Exploratory Data Analysis

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

1 Import packages

In [2]:
```
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)
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

---

2 Loading data with Pandas

In [4]:
```
client_df = pd.read_csv(r'C:\Users\hp\Desktop\Portfolio Projects\Forage BCG\Data\client_data.csv')
price_df = pd.read_csv(r'C:\Users\hp\Desktop\Portfolio Projects\Forage BCG\Data\price_data.csv')
```

Let's have a look at the data

In [5]:
```
client_df.head(10)
```

Out[5]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_prod	date_renewal
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2016-06-15	2015-11-01	2015-06-23
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2016-08-30	2009-08-21	2015-08-31
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16	2016-04-16	2010-04-16	2015-04-17
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	2010-03-30	2016-03-30	2010-03-30	2015-03-31
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526	2010-01-13	2016-03-07	2010-01-13	2015-03-09
5	1aa498825382410b098937d65c4ec26d	usilxuppasemubllopkaafesmlibmsdf	8302	0	1998	2011-12-09	2016-12-09	2015-11-01	2015-12-10
6	7ab4bf4878d8f7661dfc20e9b8e18011	foosdfpfkusacimwkcsosbicdxkicaua	45097	0	0	2011-12-02	2016-12-02	2011-12-02	2015-12-03
7	01495c955be7ec5e7f3203406785aae0	foosdfpfkusacimwkcsosbicdxkicaua	29552	0	1260	2010-04-21	2016-04-21	2010-04-21	2015-04-22
8	f53a254b1115634330c12c7fdbf7958a	usilxuppasemubllopkaafesmlibmsdf	2962	0	0	2011-09-23	2016-09-23	2011-09-23	2015-09-25
9	10c1b2f97a2d2a6f10299dc213d1a370	lmkebamcaaclubfxadlmueccxoimlema	26064	0	2188	2010-05-04	2016-05-04	2015-04-29	2015-05-05

10 rows × 26 columns

In [6]:
```
price_df.head(15)
```

Out[6]:

	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	0.000000	0.000000	44.266931	0.000000	0.000000
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.000000	0.000000	44.266931	0.000000	0.000000
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.000000	0.000000	44.266931	0.000000	0.000000
3	038af19179925da21a25619c5a24b745	2015-04-01	0.149626	0.000000	0.000000	44.266931	0.000000	0.000000
4	038af19179925da21a25619c5a24b745	2015-05-01	0.149626	0.000000	0.000000	44.266931	0.000000	0.000000
5	038af19179925da21a25619c5a24b745	2015-06-01	0.149626	0.000000	0.000000	44.266930	0.000000	0.000000
6	038af19179925da21a25619c5a24b745	2015-07-01	0.150321	0.000000	0.000000	44.444710	0.000000	0.000000
7	038af19179925da21a25619c5a24b745	2015-08-01	0.145859	0.000000	0.000000	44.444710	0.000000	0.000000
8	038af19179925da21a25619c5a24b745	2015-09-01	0.145859	0.000000	0.000000	44.444710	0.000000	0.000000
9	038af19179925da21a25619c5a24b745	2015-10-01	0.145859	0.000000	0.000000	44.444710	0.000000	0.000000
10	038af19179925da21a25619c5a24b745	2015-11-01	0.145859	0.000000	0.000000	44.444710	0.000000	0.000000
11	038af19179925da21a25619c5a24b745	2015-12-01	0.145859	0.000000	0.000000	44.444710	0.000000	0.000000
12	31f2ce549924679a3cbb2d128ae9ea43	2015-01-01	0.125976	0.103395	0.071536	40.565969	24.339581	16.226389
13	31f2ce549924679a3cbb2d128ae9ea43	2015-02-01	0.125976	0.103395	0.071536	40.565969	24.339581	16.226389
14	31f2ce549924679a3cbb2d128ae9ea43	2015-03-01	0.125976	0.103395	0.071536	40.565969	24.339581	16.226389

3 Descriptive statistics of data

3.1 Shape, Data Types & Size

Let's have a look at the shape, data types & size of data

In [7]:
```
print("Client Data Information: \n")
client_df.info(verbose = False)
print("\n---------------------------------------\n")
print("Price Data Information: \n")
price_df.info(verbose = False)
```

```
Client Data Information: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Columns: 26 entries, id to churn
dtypes: float64(11), int64(7), object(8)
memory usage: 2.9+ MB

---------------------------------------

Price Data Information: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Columns: 8 entries, id to price_mid_peak_fix
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

Inferences:

We can observe from above that there are multiple date columns in data but in dataframe information datetime dtype is not present, that means dates are stored as string.

In [8]:
```
#lets change the datatype of date columns from string into datetime
for c in client_df.columns:
    if "date" in c:
        client_df[c] = pd.to_datetime(client_df[c],format='%Y-%m-%d')

for c in price_df.columns:
    if "date" in c:
        price_df[c] = pd.to_datetime(price_df[c],format='%Y-%m-%d')
```

In [9]:
```
print("Client Data Information: \n")
client_df.info(verbose = False)
print("\n---------------------------------------\n")
print("Price Data Information: \n")
price_df.info(verbose = False)
```

```
Client Data Information: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Columns: 26 entries, id to churn
dtypes: datetime64[ns](4), float64(11), int64(7), object(4)
memory usage: 2.9+ MB

---------------------------------------

Price Data Information: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Columns: 8 entries, id to price_mid_peak_fix
dtypes: datetime64[ns](1), float64(6), object(1)
memory usage: 11.8+ MB
```

Lets look at the columns by data types

In [10]:
```
cd_cols_num = client_df.select_dtypes(include='number').columns
cd_cols_datetime = client_df.select_dtypes(include=['datetime64']).columns
cd_cols_object = client_df.select_dtypes(include=['object']).columns

pd_cols_num = price_df.select_dtypes(include='number').columns
pd_cols_datetime = price_df.select_dtypes(include=['datetime64']).columns
pd_cols_object = price_df.select_dtypes(include=['object']).columns


print("Client Data columns by datatypes: \n\nNumeric: \n{0},\n-----\nDatetime: \n{1}, \n-----\nString: \n{2}".format(cd_cols_num,cd_cols_da
print("\n---------------------------------------\n")
print("Price Data columns by datatypes: \n\nNumeric: \n{0}, \n-----\nDatetime: \n{1}, \n-----\nString: \n{2}".format(pd_cols_num,cd_cols_da
```

```
Client Data columns by datatypes: 

Numeric: 
Index(['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_price_energy_peak', 'forecast_price_pow_off_peak', 'imp_cons',
       'margin_gross_pow_ele', 'margin_net_pow_ele', 'nb_prod_act',
       'net_margin', 'num_years_antig', 'pow_max', 'churn'],
      dtype='object'),
-----
Datetime: 
Index(['date_activ', 'date_end', 'date_modif_prod', 'date_renewal'], dtype='object'), 
-----
String: 
Index(['id', 'channel_sales', 'has_gas', 'origin_up'], dtype='object')

---------------------------------------

Price Data columns by datatypes: 

Numeric: 
Index(['price_off_peak_var', 'price_peak_var', 'price_mid_peak_var',
       'price_off_peak_fix', 'price_peak_fix', 'price_mid_peak_fix'],
      dtype='object'), 
-----
Datetime: 
Index(['date_activ', 'date_end', 'date_modif_prod', 'date_renewal'], dtype='object'), 
-----
String: 
Index(['id'], dtype='object')
```

3.2 Null Values

Lets check the null values

In [11]:
```
print("Column wise missing values percentage\n\n")

print("Client Data Information: \n", round((client_df.isnull().sum()/client_df.shape[0])*100,2))
print("\n--------------------------------------\n")
print("Price Data Information: \n", round((price_df.isnull().sum()/price_df.shape[0])*100,2))
```

```
Column wise missing values percentage


Client Data Information:
 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
dtype: float64

---------------------------------------

Price Data Information:
 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
dtype: float64
```

Inferences

There are no null values,so we can go for statistics now

3.3 Statistics

In [12]:
```
#statistics for numerical columns of client dataframe
client_df.describe()
```

Out[12]:

	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_
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	14606.0
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.762906	0.966726	63.086871	0.1
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.786255	5.108289	66.165783	0.0
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.000000	0.000000	16.180000	0.1
50%	1.411550e+04	0.000000e+00	792.500000	1112.875000	314.000000	0.000000	18.795000	0.1
75%	4.076375e+04	0.000000e+00	3383.000000	2401.790000	1745.750000	0.000000	131.030000	0.1
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	175375.000000	30.000000	599.310000	0.2

In [13]:
```
#statistics for text columns of client dataframe
client_df.describe(include=['object'])
```

Out[13]:

	id	channel_sales	has_gas	origin_up
count	14606	14606	14606	14606
unique	14606	8	2	6
top	59746de7e4ad448874943bcdd4aa5ee2	foosdfpfkusacimwkcsosbicdxkicaua	f	lxidpiddsbxsbosboudacockeimpuepw
freq	1	6754	11955	7097

In [14]:
```python
#statistics for date columns of client dataframe
client_df.describe(include=['datetime64'])
```

Out[14]:

	date_activ	date_end	date_modif_prod	date_renewal
count	14606	14606	14606	14606
unique	1796	368	2129	386
top	2009-08-01 00:00:00	2016-02-01 00:00:00	2015-11-01 00:00:00	2015-06-23 00:00:00
freq	95	145	721	587
first	2003-05-09 00:00:00	2016-01-28 00:00:00	2003-05-09 00:00:00	2013-06-26 00:00:00
last	2014-09-01 00:00:00	2017-06-13 00:00:00	2016-01-29 00:00:00	2016-01-28 00:00:00

In [15]:
```python
#statistics for numerical columns of price dataframe
price_df.describe()
```

Out[15]:

	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

In [16]:
```python
#statistics for text columns of price dataframe
price_df.describe(include=[object])
```

Out[16]:

	id
count	193002
unique	16096
top	59746de7e4ad448874943bcdd4aa5ee2
freq	12

In [17]:
```python
#statistics for date columns of price dataframe
price_df.describe(include=['datetime64'])
```

Out[17]:

	price_date
count	193002
unique	12
top	2015-08-01 00:00:00
freq	16094
first	2015-01-01 00:00:00
last	2015-12-01 00:00:00

Inferences

Clients Data:

1. The consumption data is highly skewed, as exhibited by the percentile values.
2. Total 8 different sales channel & 6 different origin up.
3. We have data from updated from May-2003 to Jan-2016.

Price Data:

1. peak & mid peak values are highly skewed
2. price data is a monthly data for year 2015

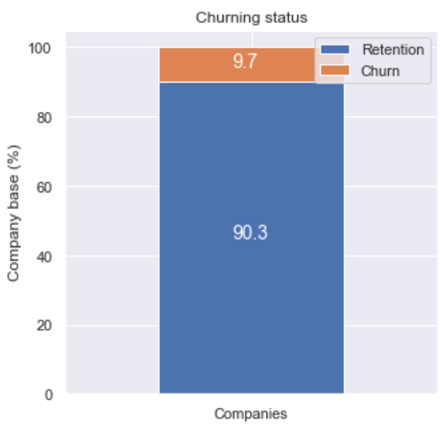
4 Data visualization

In [18]:
```
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
        )
```

In [19]:
```
def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distribution 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')
```

4.1 Churn

In [20]:
```
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))
```



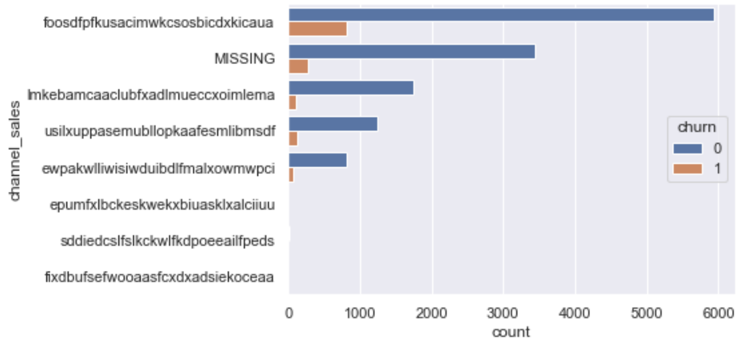
Inferences

Around 10% of clients have churned

4.2 Sales channel

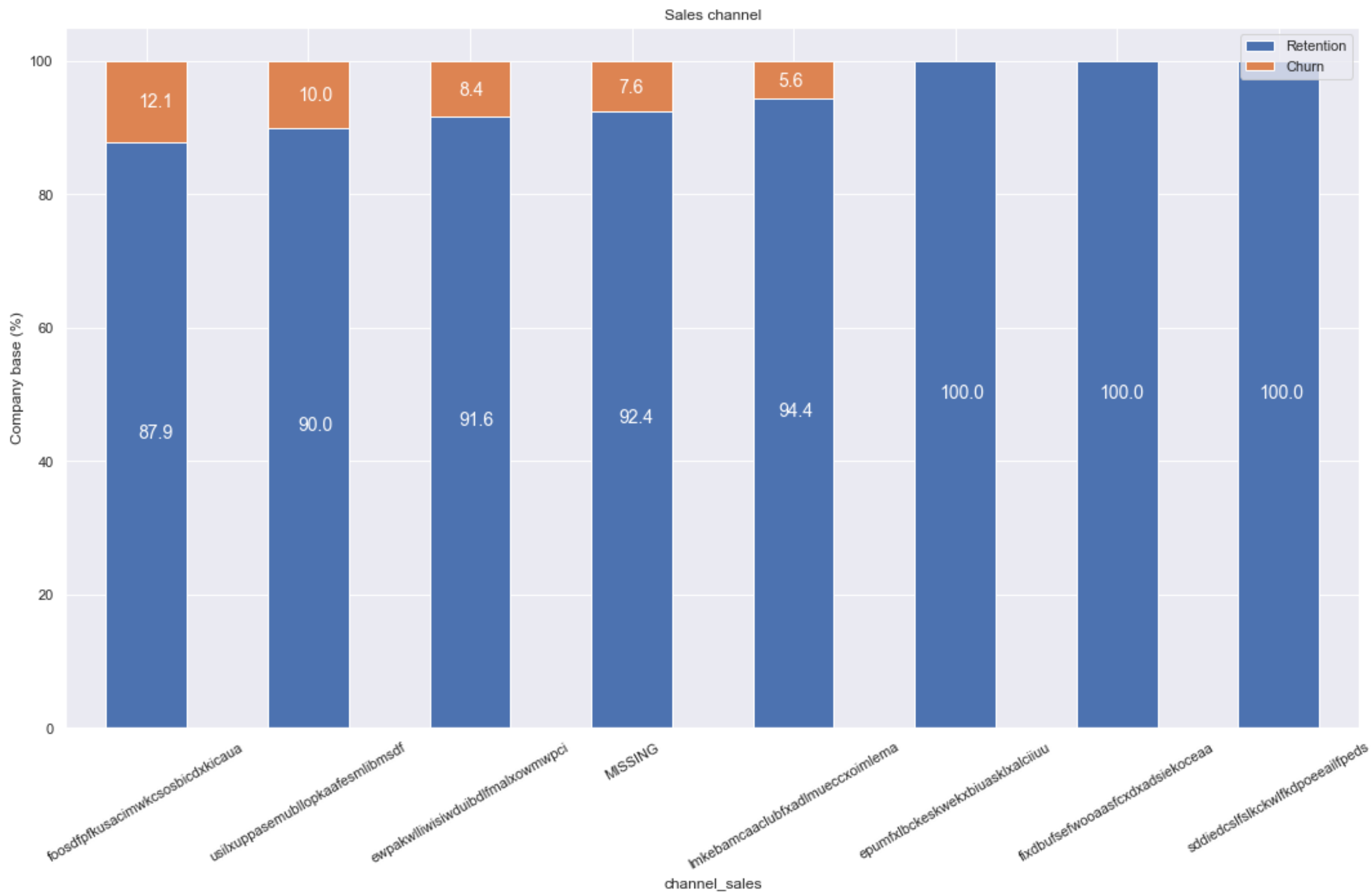
In [21]:
```
sns.countplot(data=client_df, y='channel_sales', hue="churn")
```

Out[21]:
```
<AxesSubplot:xlabel='count', ylabel='channel_sales'>
```



In [22]:
```
channel = client_df[['id', 'channel_sales', 'churn']]
channel = channel.groupby([channel['channel_sales'], channel['churn']])['id'].count().unstack(level=1).fillna(0)
channel_churn = (channel.div(channel.sum(axis=1), axis=0) * 100).sort_values(by=[1], ascending=False)
```

In [23]:
```
plot_stacked_bars(channel_churn, 'Sales channel', rot_=30)
```



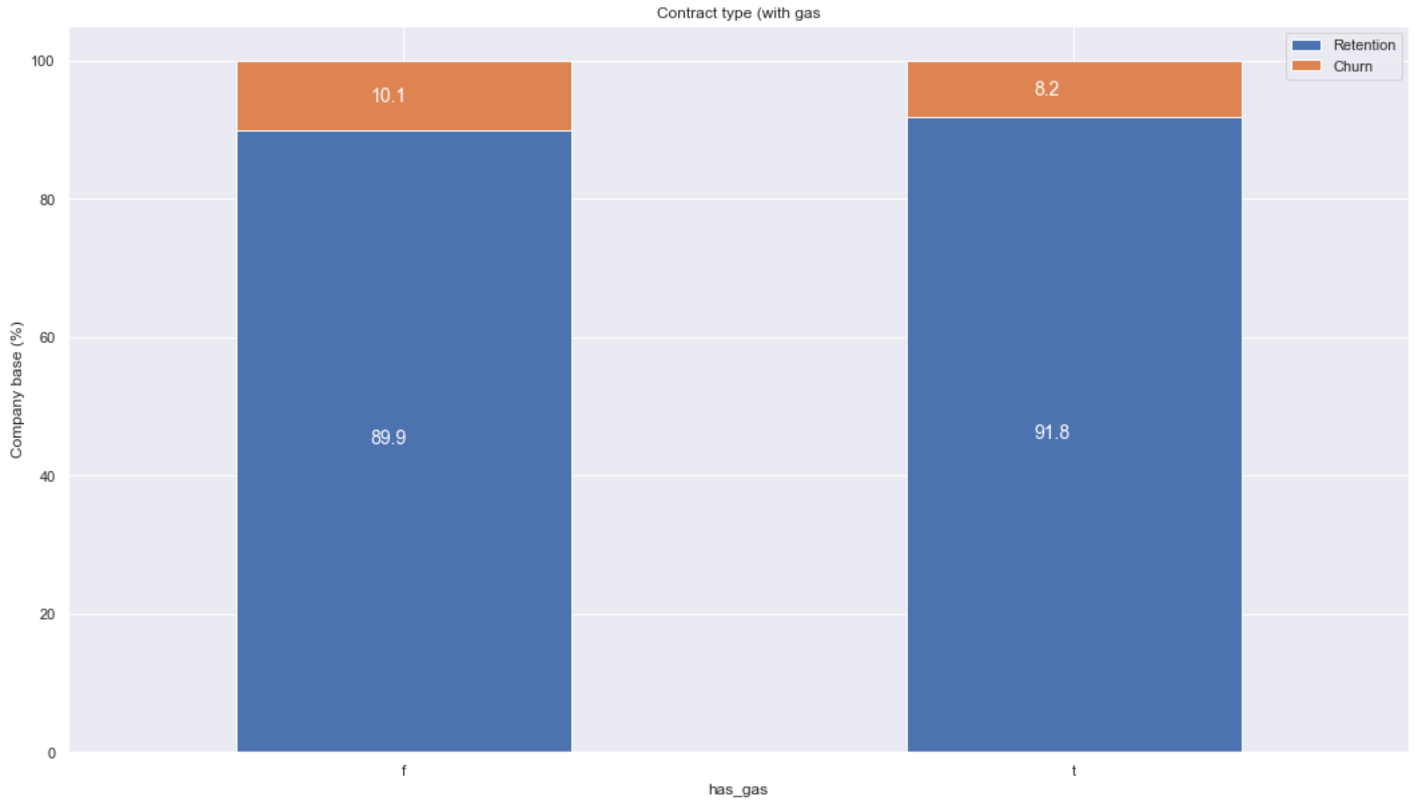
Inferences

1. Around 50% of clients are through sales channel 'foosdfpfkusacimwkcsosbicdxkicaua'
2. The churning customers are distributed over 5 different values for channel_sales.
3. MISSING indicates a missing value and was added by the team when they were cleaning the dataset. This feature could be an important feature when it comes to building our model.

4.3 Contract Type

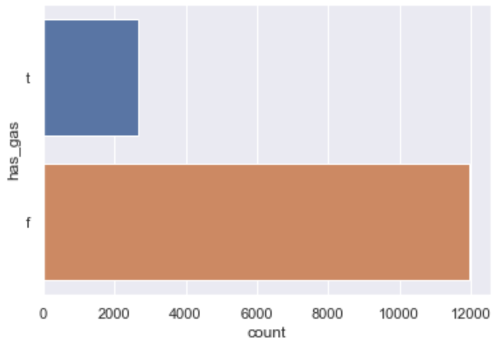
In [25]:
```
contract_type = client_df[['id', 'has_gas', 'churn']]
contract = contract_type.groupby([contract_type['churn'],contract_type['has_gas']])['id'].count().unstack(level=0)
contract_percentage = (contract.div(contract.sum(axis=1), axis=0) * 100).sort_values(by=[1], ascending=False)
```

In [26]:
```
plot_stacked_bars(contract_percentage, 'Contract type (with gas')
```



In [27]:
```
sns.countplot(data=client_df, y='has_gas')
```

Out[27]: <AxesSubplot:xlabel='count', ylabel='has_gas'>

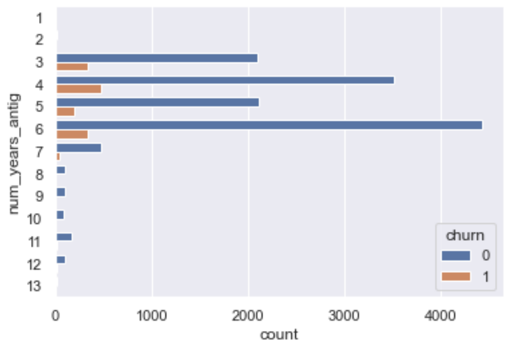


Inferences
```
1. Around 20% of clients have Gas supply also.
2. There is no big difference btw the churn of clients with gas or without gas contract
```

4.4 Antiquity

In [28]:
```
sns.countplot(data=client_df, y='num_years_antig', hue="churn")
```

Out[28]: <AxesSubplot:xlabel='count', ylabel='num_years_antig'>



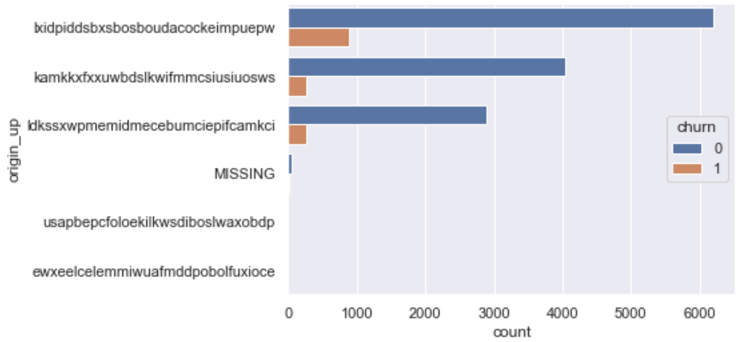
Inferences

1. Maximum Churned clients are having 4 years of antiquity.
2. Churned clients are from antiquity of 3 to 6 years

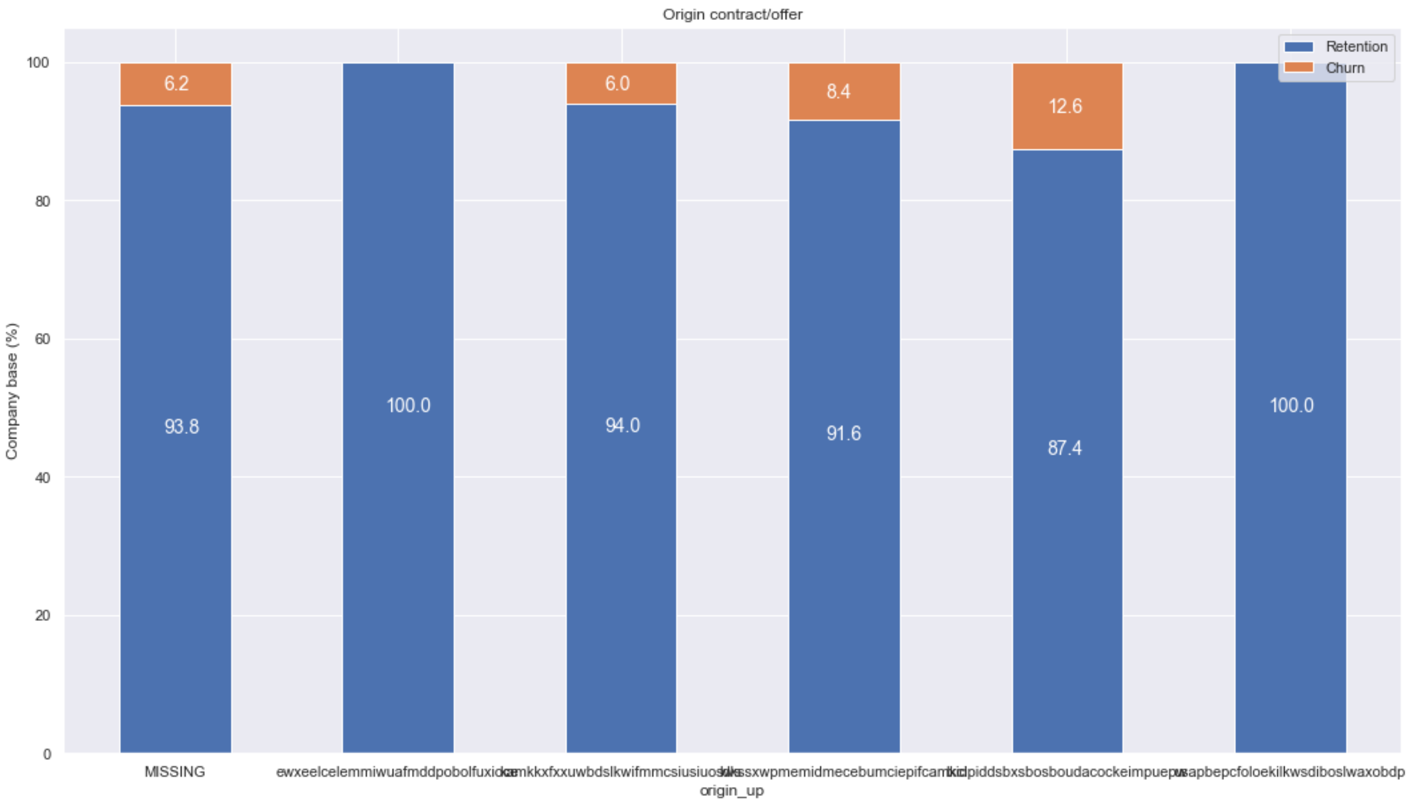
4.5 Electricity Campaign

In [29]:
```
sns.countplot(data=client_df, y='origin_up', hue="churn")
```

Out[29]:
```
<AxesSubplot:xlabel='count', ylabel='origin_up'>
```



In [30]:
```
origin_df = client_df[['id', 'origin_up','churn']]
origin = origin_df.groupby([origin_df["origin_up"],origin_df["churn"]])["id"].count().unstack(level=1)
origin_percentage = (origin.div(origin.sum(axis=1), axis=0)*100)
plot_stacked_bars(origin_percentage, "Origin contract/offer")
```



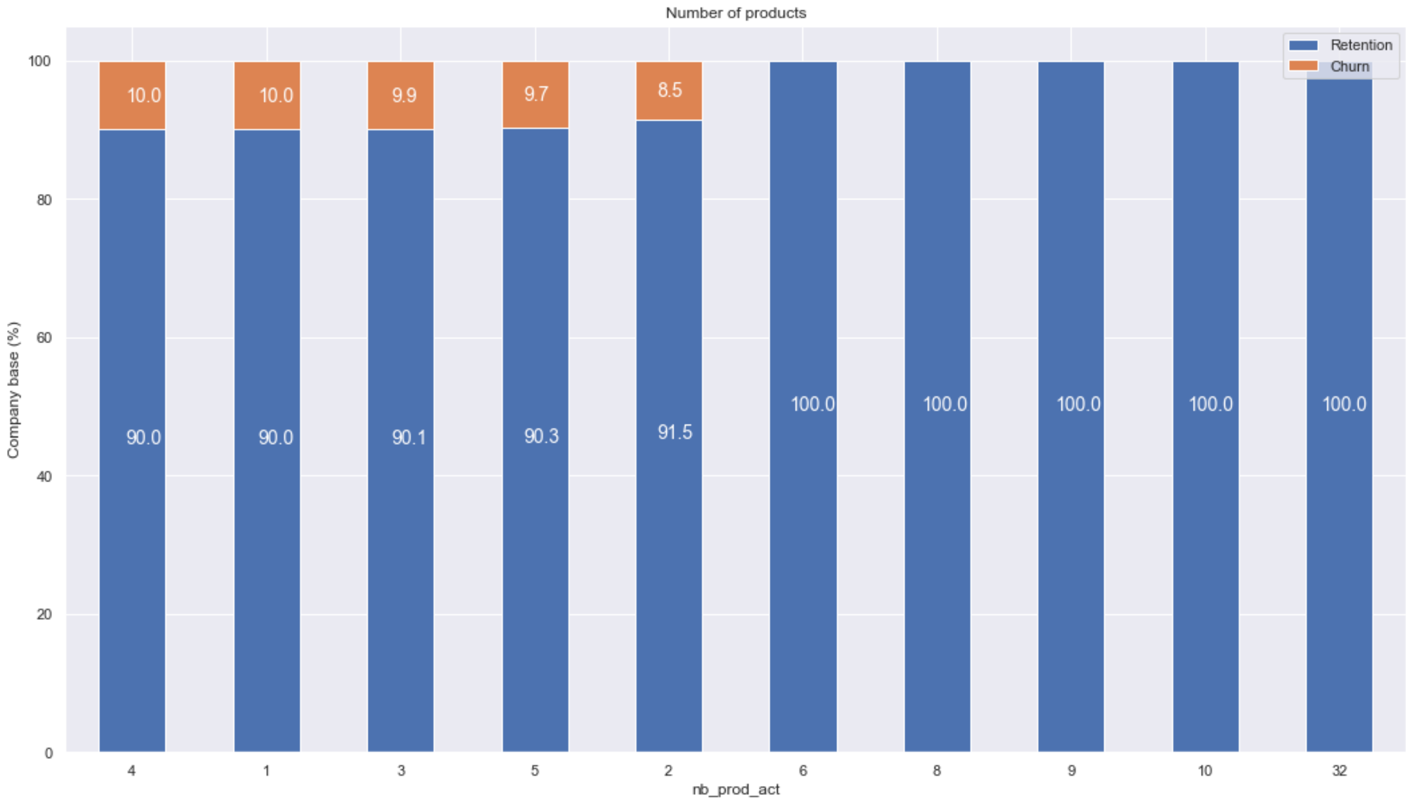
Inferences

1. Around 50% of customer first subscribed to electricity campaign 'lxidpiddsbxsbosboudacockeimpuepw'
2. The churning customers are distributed over 4 different values for electricity campaign.
3. MISSING indicates a missing value and was added by the team when they were cleaning the dataset. This feature could be an important feature when it comes to building our model.

4.6 Number of Products

In [31]:
```
nprod_df = client_df[['id', 'nb_prod_act','churn']]
products = nprod_df.groupby([nprod_df["nb_prod_act"],nprod_df["churn"]])["id"].count().unstack(level=1)
products_percentage = (products.div(products.sum(axis=1), axis=0)*100).sort_values(by=[1], ascending=False)
```

In [32]:
```
plot_stacked_bars(products_percentage, "Number of products")
```



Inferences

The churning customers are distributed over 1 to 5 products

4.7 Consumption

In [33]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month','imp_cons', 'has_gas', 'churn']]
```

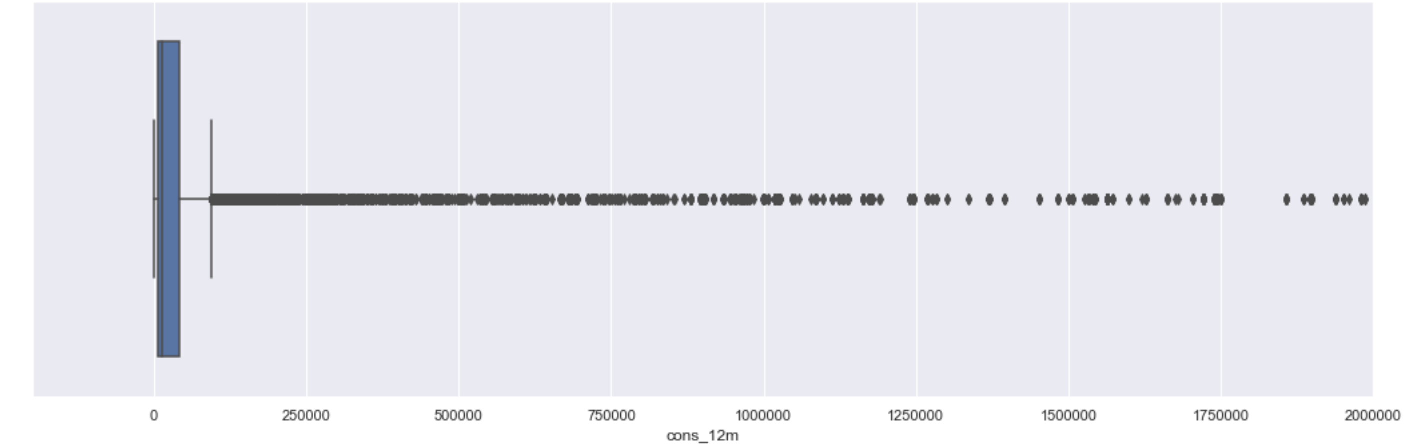
In [34]:
```
fig, axs = plt.subplots(nrows=4, figsize=(18, 25))
plot_distribution(consumption, 'cons_12m', axs[0])
plot_distribution(consumption[consumption['has_gas'] == 't'], 'cons_gas_12m',axs[1])
plot_distribution(consumption, 'cons_last_month', axs[2])
plot_distribution(consumption, 'imp_cons', axs[3])
```

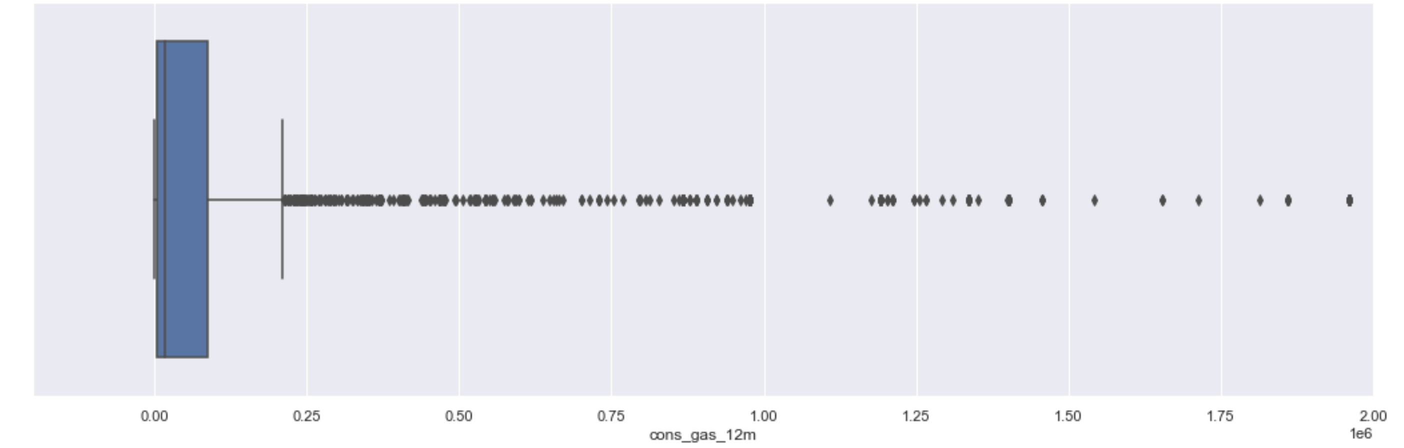
Inferenece s

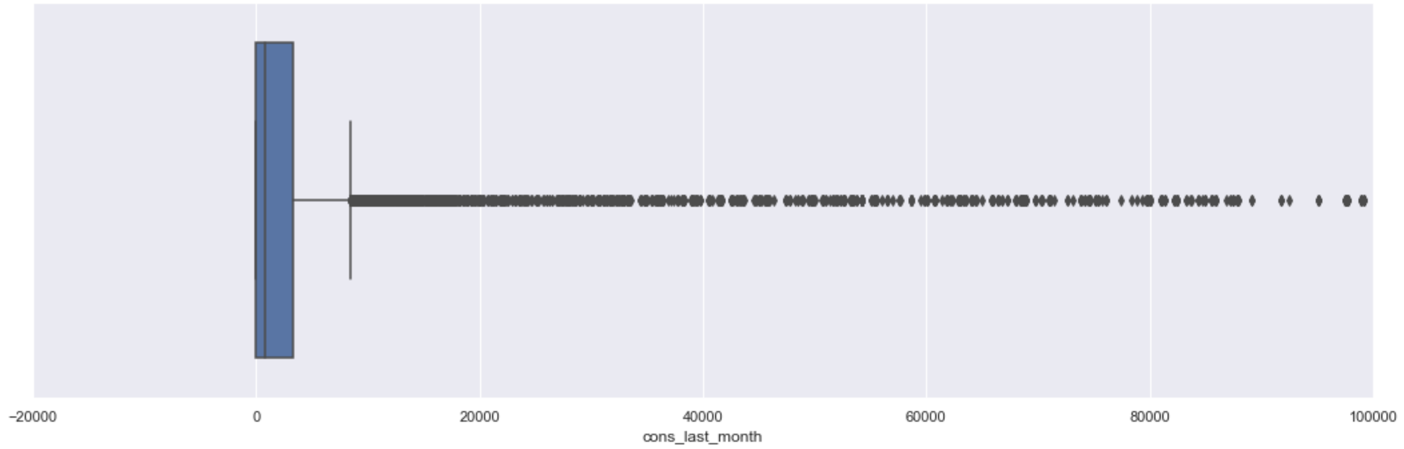
1. The consumption data is highly positively skewed, presenting a very long right-tail towards the higher values of the distribution.
2. The values on the higher and lower end of the distribution are likely to be outliers.

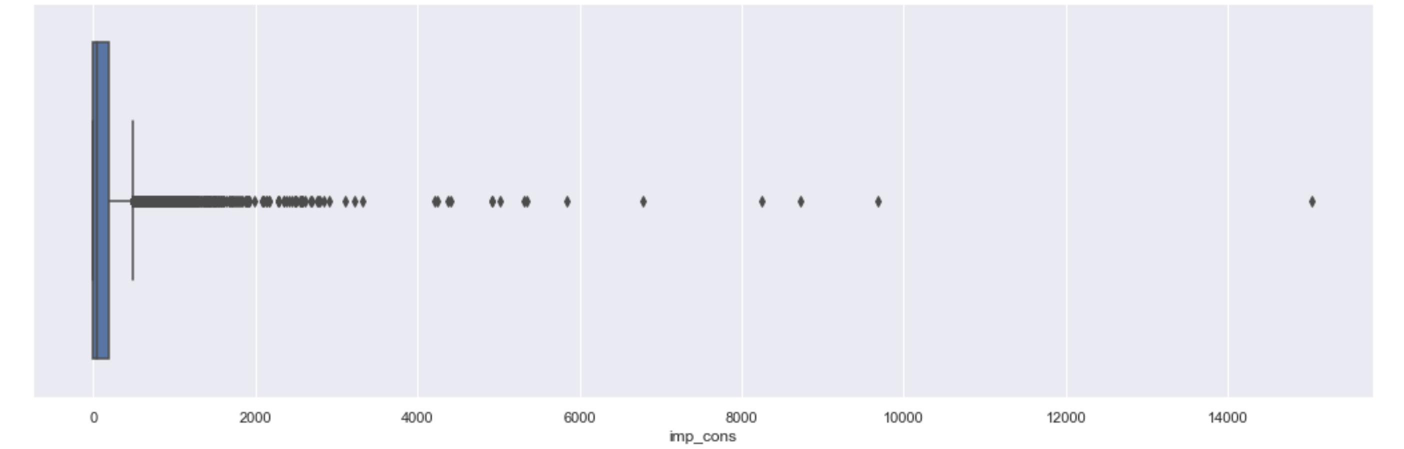
We can use a standard plot to visualise the outliers in more detail. A boxplot is a standardized way of displaying the distribution based on a five number summary: Minimum - First quartile (Q1) - Median - Third quartile (Q3) - Maximum It can reveal outliers and what their values are. It can also tell us if our data is symmetrical, how tightly our data is grouped and if/how our data is skewed.

In [35]:
```
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()
```





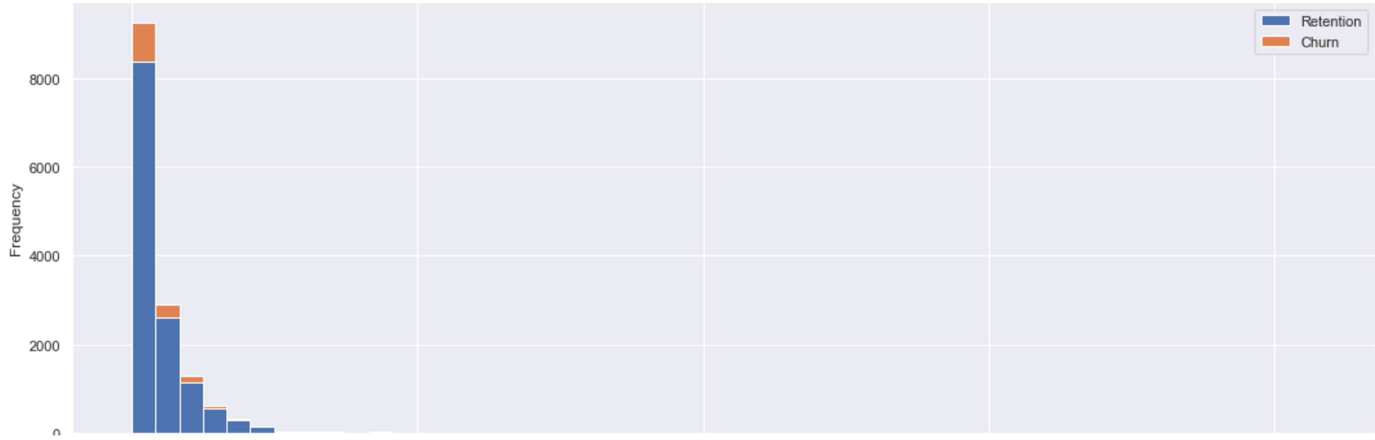




4.8 Forecast

In [36]:
```
forecast = client_df[['id','forecast_cons_12m','forecast_cons_year', 'forecast_discount_energy','forecast_meter_rent_12m', 'forecast_price_
```

In [37]:
```
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])
```



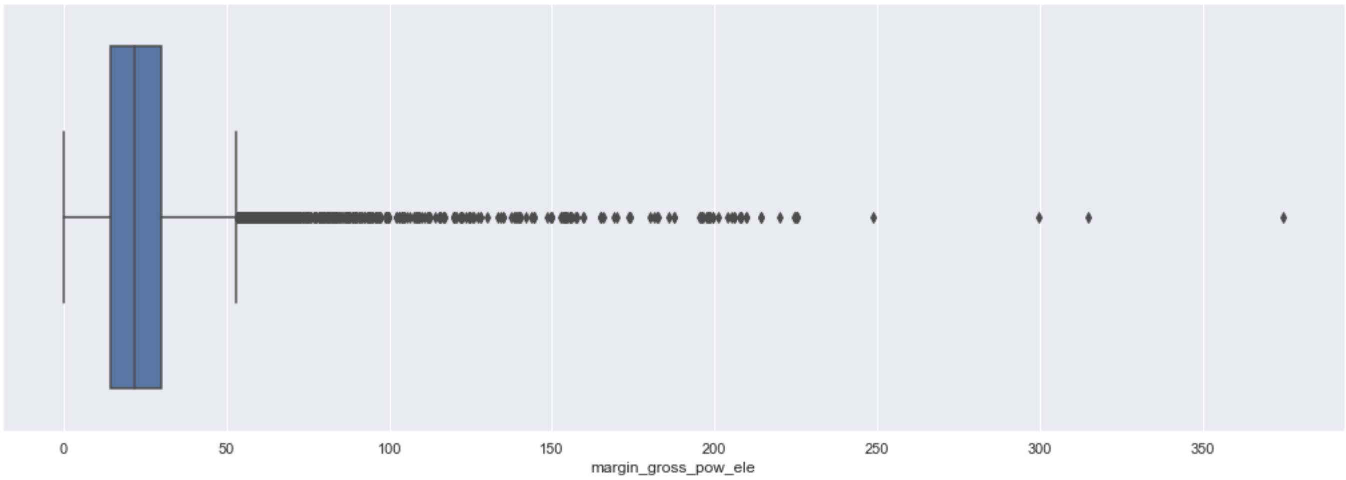
Inferences

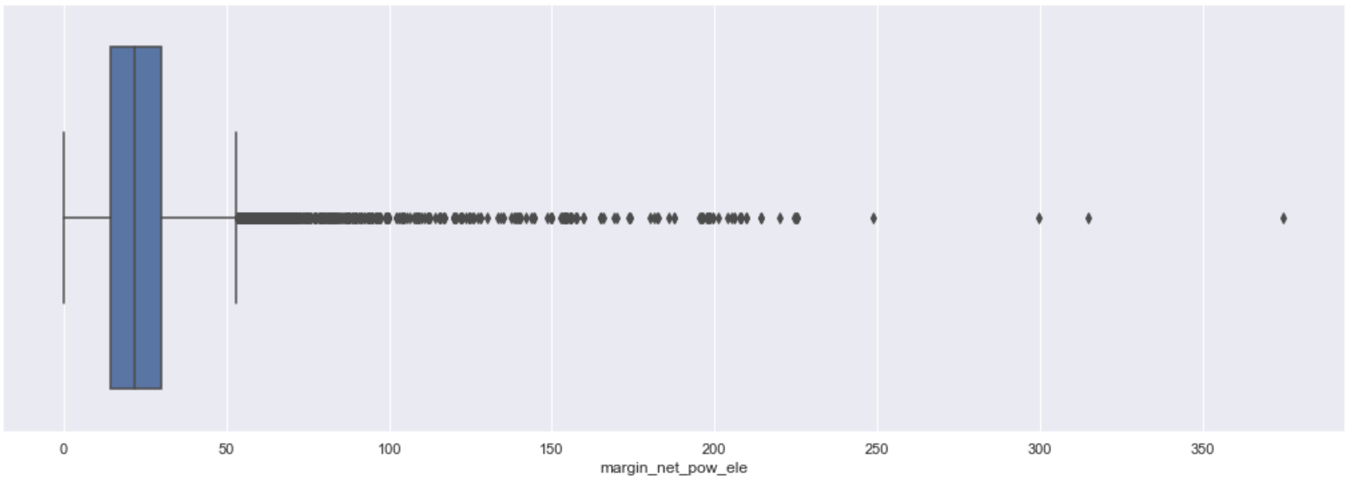
The variables are highly positively skewed, creating a very long tail for the higher values.

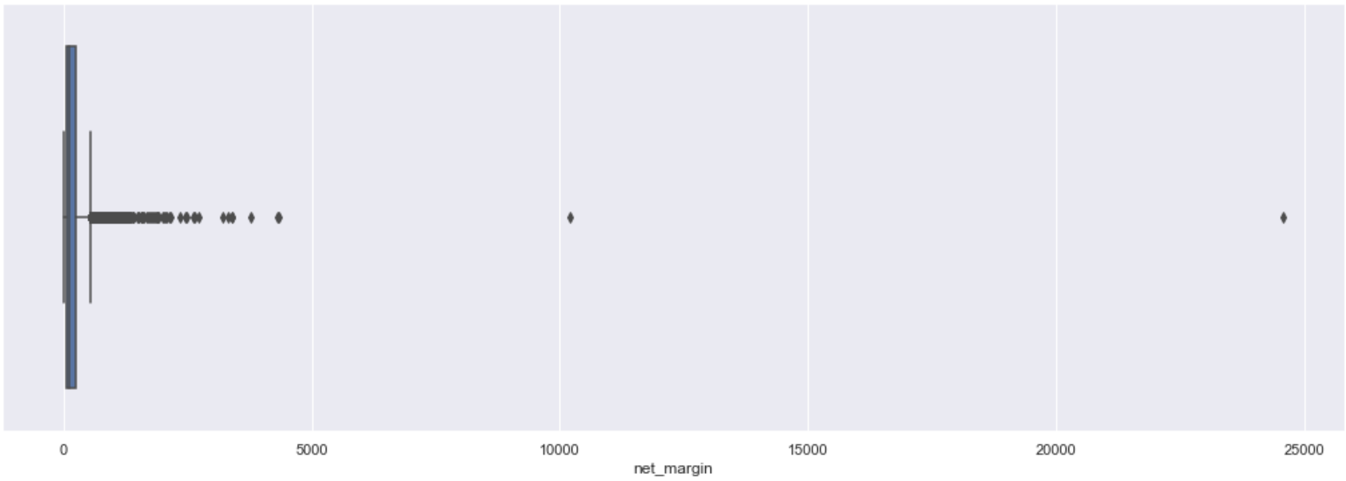
4.9 Margins

In [38]:
```
margin = client_df[['id', 'margin_gross_pow_ele', 'margin_net_pow_ele','net_margin']]
```

In [39]:
```
fig, axs = plt.subplots(nrows=3, figsize=(18,20))
# Plot histogram
sns.boxplot(margin["margin_gross_pow_ele"], ax=axs[0])
sns.boxplot(margin["margin_net_pow_ele"],ax=axs[1])
sns.boxplot(margin["net_margin"], ax=axs[2])
15
# Remove scientific notation
axs[0].ticklabel_format(style='plain', axis='x')
axs[1].ticklabel_format(style='plain', axis='x')
axs[2].ticklabel_format(style='plain', axis='x')
plt.show()
```







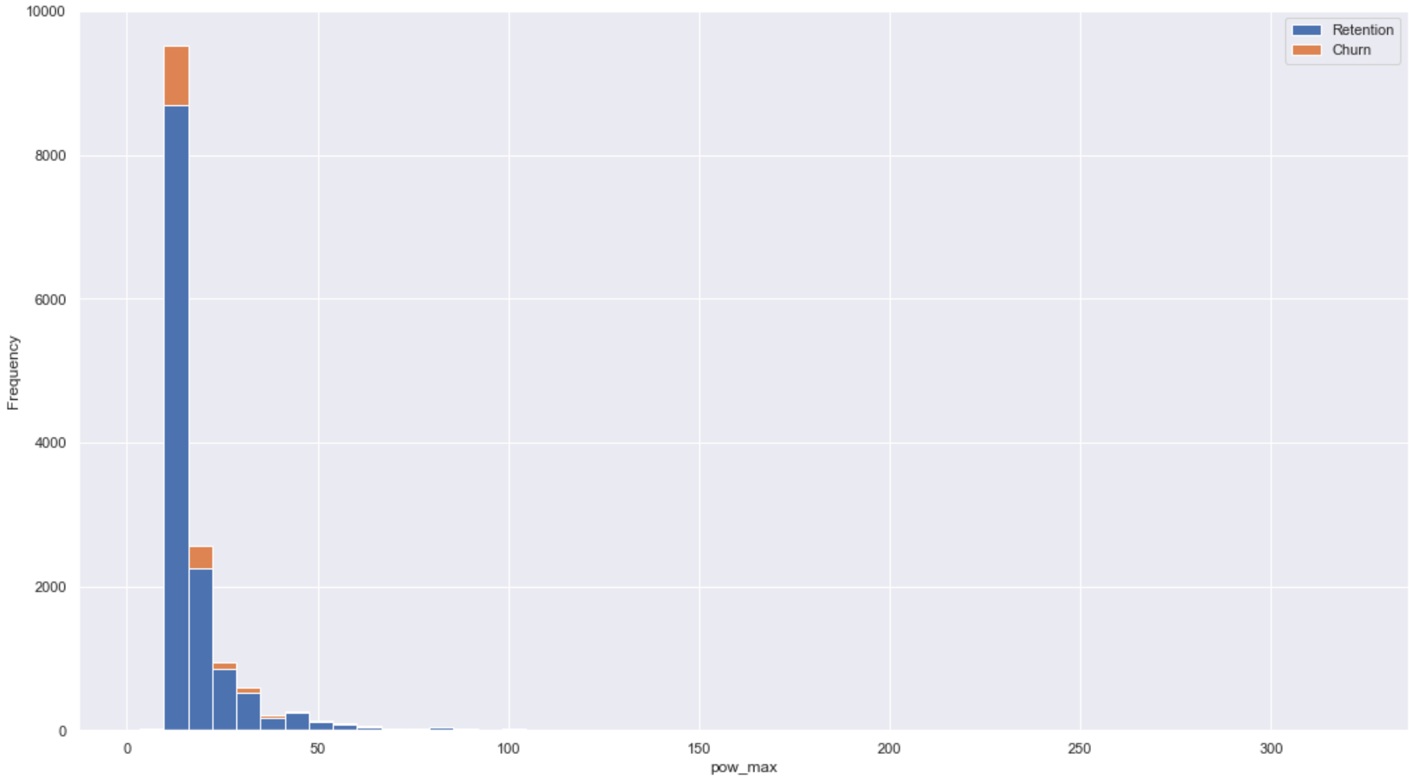
Inferences

We can see there are multiple outliers

4.10 Subscribed Power

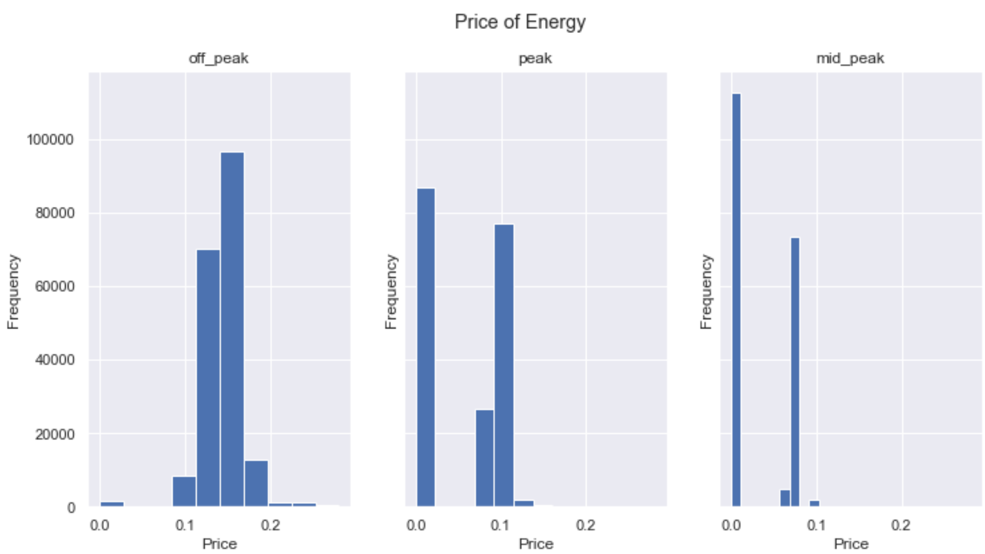
In [40]:
```
power = client_df[['id', 'pow_max', 'churn']]
```

In [41]:
```
fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)
```

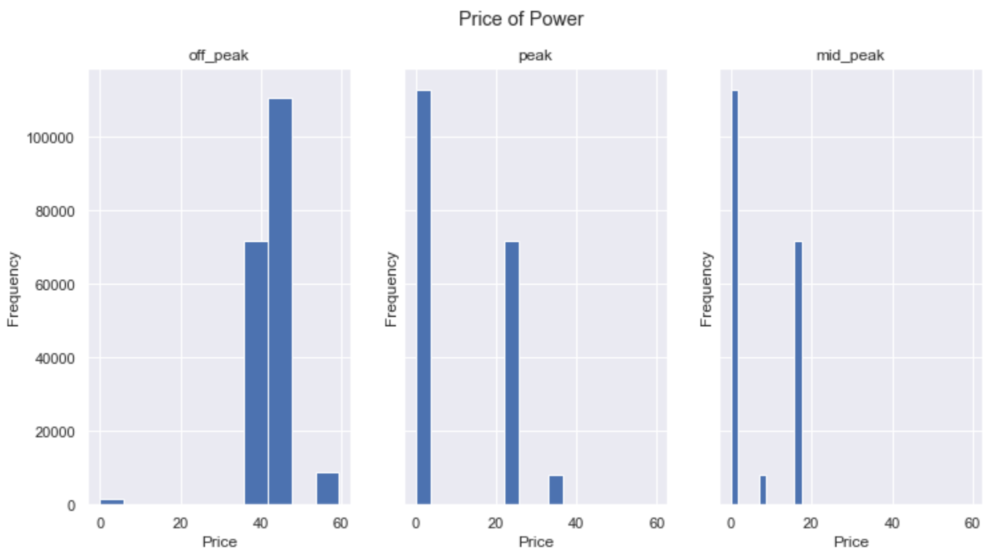


4.11 Price Data

In [42]:
```
fig, axs = plt.subplots(1, 3,figsize=(12, 6),sharey=True , sharex=True)
# Plot the histograms on the subplots
axs[0].hist(price_df["price_off_peak_var"])
axs[1].hist(price_df["price_peak_var"])
axs[2].hist(price_df["price_mid_peak_var"])

# Set titles and labels for each subplot
axs[0].set_title('off_peak')
axs[1].set_title('peak')
axs[2].set_title('mid_peak')

axs[0].set_ylabel('Frequency')
axs[0].set_xlabel('Price')
axs[1].set_ylabel('Frequency')
axs[1].set_xlabel('Price')
axs[2].set_ylabel('Frequency')
axs[2].set_xlabel('Price')

fig.suptitle('Price of Energy')

# Display the plot
plt.show()
```



In [43]:
```
fig, axs = plt.subplots(1, 3,figsize=(12, 6),sharey=True , sharex=True)
# Plot the histograms on the subplots
axs[0].hist(price_df["price_off_peak_fix"])
axs[1].hist(price_df["price_peak_fix"])
axs[2].hist(price_df["price_mid_peak_fix"])

# Set titles and labels for each subplot
axs[0].set_title('off_peak')
axs[1].set_title('peak')
axs[2].set_title('mid_peak')

axs[0].set_ylabel('Frequency')
axs[0].set_xlabel('Price')
axs[1].set_ylabel('Frequency')
axs[1].set_xlabel('Price')
axs[2].set_ylabel('Frequency')
axs[2].set_xlabel('Price')

fig.suptitle('Price of Power')

# Display the plot
plt.show()
```



5 Hypothesis Investigation

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

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

In [60]:
```
mean_6m
```

Out[60]:

	id	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_mid_peak_fix
0	0002203ffbb812588b632b9e628cc38d	0.000011	0.000003	4.860000e-10	0.000000	0.000000	0.000000
1	0004351ebdd665e6ee664792efc4fd13	0.000003	0.000000	0.000000e+00	0.000000	0.000000	0.000000
2	0010bcc39e42b3c2131ed2ce55246e3c	0.000003	0.000000	0.000000e+00	0.000000	0.000000	0.000000
3	0010ee3855fdea87602a5b7aba8e42de	0.000011	0.000003	4.860000e-10	0.000000	0.000000	0.000000
4	00114d74e963e47177db89bc70108537	0.000003	0.000000	0.000000e+00	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...
16091	ffef185810e44254c3a4c6395e6b4d8a	0.000011	0.000003	4.860000e-10	0.000000	0.000000	0.000000
16092	fffac626da707b1b5ab11e8431a4d0a2	0.000003	0.000000	0.000000e+00	0.009482	0.000000	0.000000
16093	fffc0cacd305dd51f316424bbb08d1bd	0.000011	0.000003	4.860000e-10	0.000000	0.000000	0.000000
16094	fffe4f5646aa39c7f97f95ae2679ce64	0.000014	0.000004	3.406563e-07	0.007962	0.002867	0.001274
16095	ffff7fa066f1fb305ae285bb03bf325a	0.000011	0.000003	4.860000e-10	0.000000	0.000000	0.000000

16096 rows × 7 columns

In [59]:
```
# Create yearly sensitivity features
mean_year = price_df.groupby(['id', 'price_date']).mean().groupby(['id']).var().reset_index()
# Create last 6 months sensitivity features
mean_6m = price_df[price_df['price_date'] > '2015-06-01'].groupby(['id', 'price_date']).mean().groupby(['id']).var().reset_index()
```

In [61]:
```python
# Comnbine into single dataframe
mean_year = mean_year.rename(
    index=str,
    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"
    }
)

mean_year["mean_year_price_off_peak"] = mean_year["mean_year_price_off_peak_var"] + mean_year["mean_year_price_off_peak_fix"]
mean_year["mean_year_price_peak"] = mean_year["mean_year_price_peak_var"] + mean_year["mean_year_price_peak_fix"]
mean_year["mean_year_price_mid_peak"] = mean_year["mean_year_price_mid_peak_var"] + mean_year["mean_year_price_mid_peak_fix"]

mean_6m = mean_6m.rename(
    index=str,
    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_price_off_peak_fix"]
mean_6m["mean_6m_price_peak"] = mean_6m["mean_6m_price_peak_var"] + mean_6m["mean_6m_price_peak_fix"]
mean_6m["mean_6m_price_mid_peak"] = mean_6m["mean_6m_price_mid_peak_var"] + mean_6m["mean_6m_price_mid_peak_fix"]

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

In [62]:
```python
price_features.head()
```

Out[62]:

	id	mean_year_price_off_peak_var	mean_year_price_peak_var	mean_year_price_mid_peak_var	mean_year_price_off_peak_fix	mean_year_p
0	0002203ffbb812588b632b9e628cc38d	0.000016	0.000004	1.871602e-06	4.021438e-03	
1	0004351ebdd665e6ee664792efc4fd13	0.000005	0.000000	0.000000e+00	7.661891e-03	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.000676	0.000000	0.000000e+00	5.965909e-01	
3	0010ee3855fdea87602a5b7aba8e42de	0.000025	0.000007	1.627620e-07	7.238536e-03	
4	00114d74e963e47177db89bc70108537	0.000005	0.000000	0.000000e+00	3.490909e-13	

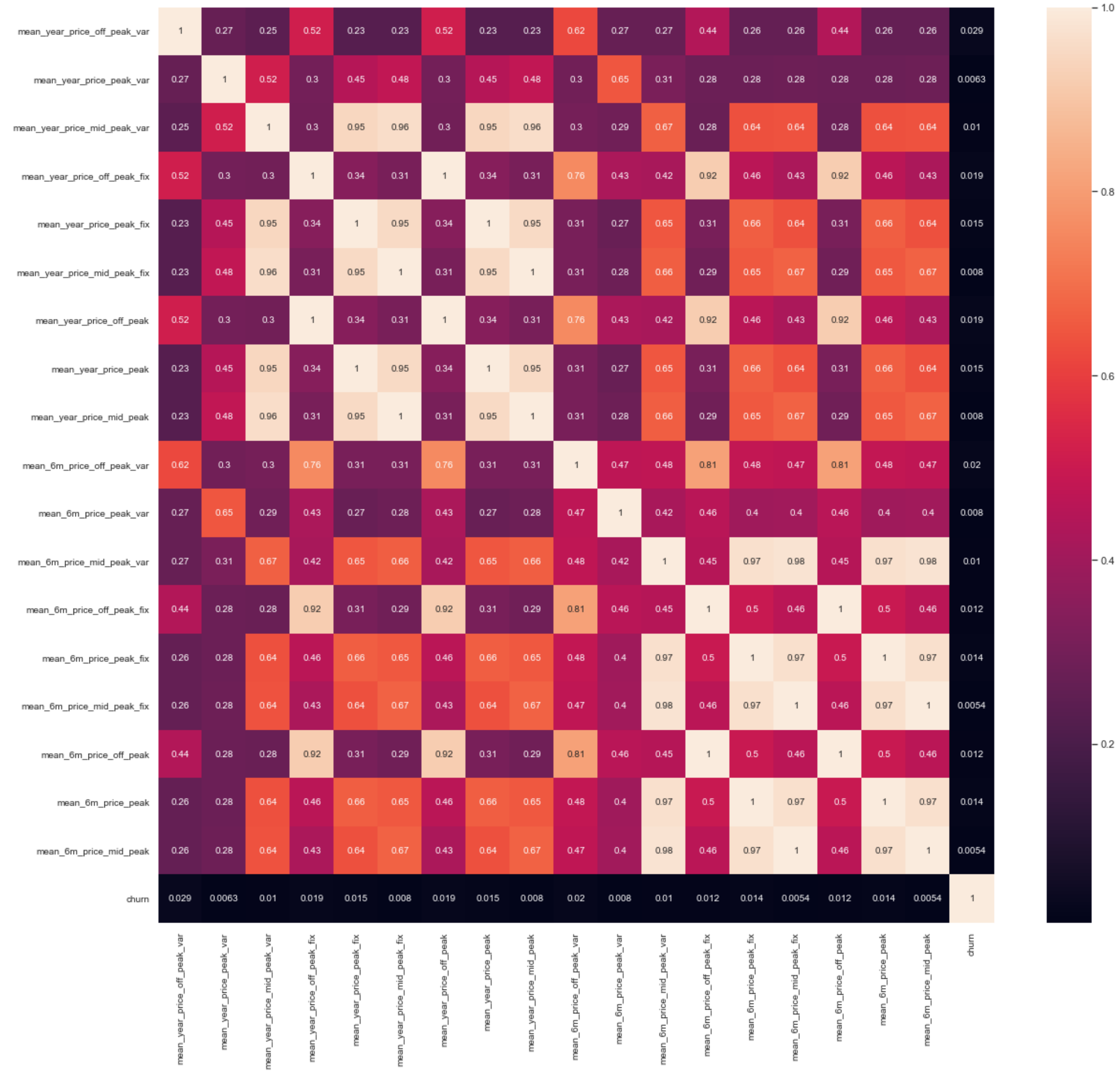
Now lets merge in the churn data and see whether price sensitivity has any correlation with churn

In [63]:
```python
price_analysis = pd.merge(price_features, client_df[['id', 'churn']], on='id')
price_analysis.head()
```

Out[63]:

	id	mean_year_price_off_peak_var	mean_year_price_peak_var	mean_year_price_mid_peak_var	mean_year_price_off_peak_fix	mean_year_p
0	0002203ffbb812588b632b9e628cc38d	0.000016	0.000004	0.000002	4.021438e-03	
1	0004351ebdd665e6ee664792efc4fd13	0.000005	0.000000	0.000000	7.661891e-03	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.000676	0.000000	0.000000	5.965909e-01	
3	00114d74e963e47177db89bc70108537	0.000005	0.000000	0.000000	3.490909e-13	
4	0013f326a839a2f6ad87a1859952d227	0.000016	0.000004	0.000002	0.000000e+00	

In [64]:
```python
corr = price_analysis.corr()
# Plot correlation
plt.figure(figsize=(20,18))
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':10})
# Axis ticks size
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



Inferences

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

In [ ]: