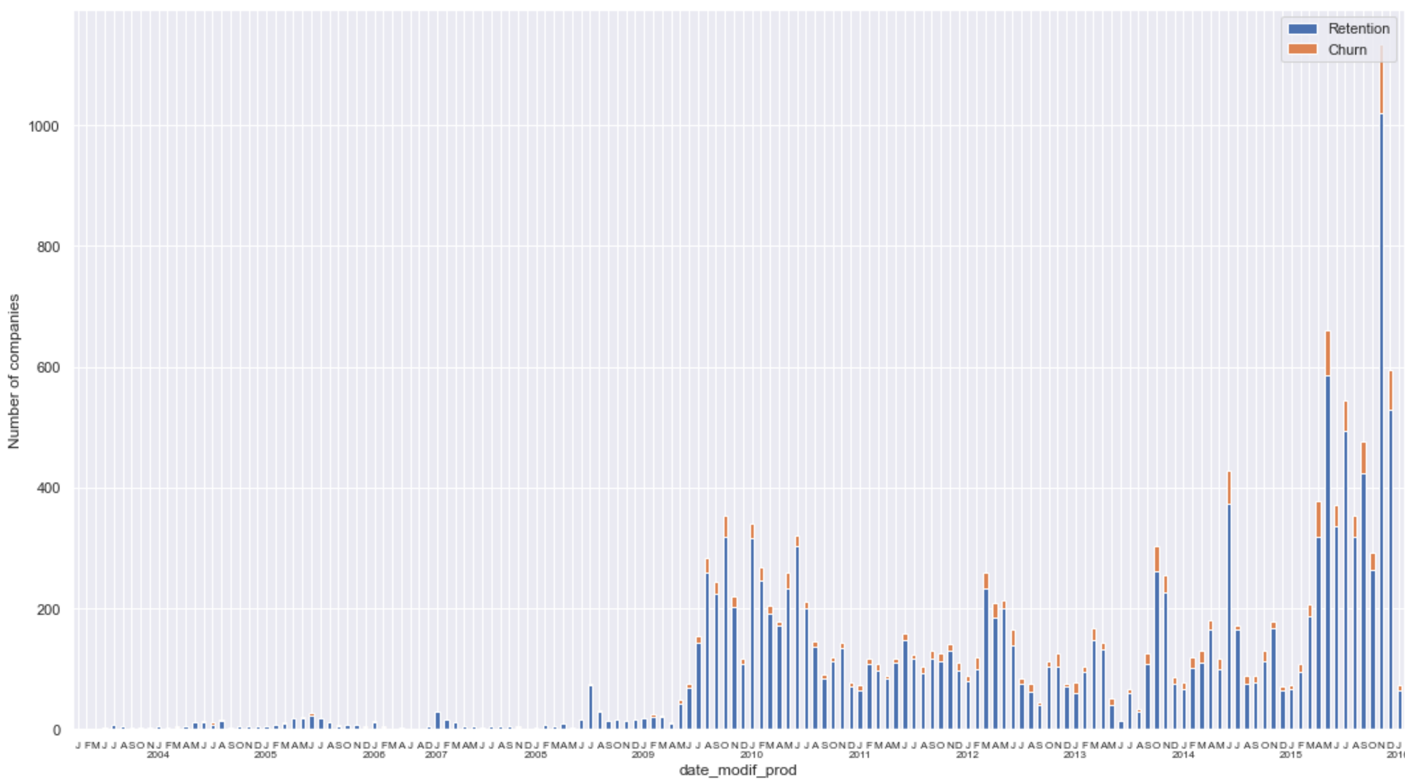
In [30]:
```python
plot_dates(dates, "date_activ", fontsize_=8)
```



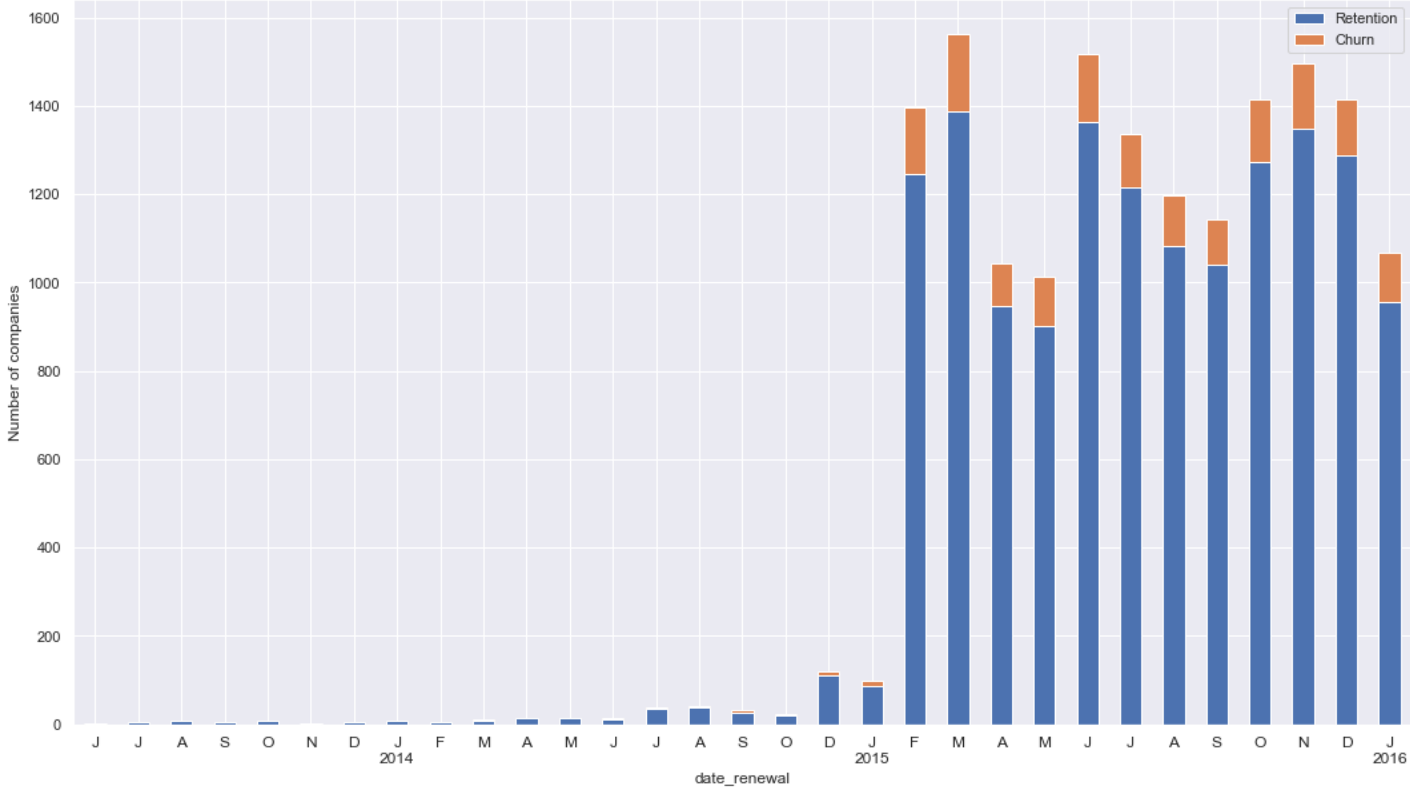
In [31]:
```
plot_dates(dates, "date_end")
```



In [32]:
```
plot_dates(dates, "date_modif_prod", fontsize_=8)
```



In [33]:
```
plot_dates(dates, "date_renewal")
```



As a remark in here, we can visualize the distribution of churned companies according to the date. However, this does not provide us with any useful insight. We will create a new feature using the raw dates provided in the next exercise.

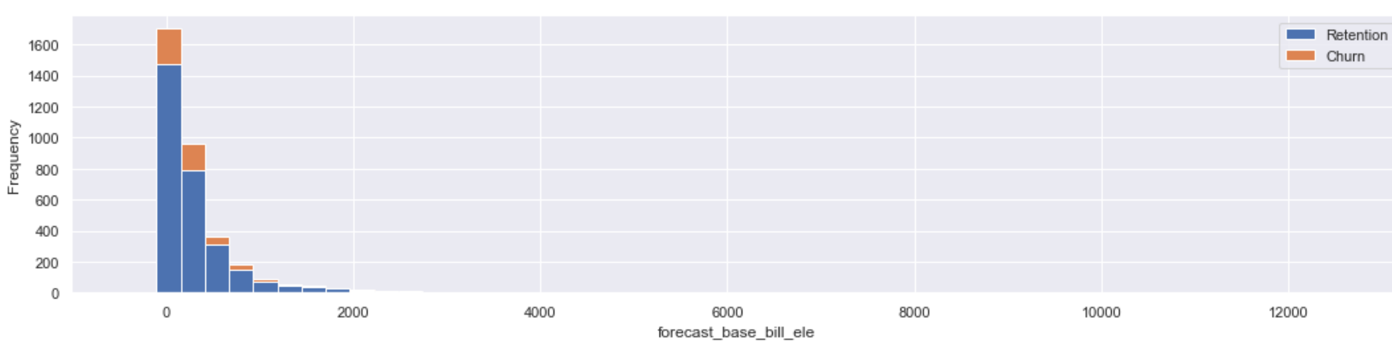
3.5 Forecating

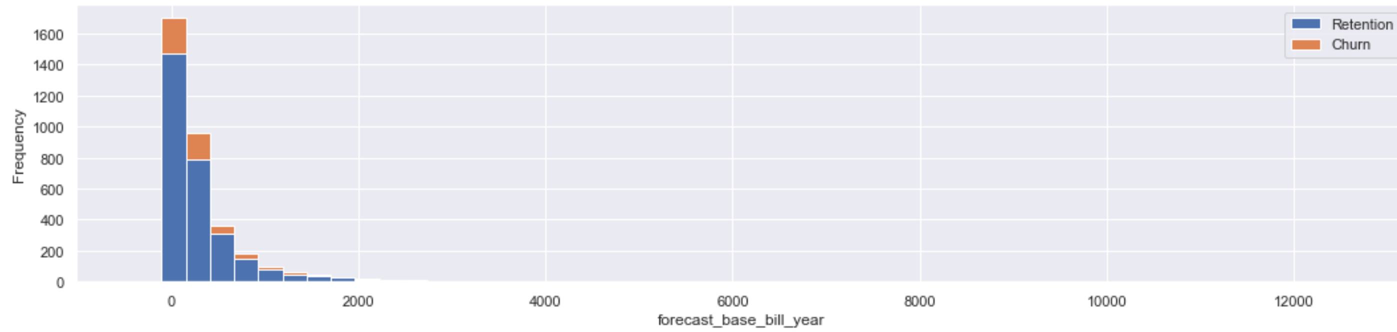
In [34]:

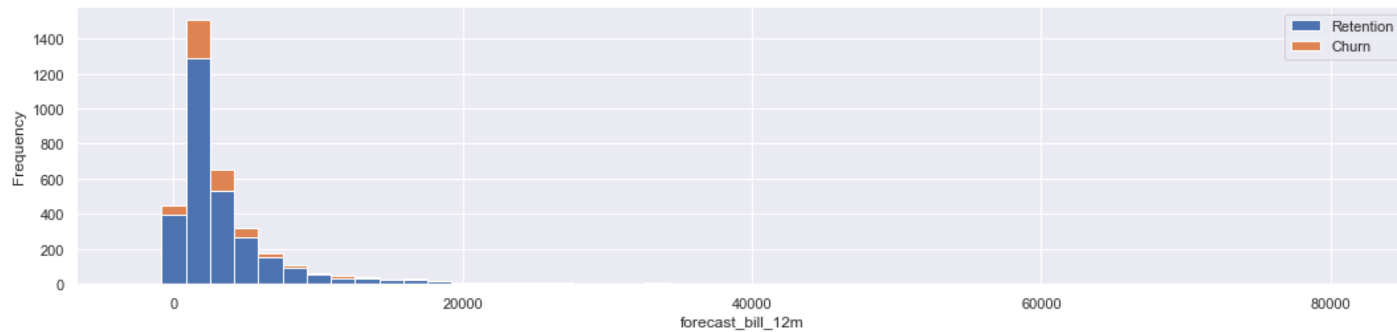
```
forecast = train[["id","forecast_base_bill_ele","forecast_base_bill_year",
                  "forecast_bill_12m","forecast_cons","forecast_cons_12m",
                  "forecast_cons_year","forecast_discount_energy","forecast_meter_rent_12m'
                  "forecast_price_energy_p1","forecast_price_energy_p2",
                  "forecast_price_pow_p1","churn"]]
```

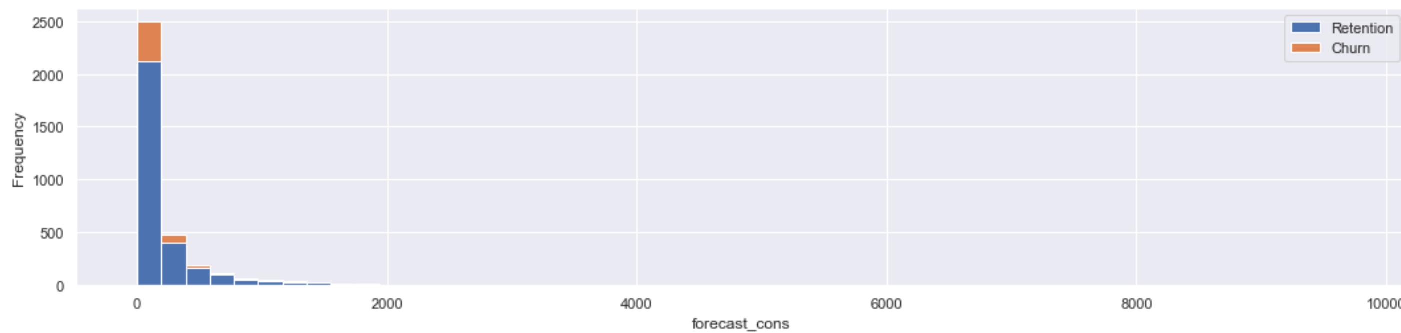
In [35]:

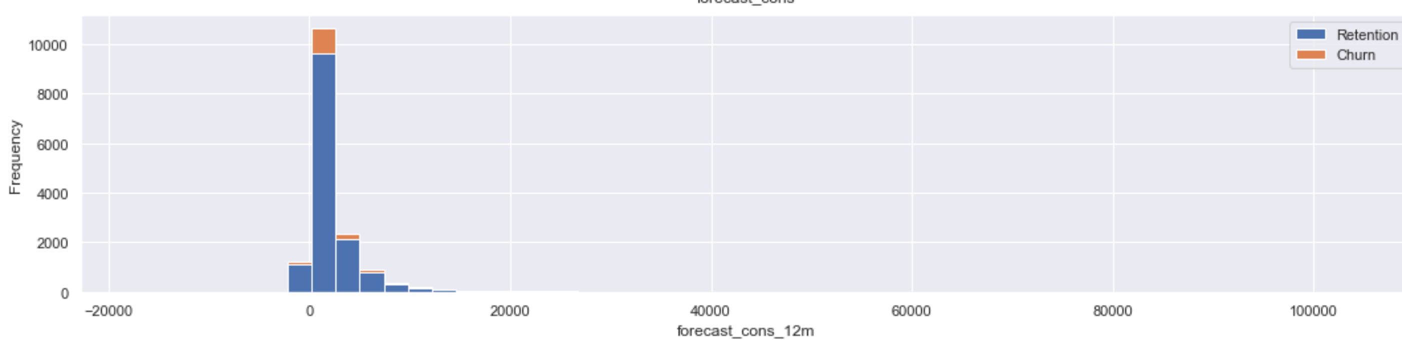
```
fig, axs = plt.subplots(nrows=11, figsize=(18,50))
# Plot histogram
plot_distribution(train, "forecast_base_bill_ele", axs[0])
plot_distribution(train, "forecast_base_bill_year", axs[1])
plot_distribution(train, "forecast_bill_12m", axs[2])
plot_distribution(train, "forecast_cons", axs[3])
plot_distribution(train, "forecast_cons_12m", axs[4])
plot_distribution(train, "forecast_cons_year", axs[5])
plot_distribution(train, "forecast_discount_energy", axs[6])
plot_distribution(train, "forecast_meter_rent_12m", axs[7])
plot_distribution(train, "forecast_price_energy_p1", axs[8])
plot_distribution(train, "forecast_price_energy_p2", axs[9])
plot_distribution(train, "forecast_price_pow_p1", axs[10])
```

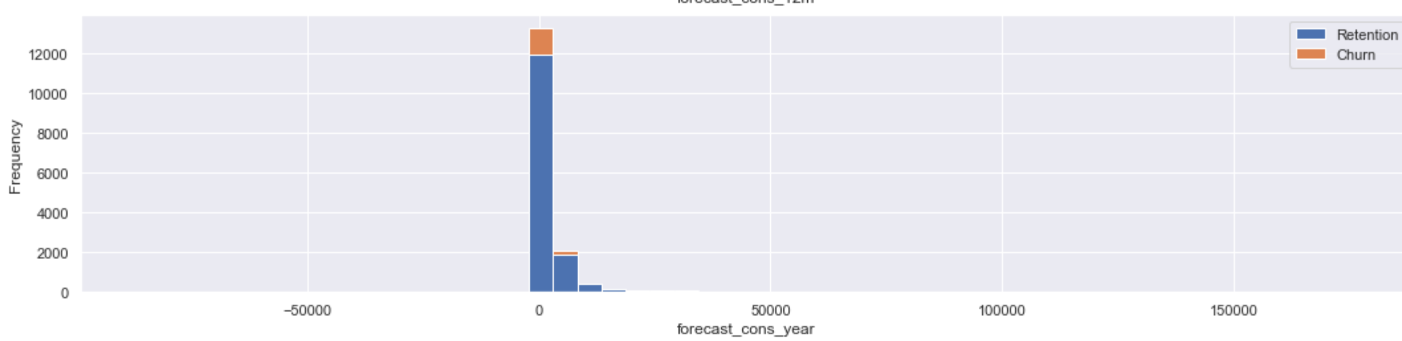


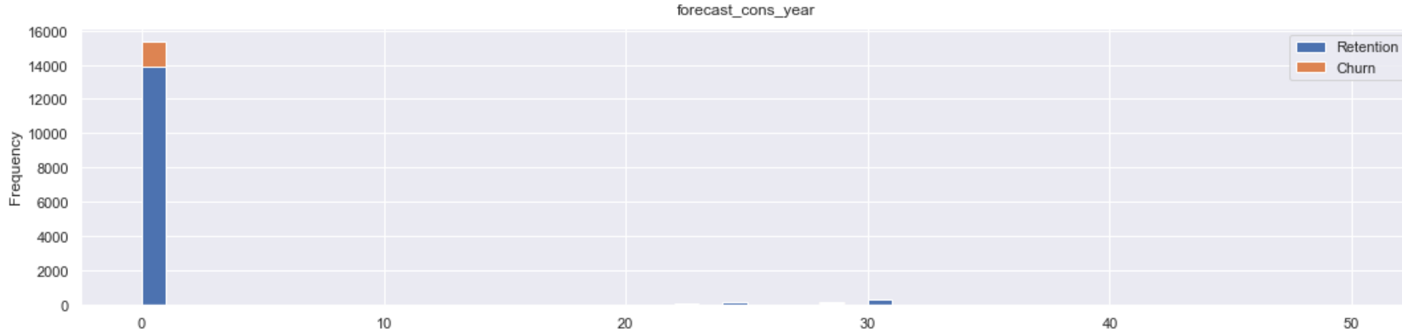

Retention
Churn
Frequency
forecast_base_bill_year

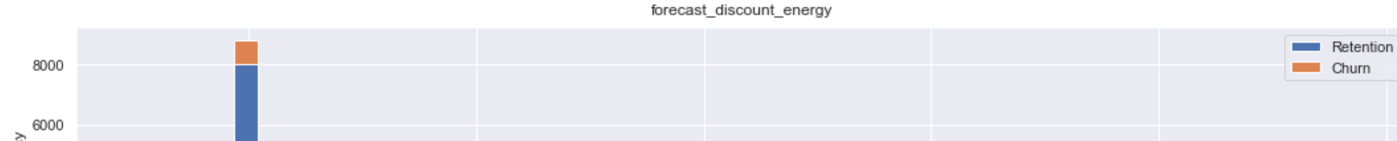

Retention
Churn
Frequency
forecast_bill_12m

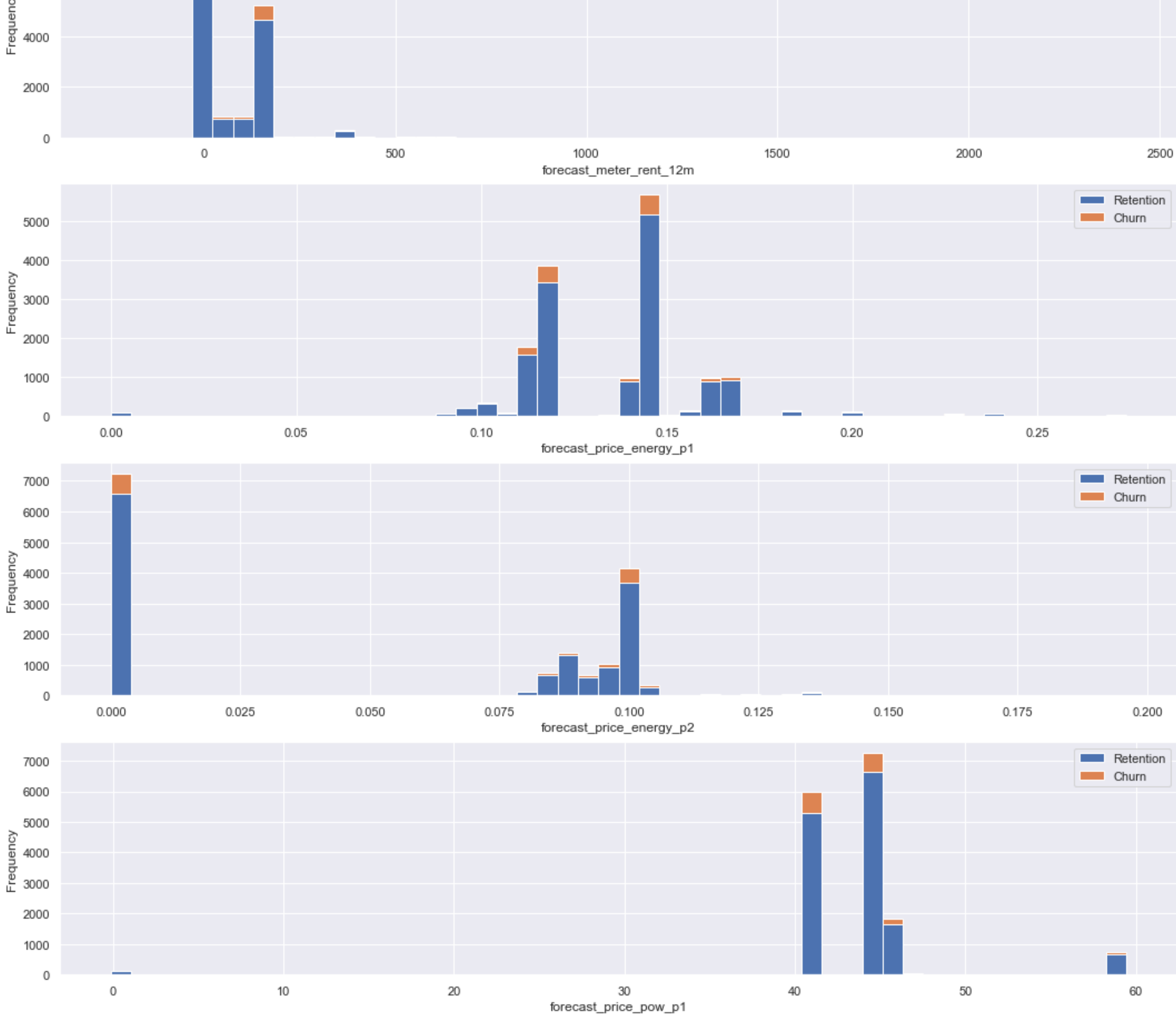

Retention
Churn
Frequency
forecast_cons


Retention
Churn
Frequency
forecast_cons_12m


Retention
Churn
Frequency
forecast_cons_year


Retention
Churn
Frequency
forecast_discount_energy


Retention
Churn



Similarly to the consumption plots, we can observe that a lot of the variables are highly skewed to the right, creating a very long tail on the higher values.

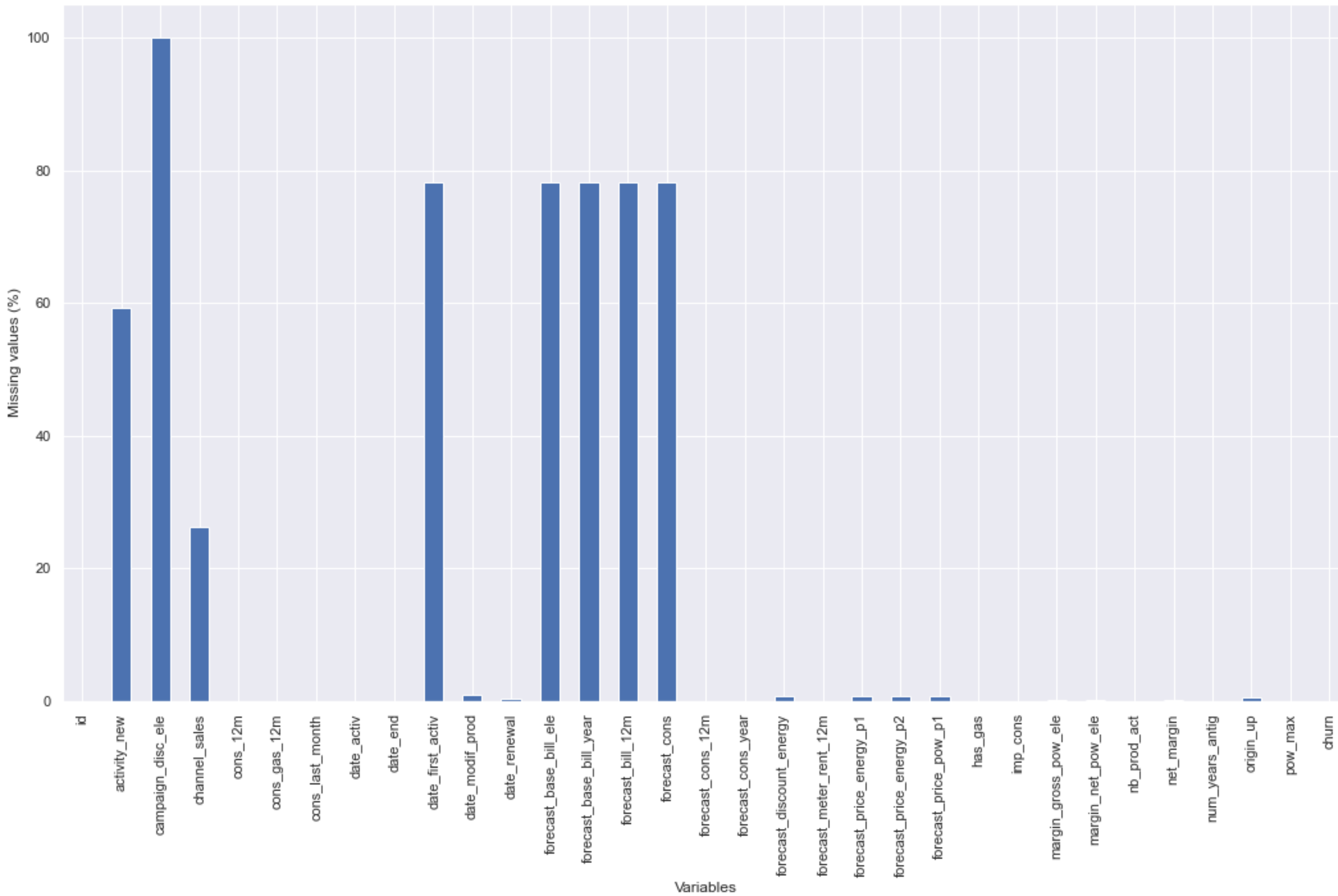
We will make some transformations to correct for this skewness

4.0 Data Cleaning

4.1 Missing Data

In [36]:

```
# Plot missing data
(train.isnull().sum()/len(train.index)*100).plot(kind="bar", figsize=(18,10))
# Set axis labels
plt.xlabel("Variables")
plt.ylabel("Missing values (%)")
plt.show()
```



In [37]:

```
train.drop(columns=["campaign_disc_ele", "date_first_activ",
 "forecast_base_bill_ele","forecast_base_bill_year",
"forecast_bill_12m", "forecast_cons"], inplace=True)
```

In [38]:

```
pd.DataFrame({"Dataframe colums": train.columns})
```

Out[38]:

	Dataframe colums
0	id
1	activity_new
2	channel_sales
3	cons_12m
4	cons_gas_12m
5	cons_last_month
6	date_activ
7	date_end
8	date_modif_prod
9	date_renewal
10	forecast_cons_12m
11	forecast_cons_year
12	forecast_discount_energy
13	forecast_meter_rent_12m

	Dataframe colums
14	forecast_price_energy_p1
15	forecast_price_energy_p2
16	forecast_price_pow_p1
17	has_gas
18	imp_cons
19	margin_gross_pow_ele
20	margin_net_pow_ele
21	nb_prod_act
22	net_margin
23	num_years_antig
24	origin_up
25	pow_max
26	churn

4.1 Duplicates

We want to make sure all the data we have is unique and we don't have any duplicated rows. For that, we're going to use the.duplicated()function in pandas.

This will tell us if there are any duplicated rows.

In [40]:

```
train[train.duplicated()]
```

Out[40]:

id	activity_new	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_m

5.0 Formatting Data

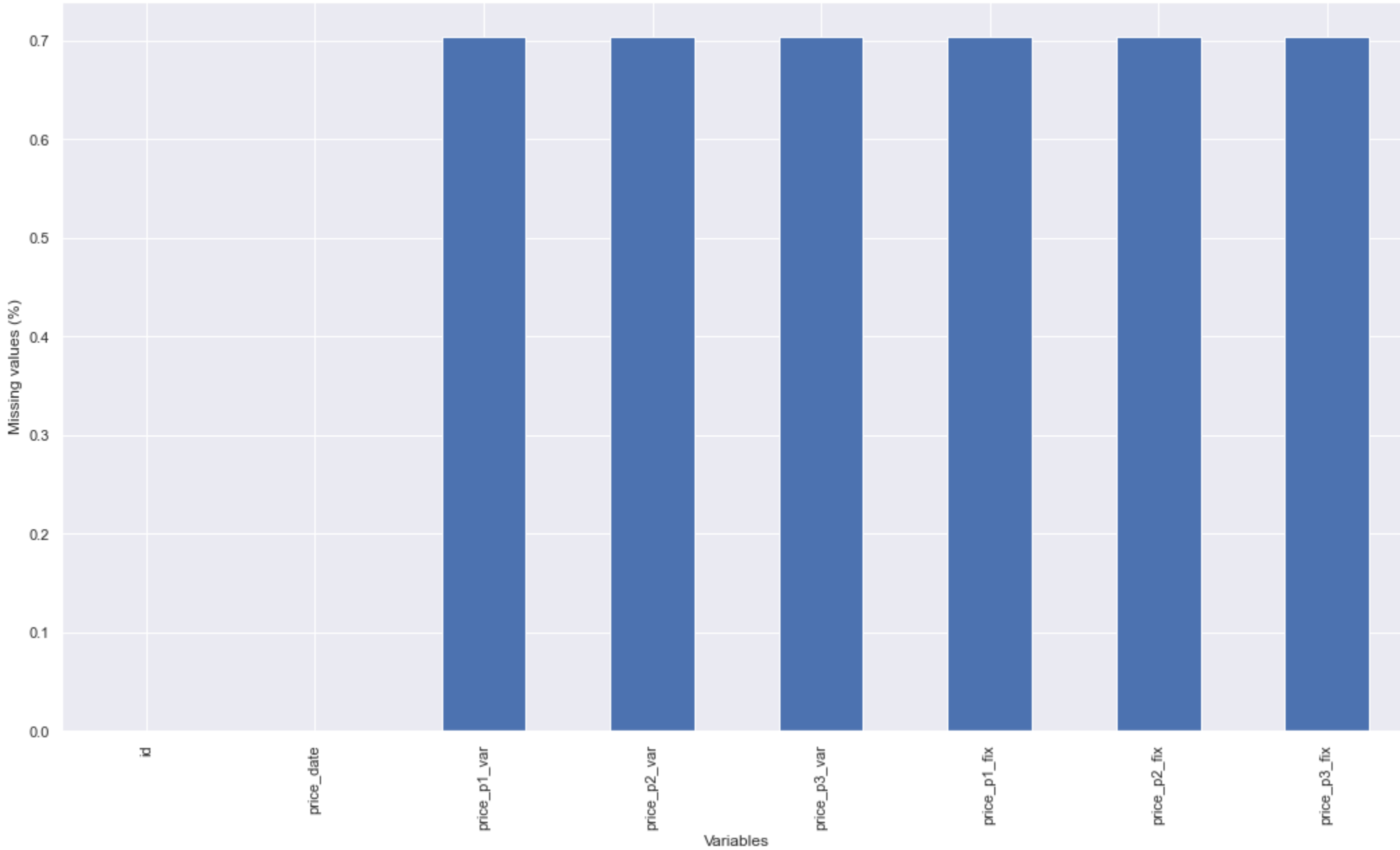
In [41]:

```
train.loc[train["date_modif_prod"].isnull(),"date_modif_prod"] = train["date_modif_prod"].
train.loc[train["date_end"].isnull(),"date_end"] = train["date_end"].value_counts().index
train.loc[train["date_renewal"].isnull(),"date_renewal"] = train["date_renewal"].value_cou
```

5.1 Identify Missing Data

In [42]:

```
missing_data_percentage = history_data.isnull().sum()/len(history_data.index)*100

missing_data_percentage.plot(kind="bar", figsize=(18,10))
# Set labels
plt.xlabel("Variables")
plt.ylabel("Missing values (%)")
plt.show()
```



There is not much data missing. Instead of removing the entries that are empty we will simply substitute them with the median.

In [43]:

```
history_data[history_data.isnull().any(axis=1)]
```

Out[43]:

	id	price_date	price_p1_var	price_p2_var	price_p3_var	price_p
75	ef716222bbd97a8bdfcbb831e3575560	2015-04-01	NaN	NaN	NaN	
221	0f5231100b2febab862f8dd8eaab3f43	2015-06-01	NaN	NaN	NaN	
377	2f93639de582fadfbe3e86ce1c8d8f35	2015-06-01	NaN	NaN	NaN	
413	f83c1ab1ca1d1802bb1df4d72820243c	2015-06-01	NaN	NaN	NaN	
461	3076c6d4a060e12a049d1700d9b09cf3	2015-06-01	NaN	NaN	NaN	
...	...	...	...	...	...	
192767	2dc2c9a9f6e6896d9a07d7bcbb9d0ce9	2015-06-01	NaN	NaN	NaN	
192788	e4053a0ad6c55e4665e8e9adb9f75db5	2015-03-01	NaN	NaN	NaN	
192875	1a788ca3bfb16ce443dcf7d75e702b5d	2015-06-01	NaN	NaN	NaN	
192876	1a788ca3bfb16ce443dcf7d75e702b5d	2015-07-01	NaN	NaN	NaN	
192886	d625f9e90d4af9986197444361e99235	2015-05-01	NaN	NaN	NaN	

1359 rows × 8 columns

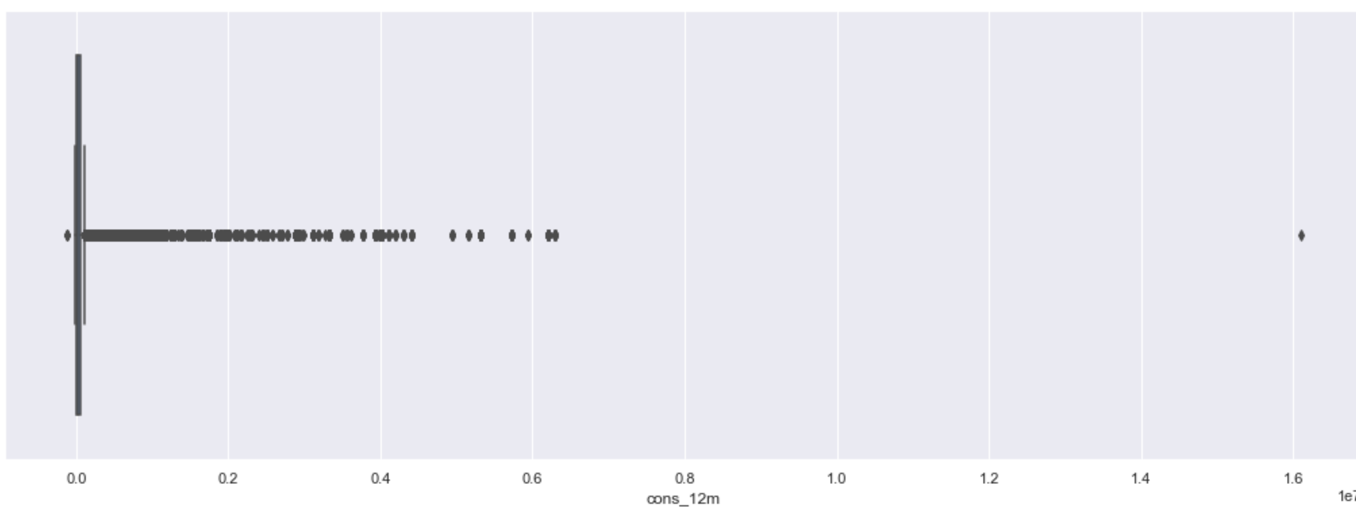
In [44]:
```
history_data.loc[history_data["price_p1_var"].isnull(),"price_p1_var"] = history_data["pri
history_data.loc[history_data["price_p2_var"].isnull(),"price_p2_var"] = history_data["pri
history_data.loc[history_data["price_p3_var"].isnull(),"price_p3_var"] = history_data["pri
history_data.loc[history_data["price_p1_fix"].isnull(),"price_p1_fix"] = history_data["pri
history_data.loc[history_data["price_p2_fix"].isnull(),"price_p2_fix"] = history_data["pri
history_data.loc[history_data["price_p3_fix"].isnull(),"price_p3_fix"] = history_data["pri
```

5.2 Formatting Dates

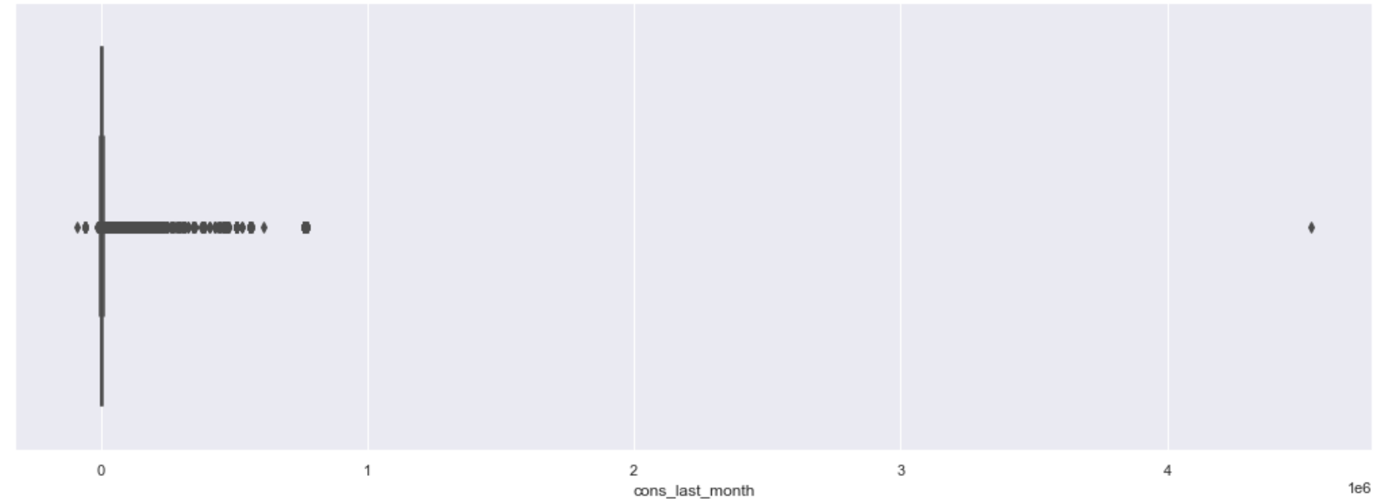
In [45]:
```
# Transform date columns to datetime type
train["date_activ"] = pd.to_datetime(train["date_activ"], format='%Y-%m-%d')
train["date_end"] = pd.to_datetime(train["date_end"], format='%Y-%m-%d')
train["date_modif_prod"] = pd.to_datetime(train["date_modif_prod"], format='%Y-%m-%d')
train["date_renewal"] = pd.to_datetime(train["date_renewal"], format='%Y-%m-%d')
```

In [46]:
```
history_data["price_date"] = pd.to_datetime(history_data["price_date"], format='%Y-%m-%d')
```

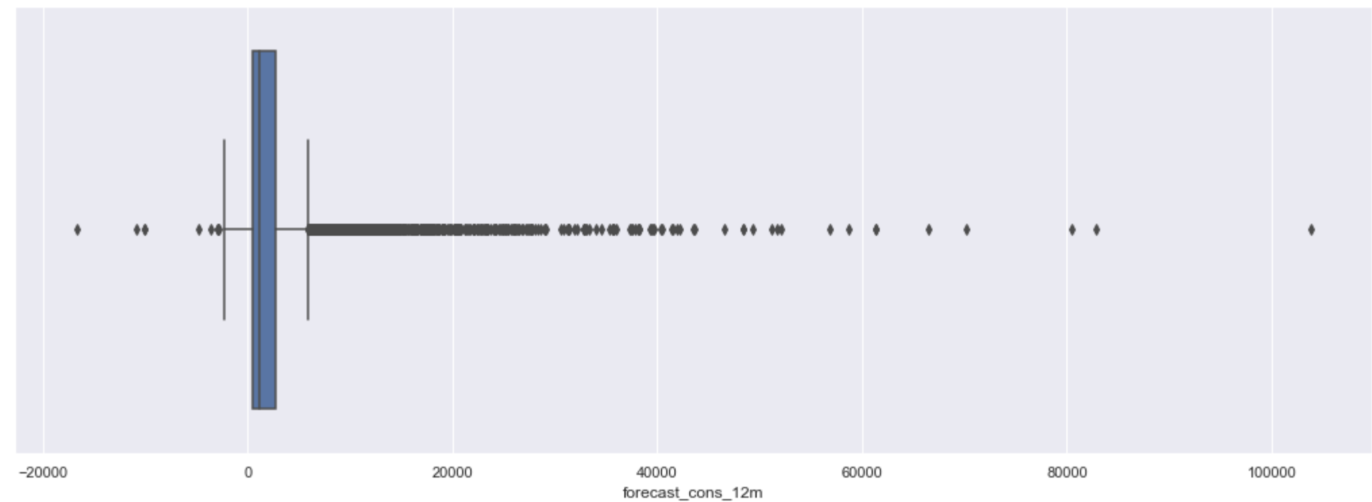
In [47]:
```
 fig, axs = plt.subplots(nrows=7, figsize=(18,50))
# Plot boxplots
sns.boxplot((train["cons_12m"].dropna()), ax=axs[0])
sns.boxplot((train[train["has_gas"]==1]["cons_gas_12m"].dropna()), ax=axs[1])
sns.boxplot((train["cons_last_month"].dropna()), ax=axs[2])
sns.boxplot((train["forecast_cons_12m"].dropna()), ax=axs[3])
#sns.boxplot((train["forecast_cons_year"].dropna()), ax=axs[4])
sns.boxplot((train["forecast_meter_rent_12m"].dropna()), ax=axs[5])
sns.boxplot((train["imp_cons"].dropna()), ax=axs[6])
plt.show()
```



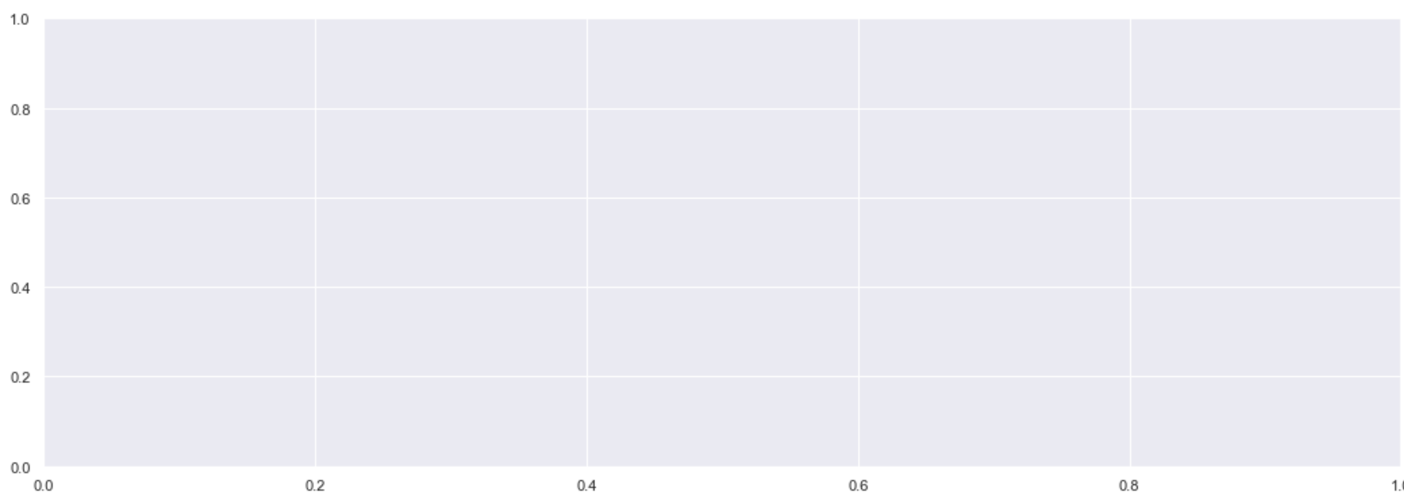
0.0
0.2
0.4
0.6
0.8
1.0
cons_gas_12m



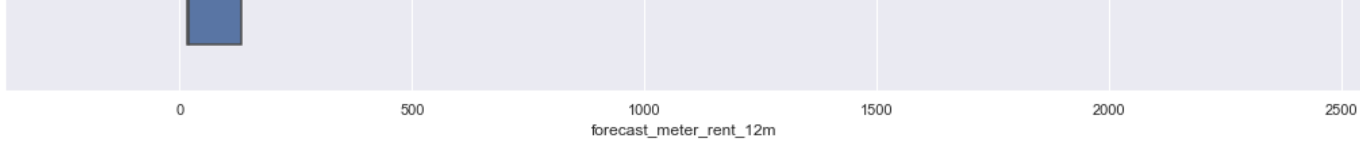
0
1
2
3
4
1e6
cons_last_month

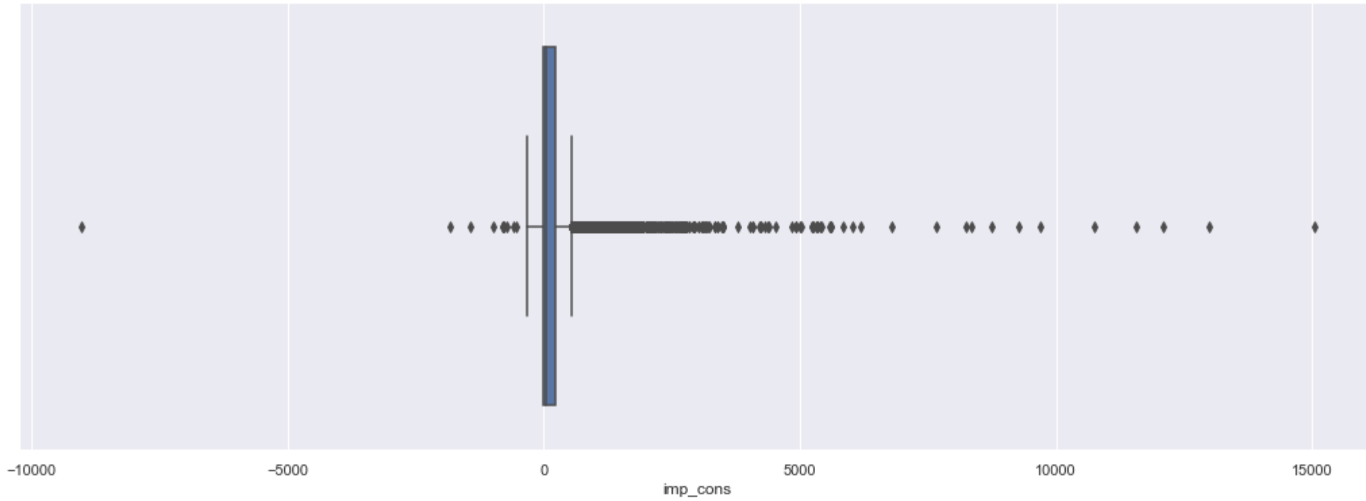


−20000
0
20000
40000
60000
80000
100000
forecast_cons_12m



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0





In [48]:

```
history_data.describe()
```

Out[48]:

	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	43.332175	10.622871	6.409981
std	0.025032	0.049924	0.036298	5.419345	12.841899	7.773595
min	0.000000	0.000000	0.000000	-0.177779	-0.097752	-0.065172
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

6.0 Pickling

In [51]:

```
if not os.path.exists(os.path.join("..", "processed_data")):
 os.makedirs(os.path.join("..", "processed_data"))
```

In [54]:

```
PICKLE_TRAIN_DIR = os.path.join("..", "processed_data", "train_data.pkl")
PICKLE_HISTORY_DIR = os.path.join("..", "processed_data", "history_data.pkl")
```

In [55]:

```
pd.to_pickle(train, PICKLE_TRAIN_DIR)
pd.to_pickle(history_data, PICKLE_HISTORY_DIR)
```