BCG_EDA_Analsis_V2

June 15, 2021

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import datetime
```

[3]:
```
# Set plot style
sns.set(color_codes=True)
# Set maximum number of columns to be displayed
pd.set_option('display.max_columns',100)
```

1 Loading Data

[300]:
```
train_data = pd.read_csv('ml_case_training_data.csv')
churn_data = pd.read_csv('ml_case_training_output.csv')
history_data = pd.read_csv('ml_case_training_hist_data.csv')
```

[251]:
```
train_data.head()
```

[251]:
```
                                 id                     activity_new  \
0  48ada52261e7cf58715202705a0451c9  esoiiifxdlbkcsluxmfuacbdckommixw
1  24011ae4ebbe3035111d65fa7c15bc57                              NaN
2  d29c2c54acc38ff3c0614d0a653813dd                              NaN
3  764c75f661154dac3a6c254cd082ea7d                              NaN
4  bba03439a292a1e166f80264c16191cb                              NaN

   campaign_disc_ele                     channel_sales  cons_12m  \
0                NaN  lmkebamcaaclubfxadlmueccxoimlema    309275
1                NaN  foosdfpfkusacimwkcsosbicdxkicaua         0
2                NaN                               NaN      4660
3                NaN  foosdfpfkusacimwkcsosbicdxkicaua       544
4                NaN  lmkebamcaaclubfxadlmueccxoimlema      1584

   cons_gas_12m  cons_last_month  date_activ    date_end date_first_activ  \
0             0            10025  2012-11-07  2016-11-06              NaN
1         54946                0  2013-06-15  2016-06-15              NaN
```

```
2            0                0  2009-08-21  2016-08-30             NaN
3            0                0  2010-04-16  2016-04-16             NaN
4            0                0  2010-03-30  2016-03-30             NaN

  date_modif_prod date_renewal  forecast_base_bill_ele  \
0      2012-11-07   2015-11-09                     NaN
1             NaN   2015-06-23                     NaN
2      2009-08-21   2015-08-31                     NaN
3      2010-04-16   2015-04-17                     NaN
4      2010-03-30   2015-03-31                     NaN

   forecast_base_bill_year  forecast_bill_12m  forecast_cons  \
0                      NaN                NaN            NaN
1                      NaN                NaN            NaN
2                      NaN                NaN            NaN
3                      NaN                NaN            NaN
4                      NaN                NaN            NaN

   forecast_cons_12m  forecast_cons_year  forecast_discount_energy  \
0           26520.30               10025                       0.0
1               0.00                   0                       0.0
2             189.95                   0                       0.0
3              47.96                   0                       0.0
4             240.04                   0                       0.0

   forecast_meter_rent_12m  forecast_price_energy_p1  \
0                   359.29                  0.095919
1                     1.78                  0.114481
2                    16.27                  0.145711
3                    38.72                  0.165794
4                    19.83                  0.146694

   forecast_price_energy_p2  forecast_price_pow_p1 has_gas  imp_cons  \
0                  0.088347              58.995952       f     831.8
1                  0.098142              40.606701       t       0.0
2                  0.000000              44.311378       f       0.0
3                  0.087899              44.311378       f       0.0
4                  0.000000              44.311378       f       0.0

   margin_gross_pow_ele  margin_net_pow_ele  nb_prod_act  net_margin  \
0                -41.76              -41.76            1     1732.36
1                 25.44               25.44            2      678.99
2                 16.38               16.38            1       18.89
3                 28.60               28.60            1        6.60
4                 30.22               30.22            1       25.46

   num_years_antig                        origin_up  pow_max
```

```
0                 3  ldkssxwpmemidmecebumciepifcamkci  180.000
1                 3  lxidpiddsbxsbosboudacockeimpuepw   43.648
2                 6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.800
3                 6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.856
4                 6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.200
```

Need to resolve the NAN in train_data

[252]:
```python
churn_data.head()
```

[252]:
```
                                 id  churn
0  48ada52261e7cf58715202705a0451c9      0
1  24011ae4ebbe3035111d65fa7c15bc57      1
2  d29c2c54acc38ff3c0614d0a653813dd      0
3  764c75f661154dac3a6c254cd082ea7d      0
4  bba03439a292a1e166f80264c16191cb      0
```

[253]:
```python
history_data.head()
```

[253]:
```
                                 id  price_date  price_p1_var  price_p2_var  \
0  038af19179925da21a25619c5a24b745  2015-01-01      0.151367           0.0
1  038af19179925da21a25619c5a24b745  2015-02-01      0.151367           0.0
2  038af19179925da21a25619c5a24b745  2015-03-01      0.151367           0.0
3  038af19179925da21a25619c5a24b745  2015-04-01      0.149626           0.0
4  038af19179925da21a25619c5a24b745  2015-05-01      0.149626           0.0

   price_p3_var  price_p1_fix  price_p2_fix  price_p3_fix
0           0.0     44.266931           0.0           0.0
1           0.0     44.266931           0.0           0.0
2           0.0     44.266931           0.0           0.0
3           0.0     44.266931           0.0           0.0
4           0.0     44.266931           0.0           0.0
```

Merging Dataset

[301]:
```python
train = train_data.merge(churn_data, on='id')
```

[255]:
```python
train.head()
```

[255]:
```
                                 id                      activity_new  \
0  48ada52261e7cf58715202705a0451c9  esoiiifxdlbkcsluxmfuacbdckommixw
1  24011ae4ebbe3035111d65fa7c15bc57                               NaN
2  d29c2c54acc38ff3c0614d0a653813dd                               NaN
3  764c75f661154dac3a6c254cd082ea7d                               NaN
4  bba03439a292a1e166f80264c16191cb                               NaN

   campaign_disc_ele                     channel_sales  cons_12m  \
0                NaN  lmkebamcaaclubfxadlmueccxoimlema    309275
```

```
1               NaN  foosdfpfkusacimwkcsosbicdxkicaua        0
2               NaN                               NaN     4660
3               NaN  foosdfpfkusacimwkcsosbicdxkicaua      544
4               NaN  lmkebamcaaclubfxadlmueccxoimlema     1584

   cons_gas_12m  cons_last_month  date_activ    date_end date_first_activ  \
0             0            10025  2012-11-07  2016-11-06              NaN
1         54946                0  2013-06-15  2016-06-15              NaN
2             0                0  2009-08-21  2016-08-30              NaN
3             0                0  2010-04-16  2016-04-16              NaN
4             0                0  2010-03-30  2016-03-30              NaN

  date_modif_prod date_renewal  forecast_base_bill_ele  \
0      2012-11-07   2015-11-09                     NaN
1             NaN   2015-06-23                     NaN
2      2009-08-21   2015-08-31                     NaN
3      2010-04-16   2015-04-17                     NaN
4      2010-03-30   2015-03-31                     NaN

   forecast_base_bill_year  forecast_bill_12m  forecast_cons  \
0                      NaN                NaN            NaN
1                      NaN                NaN            NaN
2                      NaN                NaN            NaN
3                      NaN                NaN            NaN
4                      NaN                NaN            NaN

   forecast_cons_12m  forecast_cons_year  forecast_discount_energy  \
0           26520.30               10025                       0.0
1               0.00                   0                       0.0
2             189.95                   0                       0.0
3              47.96                   0                       0.0
4             240.04                   0                       0.0

   forecast_meter_rent_12m  forecast_price_energy_p1  \
0                   359.29                  0.095919
1                     1.78                  0.114481
2                    16.27                  0.145711
3                    38.72                  0.165794
4                    19.83                  0.146694

   forecast_price_energy_p2  forecast_price_pow_p1 has_gas  imp_cons  \
0                  0.088347              58.995952       f     831.8
1                  0.098142              40.606701       t       0.0
2                  0.000000              44.311378       f       0.0
3                  0.087899              44.311378       f       0.0
4                  0.000000              44.311378       f       0.0
```

```
   margin_gross_pow_ele  margin_net_pow_ele  nb_prod_act  net_margin  \
0                -41.76              -41.76            1     1732.36
1                 25.44               25.44            2      678.99
2                 16.38               16.38            1       18.89
3                 28.60               28.60            1        6.60
4                 30.22               30.22            1       25.46

   num_years_antig                         origin_up  pow_max  churn
0                3  ldkssxwpmemidmecebumciepifcamkci  180.000      0
1                3  lxidpiddsbxsbosboudacockeimpuepw   43.648      1
2                6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.800      0
3                6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.856      0
4                6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.200      0
```

2 General statistics of datafrae

Following we want to check the datatype of data, which can suggest us how to do data preprocessing.

Such as convert '2012-02-03' to datetime object

[256]:
```
pd.DataFrame({'Data_type': train.dtypes})
```

[256]:
```
                         Data_type
id                          object
activity_new                object
campaign_disc_ele          float64
channel_sales               object
cons_12m                     int64
cons_gas_12m                 int64
cons_last_month              int64
date_activ                  object
date_end                    object
date_first_activ            object
date_modif_prod             object
date_renewal                object
forecast_base_bill_ele     float64
forecast_base_bill_year    float64
forecast_bill_12m          float64
forecast_cons              float64
forecast_cons_12m          float64
forecast_cons_year           int64
forecast_discount_energy   float64
forecast_meter_rent_12m    float64
forecast_price_energy_p1   float64
forecast_price_energy_p2   float64
forecast_price_pow_p1      float64
has_gas                     object
```

```
imp_cons                  float64
margin_gross_pow_ele      float64
margin_net_pow_ele        float64
nb_prod_act                 int64
net_margin                float64
num_years_antig             int64
origin_up                  object
pow_max                   float64
churn                       int64
```

[23]:
```
pd.DataFrame({'History_Data_type': history_data.dtypes})
```

[23]:
```
             History_Data_type
id                      object
price_date              object
price_p1_var           float64
price_p2_var           float64
price_p3_var           float64
price_p1_fix           float64
price_p2_fix           float64
price_p3_fix           float64
```

2.1 Dataframe statistics

[24]:
```
train.describe()
```

[24]:
```
       campaign_disc_ele      cons_12m  cons_gas_12m  cons_last_month  \
count                0.0  1.609600e+04  1.609600e+04     1.609600e+04
mean                 NaN  1.948044e+05  3.191164e+04     1.946154e+04
std                  NaN  6.795151e+05  1.775885e+05     8.235676e+04
min                  NaN -1.252760e+05 -3.037000e+03    -9.138600e+04
25%                  NaN  5.906250e+03  0.000000e+00     0.000000e+00
50%                  NaN  1.533250e+04  0.000000e+00     9.010000e+02
75%                  NaN  5.022150e+04  0.000000e+00     4.127000e+03
max                  NaN  1.609711e+07  4.188440e+06     4.538720e+06

       forecast_base_bill_ele  forecast_base_bill_year  forecast_bill_12m  \
count             3508.000000              3508.000000        3508.000000
mean               335.843857               335.843857        3837.441866
std                649.406000               649.406000        5425.744327
min               -364.940000              -364.940000       -2503.480000
25%                  0.000000                 0.000000        1158.175000
50%                162.955000               162.955000        2187.230000
75%                396.185000               396.185000        4246.555000
max              12566.080000             12566.080000       81122.630000

       forecast_cons  forecast_cons_12m  forecast_cons_year  \
```

```
count    3508.000000      16096.000000       16096.000000
mean      206.845165       2370.555949        1907.347229
std       455.634288       4035.085664        5257.364759
min         0.000000     -16689.260000      -85627.000000
25%         0.000000        513.230000           0.000000
50%        42.215000       1179.160000         378.000000
75%       228.117500       2692.077500        1994.250000
max      9682.890000     103801.930000      175375.000000

       forecast_discount_energy  forecast_meter_rent_12m  \
count              15970.000000             16096.000000
mean                   0.991547                70.309945
std                    5.160969                79.023251
min                    0.000000              -242.960000
25%                    0.000000                16.230000
50%                    0.000000                19.440000
75%                    0.000000               131.470000
max                   50.000000              2411.690000

       forecast_price_energy_p1  forecast_price_energy_p2  \
count              15970.000000              15970.000000
mean                   0.135901                  0.052951
std                    0.026252                  0.048617
min                    0.000000                  0.000000
25%                    0.115237                  0.000000
50%                    0.142881                  0.086163
75%                    0.146348                  0.098837
max                    0.273963                  0.195975

       forecast_price_pow_p1      imp_cons  margin_gross_pow_ele  \
count           15970.000000  16096.000000          16083.000000
mean               43.533496    196.123447             22.462276
std                 5.212252    494.366979             23.700883
min                -0.122184  -9038.210000           -525.540000
25%                40.606701      0.000000             11.960000
50%                44.311378     44.465000             21.090000
75%                44.311378    218.090000             29.640000
max                59.444710  15042.790000            374.640000

       margin_net_pow_ele   nb_prod_act    net_margin  num_years_antig  \
count        16083.000000  16096.000000  16081.000000     16096.000000
mean            21.460318      1.347788    217.987028         5.030629
std             27.917349      1.459808    366.742030         1.676101
min           -615.660000      1.000000  -4148.990000         1.000000
25%             11.950000      1.000000     51.970000         4.000000
50%             20.970000      1.000000    119.680000         5.000000
75%             29.640000      1.000000    275.810000         6.000000
```

```
max           374.640000     32.000000  24570.650000        16.000000

            pow_max         churn
count  16093.000000  16096.000000
mean      20.604131      0.099093
std       21.772421      0.298796
min        1.000000      0.000000
25%       12.500000      0.000000
50%       13.856000      0.000000
75%       19.800000      0.000000
max      500.000000      1.000000
```

2.1.1 Insight

1. (Missed before) **KEY: the minimum of consumption & forecastas is negative** This could means the Clients also generate power and 'return' thes energy, which is unlikely. **Such result indicate this data is corrupted**
2. Later will find out "Camaign_disc_ele" is completly empty.
3. Highly skewed data when looks at percentiles

[25]:
```python
history_data.describe()
```

[25]:
```
         price_p1_var   price_p2_var   price_p3_var   price_p1_fix  \
count  191643.000000  191643.000000  191643.000000  191643.000000
mean        0.140991       0.054412       0.030712      43.325546
std         0.025117       0.050033       0.036335       5.437952
min         0.000000       0.000000       0.000000      -0.177779
25%         0.125976       0.000000       0.000000      40.728885
50%         0.146033       0.085483       0.000000      44.266930
75%         0.151635       0.101780       0.072558      44.444710
max         0.280700       0.229788       0.114102      59.444710

         price_p2_fix   price_p3_fix
count  191643.000000  191643.000000
mean       10.698201       6.455436
std        12.856046       7.782279
min        -0.097752      -0.065172
25%         0.000000       0.000000
50%         0.000000       0.000000
75%        24.339581      16.226389
max        36.490692      17.458221
```

2.2 Missing Data

[238]:
```
total_null = train.isnull().sum().sort_values(ascending = False)
null_percentage = (train.isnull().sum() / train.isnull().count()).
 ↪sort_values(ascending = False)
missing_data = pd.concat([total_null,␣
 ↪null_percentage],keys=['Total_null','Null_percentage'] ,axis = 1)
missing_data
```

[238]:

	Total_null	Null_percentage
campaign_disc_ele	16096	1.000000
forecast_base_bill_ele	12588	0.782058
date_first_activ	12588	0.782058
forecast_cons	12588	0.782058
forecast_bill_12m	12588	0.782058
forecast_base_bill_year	12588	0.782058
activity_new	9545	0.593004
channel_sales	4218	0.262053
date_modif_prod	157	0.009754
forecast_discount_energy	126	0.007828
forecast_price_energy_p2	126	0.007828
forecast_price_pow_p1	126	0.007828
forecast_price_energy_p1	126	0.007828
origin_up	87	0.005405
date_renewal	40	0.002485
net_margin	15	0.000932
margin_net_pow_ele	13	0.000808
margin_gross_pow_ele	13	0.000808
pow_max	3	0.000186
date_end	2	0.000124
nb_prod_act	0	0.000000
imp_cons	0	0.000000
num_years_antig	0	0.000000
id	0	0.000000
forecast_cons_12m	0	0.000000
has_gas	0	0.000000
forecast_meter_rent_12m	0	0.000000
forecast_cons_year	0	0.000000
date_activ	0	0.000000
cons_last_month	0	0.000000
cons_gas_12m	0	0.000000
cons_12m	0	0.000000
churn	0	0.000000

Note that we might need to remove a few columns since they have more than 75% null percentage

[239]:
```
Discarded_columns = missing_data[missing_data.Null_percentage > 0.75].index
Discarded_columns
```

[239]:
```
Index(['campaign_disc_ele', 'forecast_base_bill_ele', 'date_first_activ',
       'forecast_cons', 'forecast_bill_12m', 'forecast_base_bill_year'],
      dtype='object')
```

[31]:
```
# Null for history data
total_null = history_data.isnull().sum().sort_values(ascending = False)
null_percentage = (history_data.isnull().sum() / history_data.isnull().count()).
 ↪sort_values(ascending = False)
missing_data = pd.concat([total_null,␣
 ↪null_percentage],keys=['Total_null','Null_percentage'] ,axis = 1)
missing_data
```

[31]:

	Total_null	Null_percentage
price_p1_var	1359	0.007041
price_p2_var	1359	0.007041
price_p3_var	1359	0.007041
price_p1_fix	1359	0.007041
price_p2_fix	1359	0.007041
price_p3_fix	1359	0.007041
id	0	0.000000
price_date	0	0.000000

As for history data, the null percentage is small, can easily to be dealt with

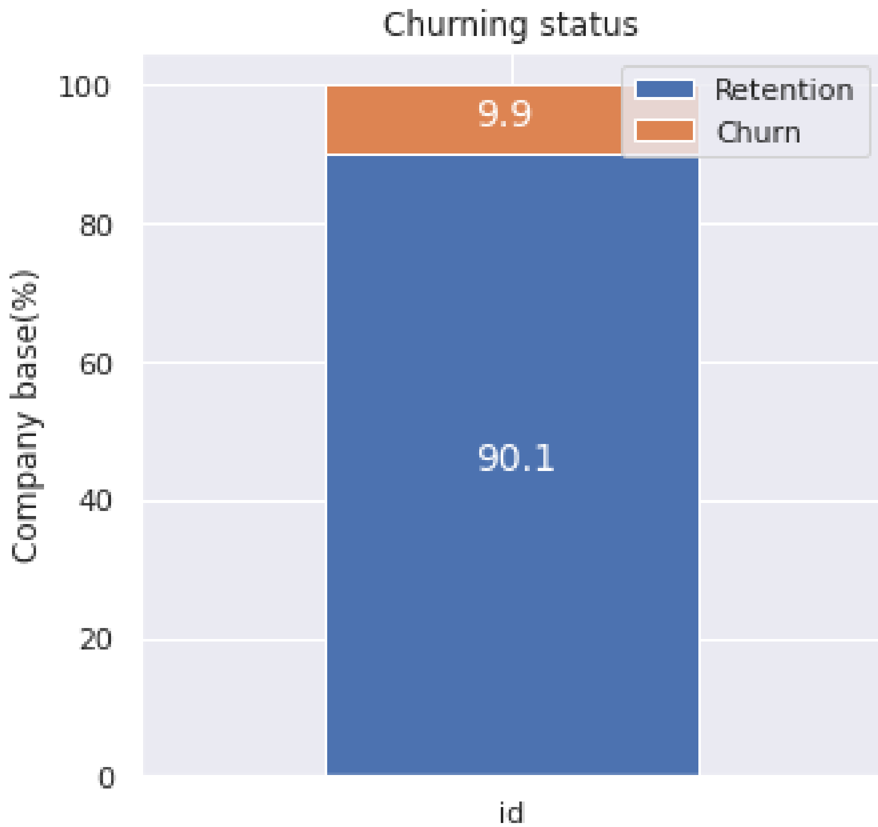
3 Data visualization

[74]:
```
def annotate_stacked_bar(ax, pad=0.99, colour = 'white', textsize=13):
    """
    Add values annotations to the bars
    """
    # Iterate over the plotted rectageL
    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_stacked_bars(df, title_, size_ = (18,10), rot_=0, legend_='upper␣
↪right'):
    """
    Plot stacked bars with annotations
    """
    ax = df.plot( kind = "bar",
                  stacked = True,
                  figsize = size_,
                  rot= rot_,
                  title= title_
                )
    # Annotate bars
    annotate_stacked_bar(ax, textsize=14)
    # Rename legend
    plt.legend(["Retention","Churn"], loc=legend_)
    # Label
    plt.ylabel("Company base(%)")
    plt.show()
```

[75]:
```
churn_total = churn_data.groupby(churn_data['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
```

[78]:
```
plot_stacked_bars(churn_percentage.T,"Churning status",(5,5))
```




```
[ ]: # KEY, df need to transpose, to seperate categories, then stacked = True will␣
     ↪do the effec
     # churn_percentage.plot(kind="bar",stacked=True, rot= 1)
     # churn_percentage.T.plot(kind="bar",stacked=True, rot= 1)
```

We can observe about 10% of total customers have churned(Reasonable ratio)

3.1 SME activities(Previous missed)

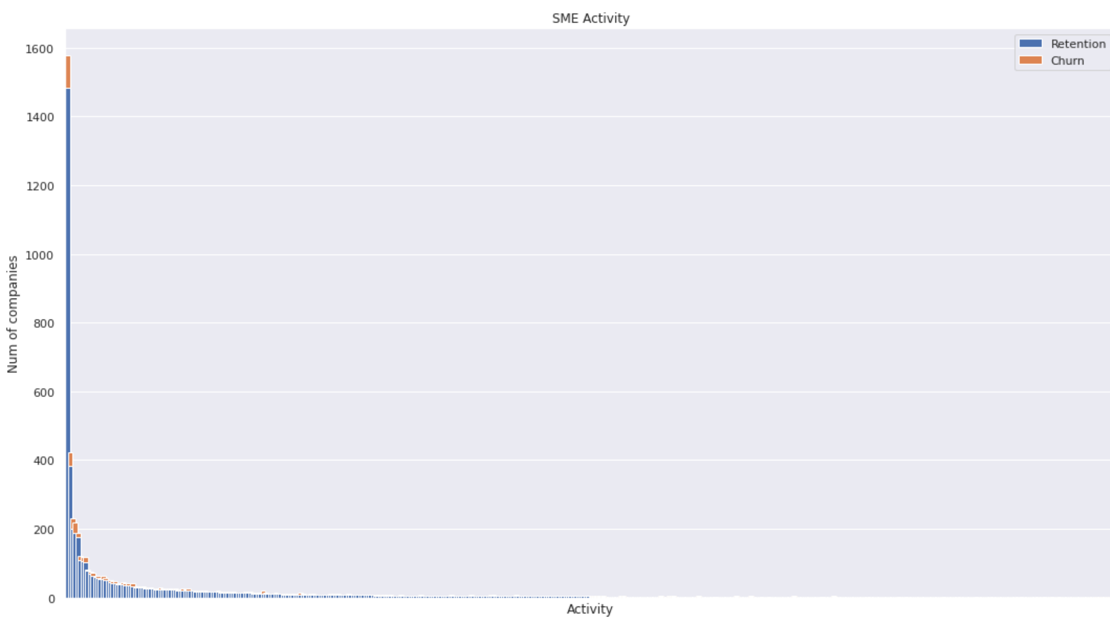
Let's inspect how the SME activities is correlated with churn

```
[84]: activity = train[['id','activity_new','churn']]
```

```
[98]: # Group by ['activity_new' and 'churn'], then count the unique ID, can show how␣
      ↪many customers
      # belong to ceratin category of activities
      activity.groupby([activity['activity_new'],activity['churn']])['id'].count()
      # before unstack, the col is churn and how many corresponding customers
```

```
activity = activity.groupby([activity['activity_new'],activity['churn']])['id'].
↪count().unstack(level=1).sort_values(by=[0],ascending = False)
# after unstack, the col become "Whether churned"
```

[101]:
```
activity.plot(kind = "bar",
             figsize = (18,10),
             width = 2,
             stacked = True,
             title = "SME Activity")
# Labels
plt.ylabel('Num of companies')
plt.xlabel('Activity')
#Legend
plt.legend(['Retention','Churn'], loc="upper right")
# remove label of xtick, since they're encoded
plt.xticks([])
plt.show()
```



We can see **churn is not specifically realted to any SME category in particular**
If we take a further look at the values percenatage-wise

[109]:
```
activity.isnull().sum()
```

[109]:
```
churn
0      8
1    233
dtype: int64
```

[116]:
```
# Count the total number of company in each category, including churn and␣
↪non-churn
activity_total = activity.fillna(0)[0] + activity.fillna(0)[1]
activity_percentage = activity.fillna(0)[1] / activity_total *100
pd.DataFrame({ "Churn Percentage": activity_percentage,
              "Total companies": activity_total
            }).sort_values(by="Total companies", ascending = False)
```

[116]:

activity_new	Churn Percentage	Total companies
apdekpcbwosbxepsfxclislboipuxpop	5.897273	1577.0
kkklcdamwfafdcfwofuscwfwadblfmce	9.004739	422.0
kwuslieomapmswolewpobpplkaooaaew	13.043478	230.0
fmwdwsxillemwbbwelxsampiuwwpcdcb	14.611872	219.0
ckfxocssowaeipxueikxcmaxdmcduxsa	6.878307	189.0
…	…	…
eckbfdkkkfoxpeffpacbikwpeicksulu	0.000000	1.0
eamiapdokbfumefocubefudcowecllla	0.000000	1.0
dxmfpsflslufmxlmwdmbkikffowmfmum	0.000000	1.0
dwdflbsopucwoxdmccmulwiiefiiabel	0.000000	1.0
xwkaesbkfsacseixxksofpddwfkbobki	100.000000	1.0

[419 rows x 2 columns]

[117]:
```
pd.DataFrame({ "Churn Percentage": activity_percentage,
              "Total companies": activity_total
            }).sort_values(by="Churn Percentage", ascending = False)
```

[117]:

activity_new	Churn Percentage	Total companies
xwkaesbkfsacseixxksofpddwfkbobki	100.0	1.0
wkwdccuiboaeaalcaawlwmldiwmpewma	100.0	1.0
ikiucmkuisupefxcxfxxulkpwssppfuo	100.0	1.0
opoiuuwdmxdssidluooopfswlkkkcsxf	100.0	1.0
pfcocskbxlmofswiflsbcefcpufbopuo	100.0	2.0
…	…	…
pudsxpkefiudxxfcumemocbpuklxiufa	0.0	8.0
ambaaxsxxwfuspsuabupewfpbbksmcoo	0.0	2.0
sbolemmfddlosupuwbcawusmbwmdmdfw	0.0	2.0
pkakblpskuwxskooaelouomofdulxpdw	0.0	2.0
lplipswbpdloacuapedmspmesfkbeocu	0.0	3.0

[419 rows x 2 columns]

Note: In some categories, the churn percentage can rise to 100%, but it also due to the fact that only a few companies belong to that activity

On the other hand, for those croweded categories, the churn percentage tends to be low **Impact**

from SME activity on predictive model

```
It is likely that the predictive model will struggle predicting the SME activity due to the la
```

3.2 Sales channel(Previous missed)

The sales channel may be important as well, it's not the same if the sales were through email or throgh telephone

[121]:
```
channel = train[['id','channel_sales','churn']]
```

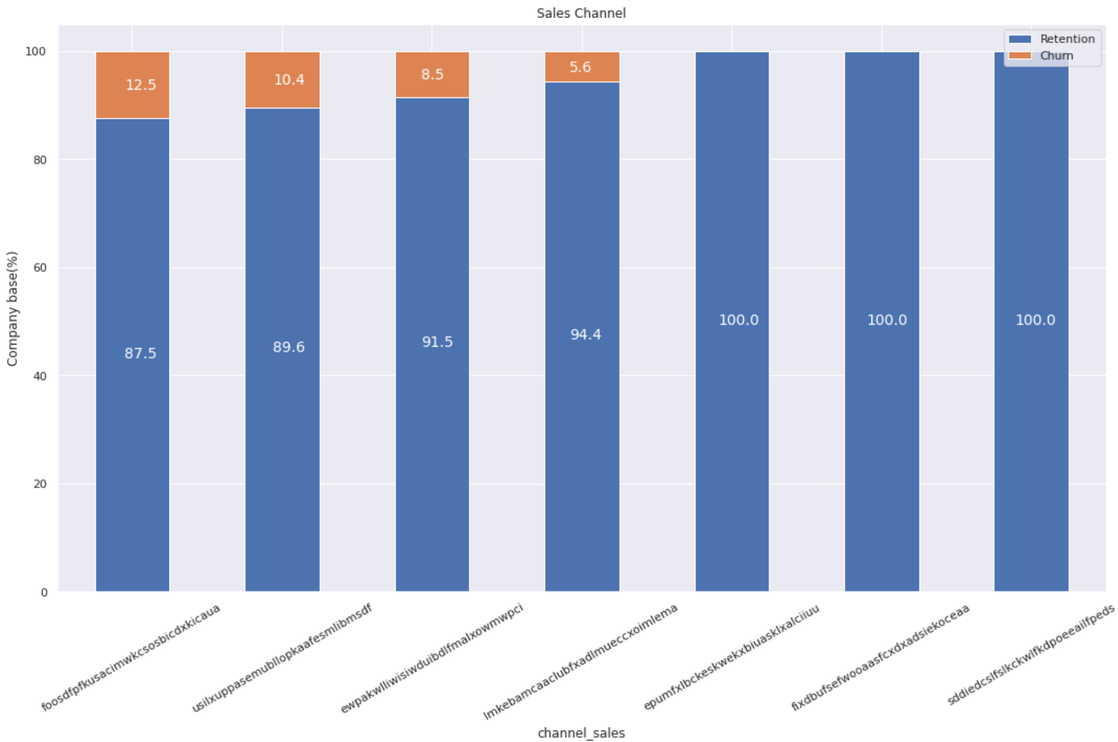
[126]:
```
channel = channel.groupby(['channel_sales','churn'])['id'].count().
↪unstack(level = 1).fillna(0)
```

[127]:
```
channel
```

[127]:
```
churn                                 0      1
channel_sales
epumfxlbckeskwekxbiuasklxalciiuu     4.0    0.0
ewpakwlliwisiwduibdlfmalxowmwpci   884.0   82.0
fixdbufsefwooaasfcxdxadsiekoceaa     2.0    0.0
foosdfpfkusacimwkcsosbicdxkicaua  6455.0  922.0
lmkebamcaaclubfxadlmueccxoimlema  1957.0  116.0
sddiedcslfslkckwlfkdpoeeailfpeds    12.0    0.0
usilxuppasemubllopkaafesmlibmsdf  1294.0  150.0
```

[131]:
```
# Perform seperate division to churn and non-churn,
channel_churn = (channel.div(channel.sum(axis = 1), axis = 0) *100).
↪sort_values(by= [1], ascending = False)
```

[133]:
```
plot_stacked_bars(channel_churn, "Sales Channel", rot_ = 30)
```



Take a look at percentage-wise
Col: churn percentage, total companies

[145]:
```
churn_percentage = channel[1] / channel.sum(axis=1) *100
total_companies = channel.sum(axis=1)
pd.DataFrame({ 'Churn Percentage': churn_percentage,
               'Total Companies': total_companies
            }).sort_values(by='Churn Percentage',ascending = False)
```

[145]:

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

[143]:
```
total_companies
```

[143]:
```
channel_sales
epumfxlbckeskwekxbiuasklxalciiuu      4.0
ewpakwlliwisiwduibdlfmalxowmwpci    966.0
```

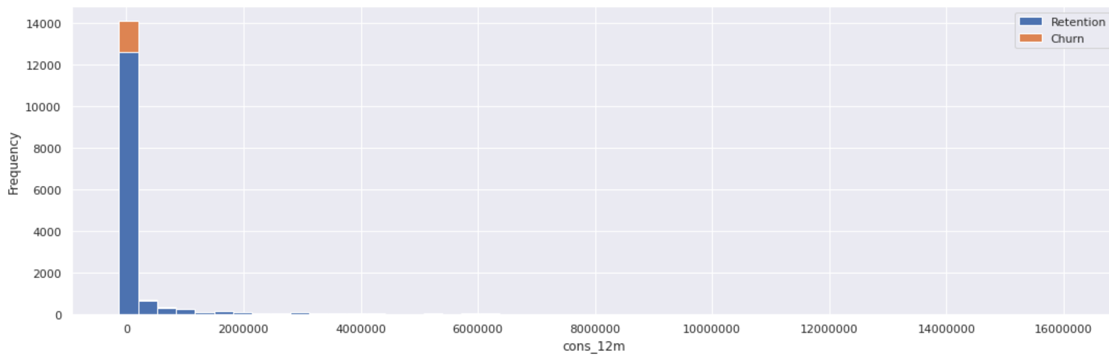
```
fixdbufsefwooaasfcxdxadsiekoceaa       2.0
foosdfpfkusacimwkcsosbicdxkicaua    7377.0
lmkebamcaaclubfxadlmueccxoimlema    2073.0
sddiedcslfslkckwlfkdpoeeailfpeds      12.0
usilxuppasemubllopkaafesmlibmsdf    1444.0
dtype: float64
```

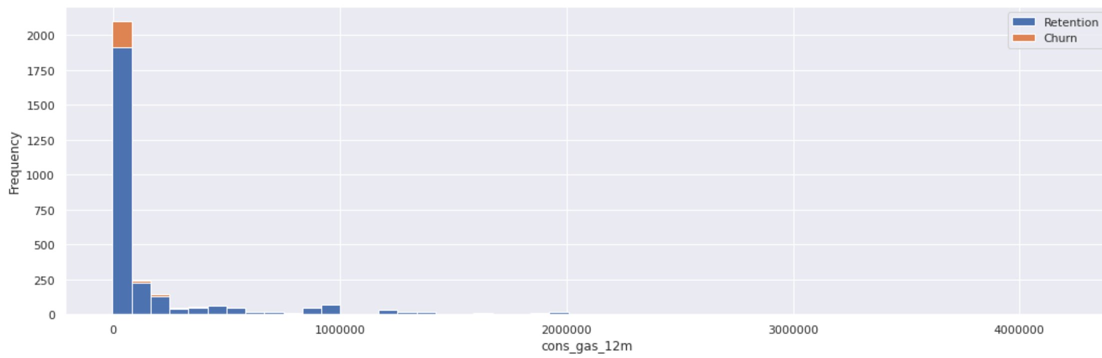
3.3 Consumption

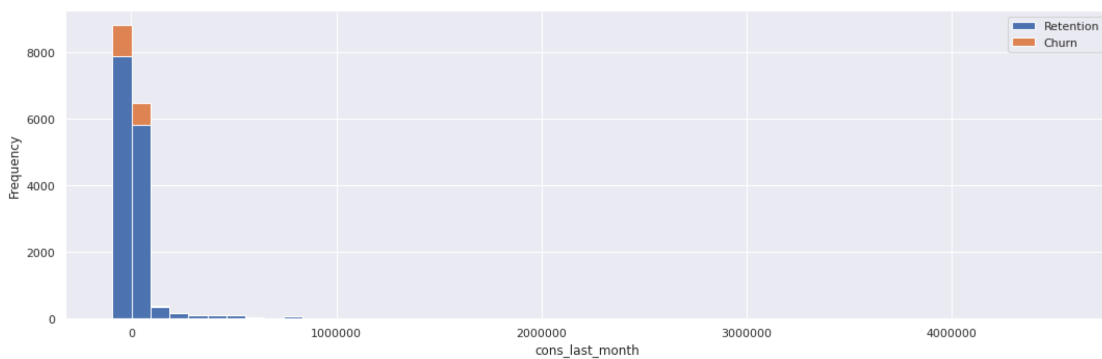
[147]:
```
consumption =␣
↪train[['id','cons_12m','cons_gas_12m','cons_last_month','imp_cons','has_gas','churn']]
```

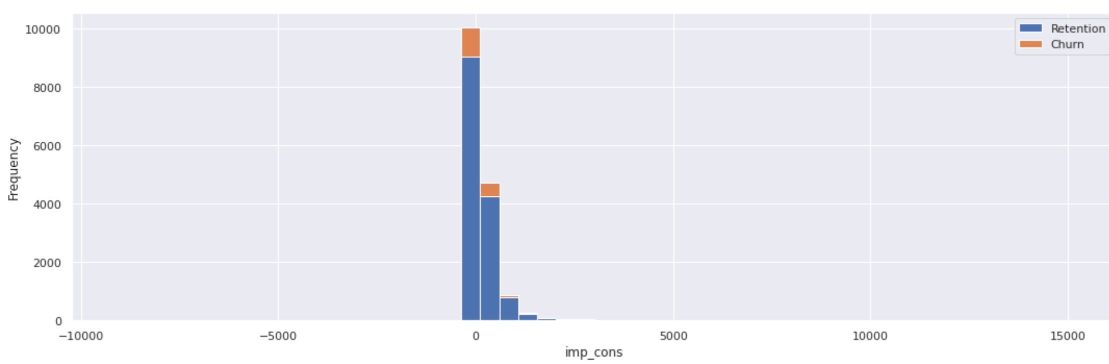
[152]:
```
def plot_distribution(df, column, ax, bins_ = 50):
    """
    Plot variable distribution in a stacked histogram of churned or retained␣
 ↪comopany
    """
    tmp = pd.DataFrame({"Retention": df[df['churn'] == 0][column],
                        "Churn": df[df['churn'] == 1][column]
                        })
    # Plot the histogram
    tmp.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')
```

[155]:
```
fig, axs = plt.subplots(nrows = 4, figsize=(18,25))
# Plot histogram
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])
# plot_distribution(consumption,"cons_12m", axs[0])
```







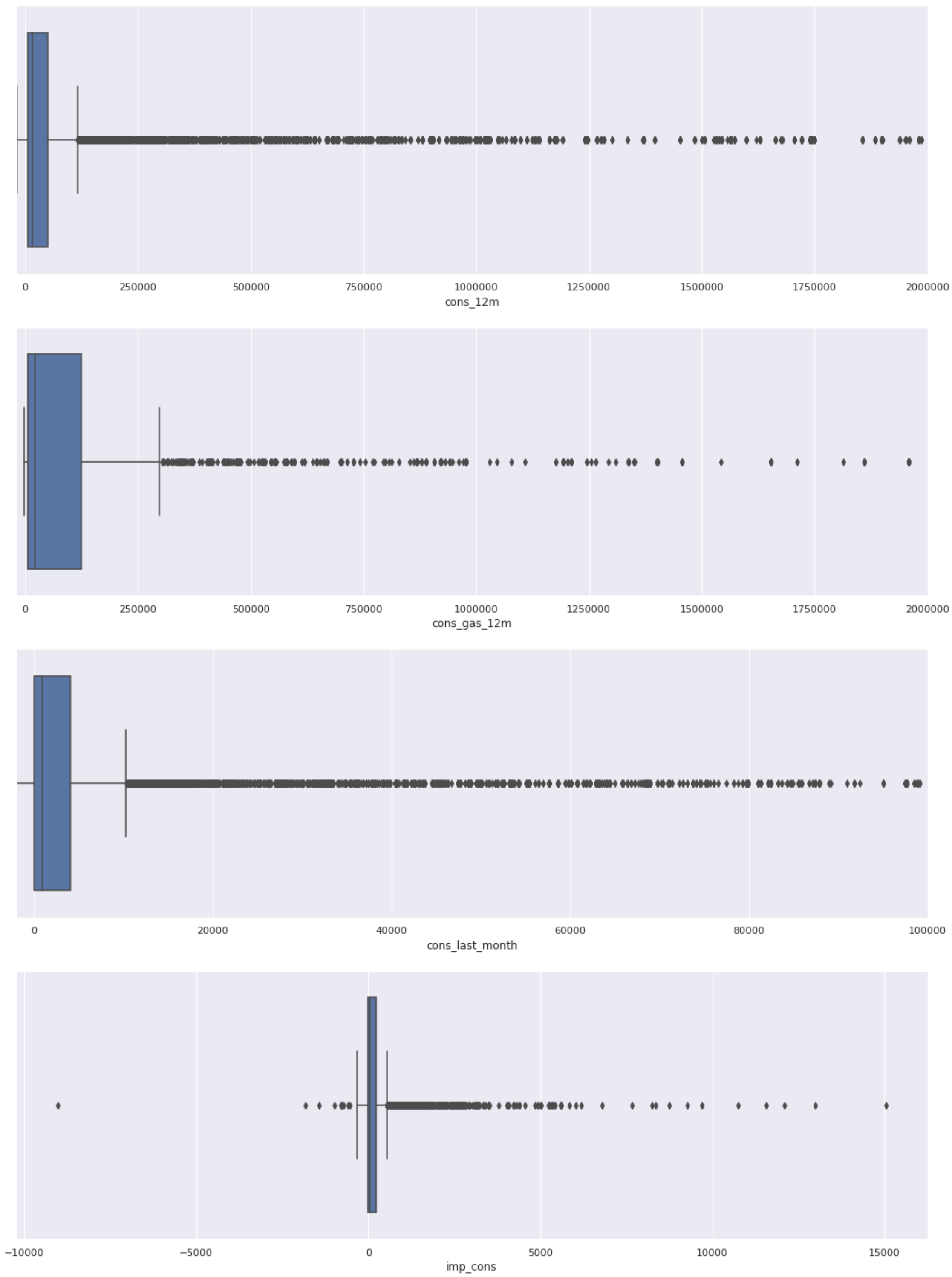


We can cleary see that the consumption is highly **skewed to the right**,presenting a very long tail towards the high values of the distribution

We can use box-plot to further visualize outliers.It also can tell us about our outliers and what their values are. It can also tell us if our data is symmetrical, how tightlyour data is grouped, and if and how our data is skewed.

[161]:
```
fig, axs = plt.subplots(nrows = 4, figsize=(18,25))
# plot box-plot
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])
for ax in axs:
    ax.ticklabel_format(style='plain',axis='x')
# Set x-axis limit
axs[0].set_xlim(-20000,2000000)
axs[1].set_xlim(-20000,2000000)
axs[2].set_xlim(-2000,100000)
plt.show()
```

```
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

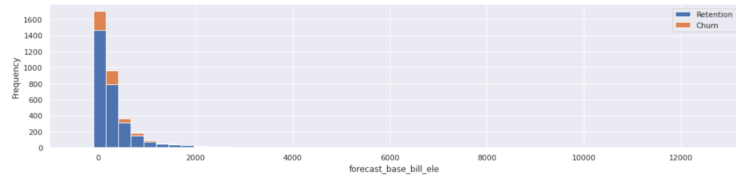


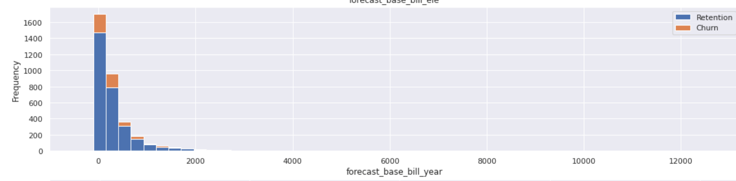
It is very clear now that we have a **highly skewed distribution, and numerous outliers**. We will deal with the skewness and outliers in the next exercise (Data cleaning)

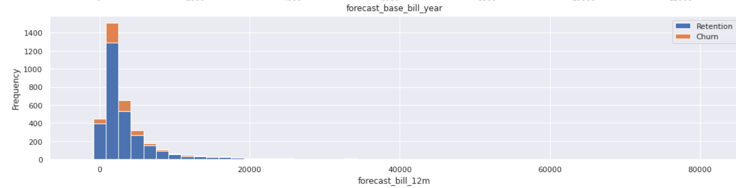
3.4 Forecast

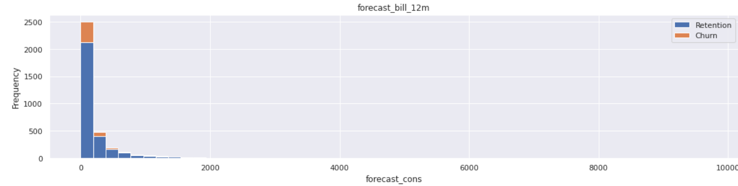
[163]:
```
forecast_col =␣
↪["id","forecast_base_bill_ele","forecast_base_bill_year","forecast_bill_12m"
               ␣
↪,"forecast_cons","forecast_cons_12m","forecast_cons_year","forecast_discount_energy",
               ␣
↪"forecast_meter_rent_12m","forecast_price_energy_p1","forecast_price_energy_p2","forecast_p
                ,"churn"
                ]

forecast = train[forecast_col]
```

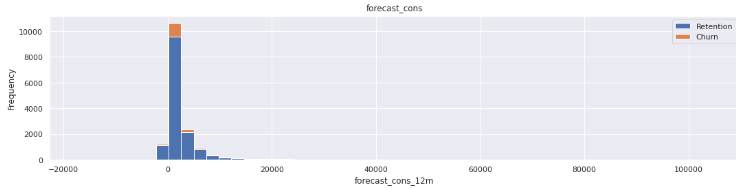
[167]:
```
fig ,axs = plt.subplots(nrows = 11, figsize=(18,50))
# plot histogram
for i in range(1,len(forecast_col)-1):
    plot_distribution(train, forecast_col[i],axs[i-1])
```

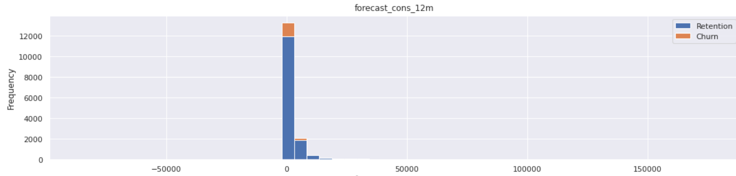

Retention
Churn
Frequency
0 200 400 600 800 1000 1200 1400 1600
0 2000 4000 6000 8000 10000 12000
forecast_base_bill_ele

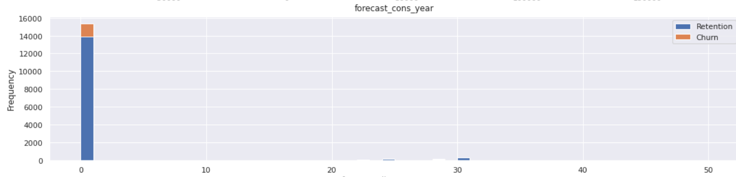

Retention
Churn
Frequency
0 200 400 600 800 1000 1200 1400 1600
0 2000 4000 6000 8000 10000 12000
forecast_base_bill_year

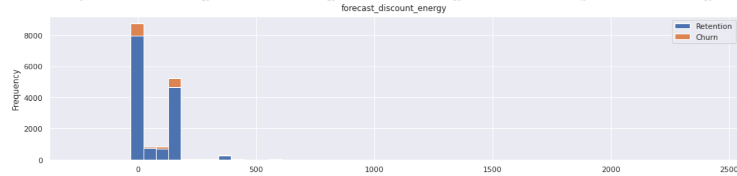

Retention
Churn
Frequency
0 200 400 600 800 1000 1200 1400
0 20000 40000 60000 80000
forecast_bill_12m

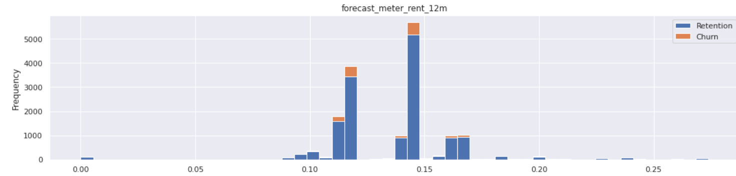

Retention
Churn
Frequency
0 500 1000 1500 2000 2500
0 2000 4000 6000 8000 10000
forecast_cons

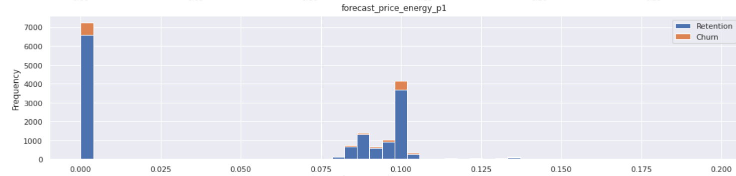

Retention
Churn
Frequency
0 2000 4000 6000 8000 10000
−20000 0 20000 40000 60000 80000 100000
forecast_cons_12m

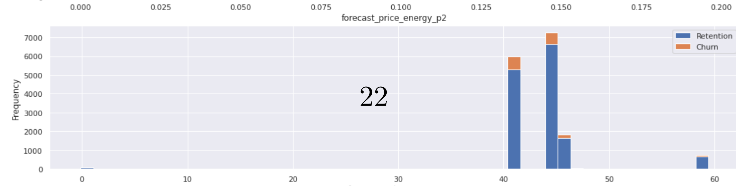

Retention
Churn
Frequency
0 2000 4000 6000 8000 10000 12000
−50000 0 50000 100000 150000
forecast_cons_year


Retention
Churn
Frequency
0 2000 4000 6000 8000 10000 12000 14000 16000
0 10 20 30 40 50
forecast_discount_energy


Retention
Churn
Frequency
0 2000 4000 6000 8000
0 500 1000 1500 2000 2500
forecast_meter_rent_12m


Retention
Churn
Frequency
0 1000 2000 3000 4000 5000
0.00 0.05 0.10 0.15 0.20 0.25
forecast_price_energy_p1


Retention
Churn
Frequency
0 1000 2000 3000 4000 5000 6000 7000
0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175 0.200
forecast_price_energy_p2

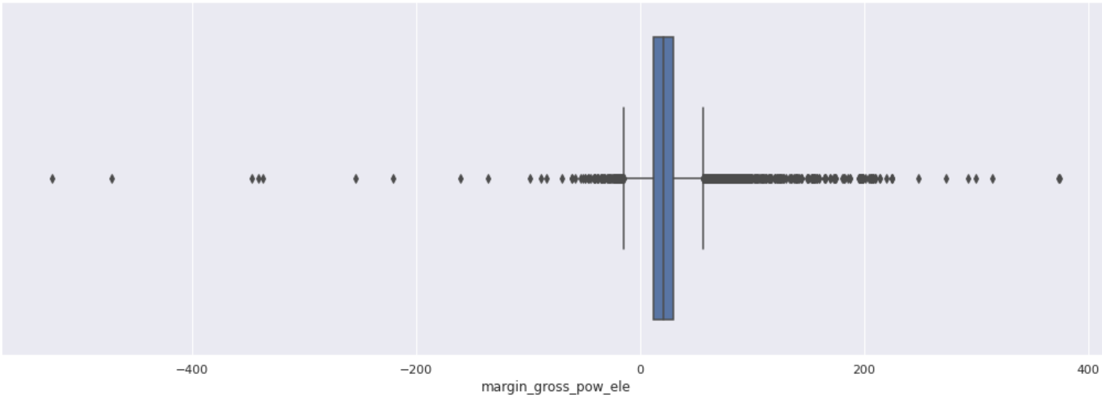

Retention
Churn
Frequency
0 1000 2000 3000 4000 5000 6000 7000
0 10 20 30 40 50 60
forecast_price_pow_p1

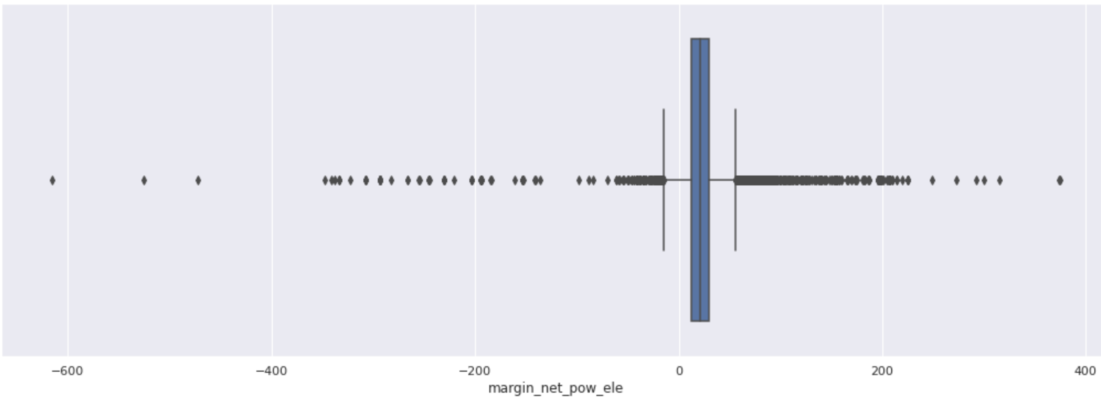
Similarly to the consumption plots, we can observe that a **lot of the variables are highly skewed to the right, creating a very long tail on the highervalues.**
We will make some transformations to correct for this skewness

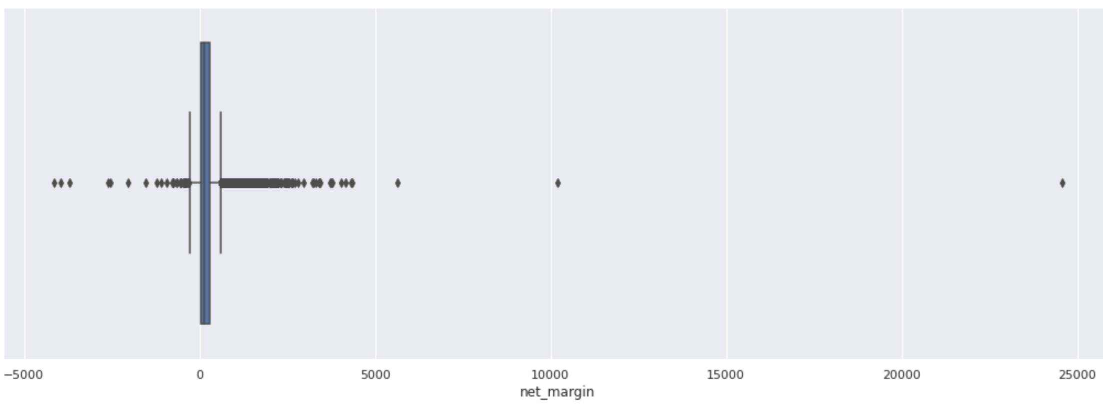
3.5 Margins

[168]:
```
fig ,axs = plt.subplots(nrows = 3, figsize=(18,20))
# plot boxplot
sns.boxplot(train['margin_gross_pow_ele'], ax = axs[0])
sns.boxplot(train['margin_net_pow_ele'], ax = axs[1])
sns.boxplot(train['net_margin'], ax = axs[2])

# Remove scientific notation
plt.show()
```

```
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
/home/brian/miniconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```





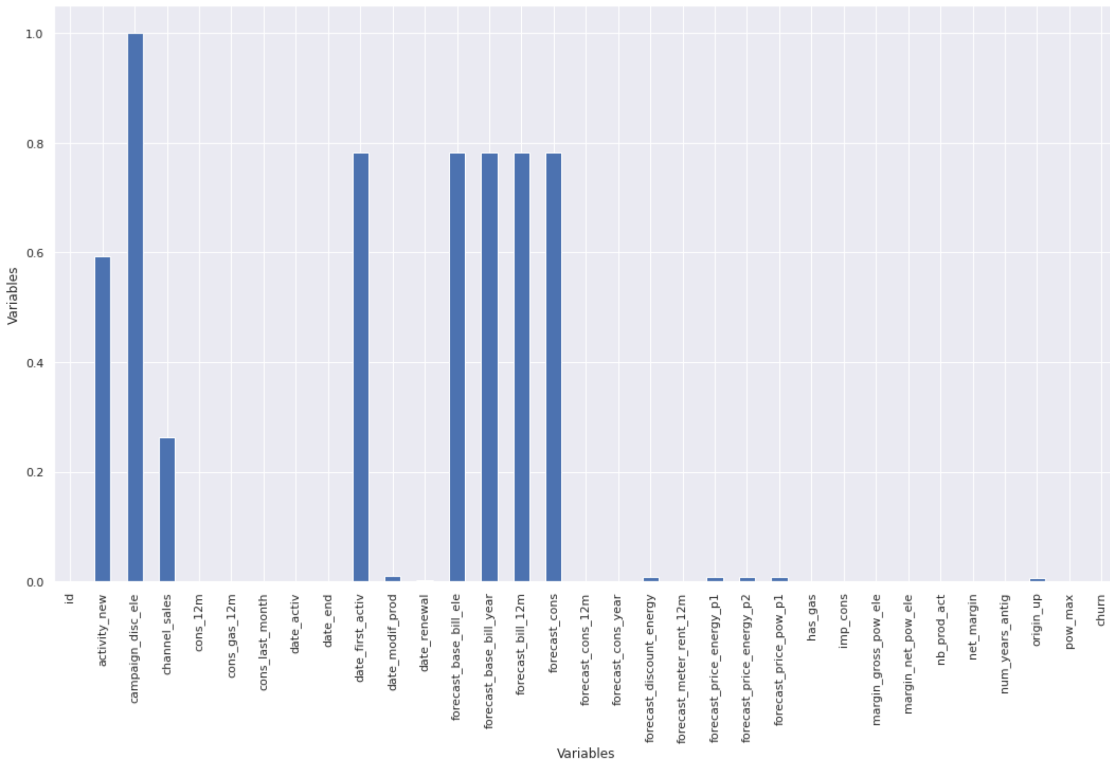


We can observe a few outliers in here as well.

4 Data Cleaning

[172]:
```
(train.isnull().sum() / train.isnull().count()).plot(kind = 'bar', figsize␣
↪=(18,10))

plt.xlabel('Variables')
plt.ylabel('Variables')
```

```
plt.show()
```



[302]:
```
train = train.drop(columns = Discarded_columns)
```

[241]:
```
## Delete Duplicates
train = train.T.drop_duplicates().T
```

[303]:
```
train.shape
```

[303]: (16096, 27)

[304]:
```
num_col = train._get_numeric_data().columns.tolist()
cat_col = set(train.columns) - set(num_col)
```

[313]:
```
# num_col
```

4.1 Deal with missing date in train_data

[311]:
```
for col in cat_col:
    print(col)
    # Fill the categorical column with Mode
    train[col]
```

```
    mode = train[col].mode()[0]
    train[col].fillna(mode, inplace=True)
```

```
activity_new
date_renewal
date_modif_prod
origin_up
has_gas
id
date_end
date_activ
channel_sales
```

[312]:
```
train[cat_col].isnull().sum()
```

[312]:
```
activity_new       0
date_renewal       0
date_modif_prod    0
origin_up          0
has_gas            0
id                 0
date_end           0
date_activ         0
channel_sales      0
dtype: int64
```

[315]:
```
for col in num_col:
    train[col].fillna(train[col].median(), inplace = True)
```

[316]:
```
train[num_col].isnull().sum()
```

[316]:
```
cons_12m                    0
cons_gas_12m                0
cons_last_month             0
forecast_cons_12m           0
forecast_cons_year          0
forecast_discount_energy    0
forecast_meter_rent_12m     0
forecast_price_energy_p1    0
forecast_price_energy_p2    0
forecast_price_pow_p1       0
imp_cons                    0
margin_gross_pow_ele        0
margin_net_pow_ele          0
nb_prod_act                 0
net_margin                  0
num_years_antig             0
```

```
pow_max                    0
churn                      0
dtype: int64
```

4.2 Deal with missing data in history data

[317]:
```python
h_num_col = history_data._get_numeric_data().columns
```

[319]:
```python
for col in h_num_col:
    history_data[col].fillna(history_data[col].median(), inplace= True)
```

[320]:
```python
history_data.isnull().sum()
```

[320]:
```
id              0
price_date      0
price_p1_var    0
price_p2_var    0
price_p3_var    0
price_p1_fix    0
price_p2_fix    0
price_p3_fix    0
dtype: int64
```

[323]:
```python
history_data.describe()
```

[323]:
```
        price_p1_var   price_p2_var   price_p3_var   price_p1_fix  \
count  193002.000000  193002.000000  193002.000000  193002.000000
mean        0.141027       0.054630       0.030496      43.332175
std         0.025032       0.049924       0.036298       5.419345
min         0.000000       0.000000       0.000000      -0.177779
25%         0.125976       0.000000       0.000000      40.728885
50%         0.146033       0.085483       0.000000      44.266930
75%         0.151635       0.101673       0.072558      44.444710
max         0.280700       0.229788       0.114102      59.444710

        price_p2_fix   price_p3_fix
count  193002.000000  193002.000000
mean       10.622871       6.409981
std        12.841899       7.773595
min        -0.097752      -0.065172
25%         0.000000       0.000000
50%         0.000000       0.000000
75%        24.339581      16.226389
max        36.490692      17.458221
```

We can see that there are negative values for
price_p1_fix, price_p2_fix and price_p3_fix.

Further exploring on those we can see there are only about 10 entries which are **negative. This is more likely to be due to corrupted data rather thana "price discount".**
We will replace the negative values with the median

[335]:
```
deal_col = ["price_p1_fix", "price_p2_fix", "price_p3_fix"]

for col in deal_col:
#     print(history_data[history_data[col] < 0][col])
    history_data.loc[history_data[col] < 0, col] = history_data[col].median()
```

[336]:
```
# Check
for col in deal_col:
    print(history_data[history_data[col] < 0][col])
```

```
Series([], Name: price_p1_fix, dtype: float64)
Series([], Name: price_p2_fix, dtype: float64)
Series([], Name: price_p3_fix, dtype: float64)
```

[337]:
```
history_data.describe()
```

[337]:

	price_p1_var	price_p2_var	price_p3_var	price_p1_fix
count	193002.000000	193002.000000	193002.000000	193002.000000
mean	0.141027	0.054630	0.030496	43.334477
std	0.025032	0.049924	0.036298	5.410297
min	0.000000	0.000000	0.000000	0.000000
25%	0.125976	0.000000	0.000000	40.728885
50%	0.146033	0.085483	0.000000	44.266930
75%	0.151635	0.101673	0.072558	44.444710
max	0.280700	0.229788	0.114102	59.444710

	price_p2_fix	price_p3_fix
count	193002.000000	193002.000000
mean	10.622875	6.409984
std	12.841895	7.773592
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	24.339581	16.226389
max	36.490692	17.458221

[ ]: