Exploratory Data Analysis Starter

Import packages

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Shows plots in jupyter notebook
%matplotlib inline

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

Loading data with Pandas

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

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

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

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

Out[3]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	

3 rows × 26 columns

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

Out[4]:

	id	price_date	price_off_peak_var	price_peak_var	price_mid_peak_va
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.

Descriptive statistics of data

Data types

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

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

In [5]:
```
client_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              14606 non-null  object 
 1   channel_sales                   14606 non-null  object 
 2   cons_12m                        14606 non-null  int64  
 3   cons_gas_12m                    14606 non-null  int64  
 4   cons_last_month                 14606 non-null  int64  
 5   date_activ                      14606 non-null  object 
 6   date_end                        14606 non-null  object 
 7   date_modif_prod                 14606 non-null  object 
 8   date_renewal                    14606 non-null  object 
 9   forecast_cons_12m               14606 non-null  float64
 10  forecast_cons_year              14606 non-null  int64  
 11  forecast_discount_energy        14606 non-null  float64
 12  forecast_meter_rent_12m         14606 non-null  float64
 13  forecast_price_energy_off_peak  14606 non-null  float64
 14  forecast_price_energy_peak      14606 non-null  float64
 15  forecast_price_pow_off_peak     14606 non-null  float64
 16  has_gas                         14606 non-null  object 
 17  imp_cons                        14606 non-null  float64
 18  margin_gross_pow_ele            14606 non-null  float64
 19  margin_net_pow_ele              14606 non-null  float64
 20  nb_prod_act                     14606 non-null  int64  
 21  net_margin                      14606 non-null  float64
 22  num_years_antig                 14606 non-null  int64  
 23  origin_up                       14606 non-null  object 
 24  pow_max                         14606 non-null  float64
 25  churn                           14606 non-null  int64  
dtypes: float64(11), int64(7), object(8)
memory usage: 2.9+ MB
```

In [6]:
```
price_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 8 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  193002 non-null  object 
 1   price_date          193002 non-null  object 
 2   price_off_peak_var  193002 non-null  float64
 3   price_peak_var      193002 non-null  float64
 4   price_mid_peak_var  193002 non-null  float64
 5   price_off_peak_fix  193002 non-null  float64
 6   price_peak_fix      193002 non-null  float64
 7   price_mid_peak_fix  193002 non-null  float64
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

Checking for duplicates

In [7]:
```
# A function to check for duplicates in our datasets
def identify_duplicates(df):
    """Simple function to identify any duplicates"""
    # identify the duplicates (dataframename.duplicated() , can add .sum() to get to
    # empty list to store Bool results from duplicated
    duplicates = []
    for i in df.duplicated():
        duplicates.append(i)
    # identify if there is any duplicates. (If there is any we expect a True value i
    duplicates_set = set(duplicates)
    if (len(duplicates_set) == 1):
        print("The Data has no duplicates")
    else:
        no_true = 0
        for val in duplicates:
            if (val == True):
                no_true += 1
        # percentage of the data represented by duplicates
        duplicates_percentage = np.round(((no_true / len(df)) * 100), 3)
        print(f"The Data has {no_true} duplicated rows.\nThis constitutes {duplicate

identify_duplicates(client_df)
```

```
The Data has no duplicates
```

In [8]:
```
# A function to check for duplicates in our datasets
def identify_duplicates(df):
    """Simple function to identify any duplicates"""
    # identify the duplicates (dataframename.duplicated() , can add .sum() to get to
    # empty list to store Bool results from duplicated
    duplicates = []
    for i in df.duplicated():
        duplicates.append(i)
    # identify if there is any duplicates. (If there is any we expect a True value i
    duplicates_set = set(duplicates)
    if (len(duplicates_set) == 1):
        print("The Data has no duplicates")
    else:
        no_true = 0
        for val in duplicates:
            if (val == True):
                no_true += 1
        # percentage of the data represented by duplicates
        duplicates_percentage = np.round(((no_true / len(df)) * 100), 3)
        print(f"The Data has {no_true} duplicated rows.\nThis constitutes {duplicate

identify_duplicates(price_df)
```

```
The Data has no duplicates
```

Checking for Missing Values

In [9]:
```
# A function to check for missing values in our dataset
def check_missing_values(data):
    # Count missing values in each column
    missing_values = data.isnull().sum()

    # Convert missing values count to percentage of total rows
    missing_percent = (missing_values / len(data)) * 100

    # Combine the missing values count and percent into a DataFrame
    missing_df = pd.concat([missing_values, missing_percent], axis=1)
    missing_df.columns = ['Missing Values', '% of Total']
# Return only columns with missing values
    missing_df = missing_df[missing_df['Missing Values'] > 0]

    return missing_df

# Check missing values in each dataset
display(check_missing_values(client_df))
```

Missing Values	% of Total

In [10]:
```
# A function to check for missing values in our dataset
def check_missing_values(data):
    # Count missing values in each column
    missing_values = data.isnull().sum()

    # Convert missing values count to percentage of total rows
    missing_percent = (missing_values / len(data)) * 100

    # Combine the missing values count and percent into a DataFrame
    missing_df = pd.concat([missing_values, missing_percent], axis=1)
    missing_df.columns = ['Missing Values', '% of Total']
# Return only columns with missing values
    missing_df = missing_df[missing_df['Missing Values'] > 0]

    return missing_df

# Check missing values in each dataset
display(check_missing_values(price_df))
```

Missing Values	% of Total

Statistics

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

In [11]:
```
client_df.describe()
```

Out[11]:

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

In [12]:
```
price_df.describe()
```

Out[12]:

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_
count	193002.000000	193002.000000	193002.000000	193002.000000	193002.000000	19
mean	0.141027	0.054630	0.030496	43.334477	10.622875	
std	0.025032	0.049924	0.036298	5.410297	12.841895	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.125976	0.000000	0.000000	40.728885	0.000000	
50%	0.146033	0.085483	0.000000	44.266930	0.000000	
75%	0.151635	0.101673	0.072558	44.444710	24.339581	
max	0.280700	0.229788	0.114102	59.444710	36.490692	

Data visualization

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

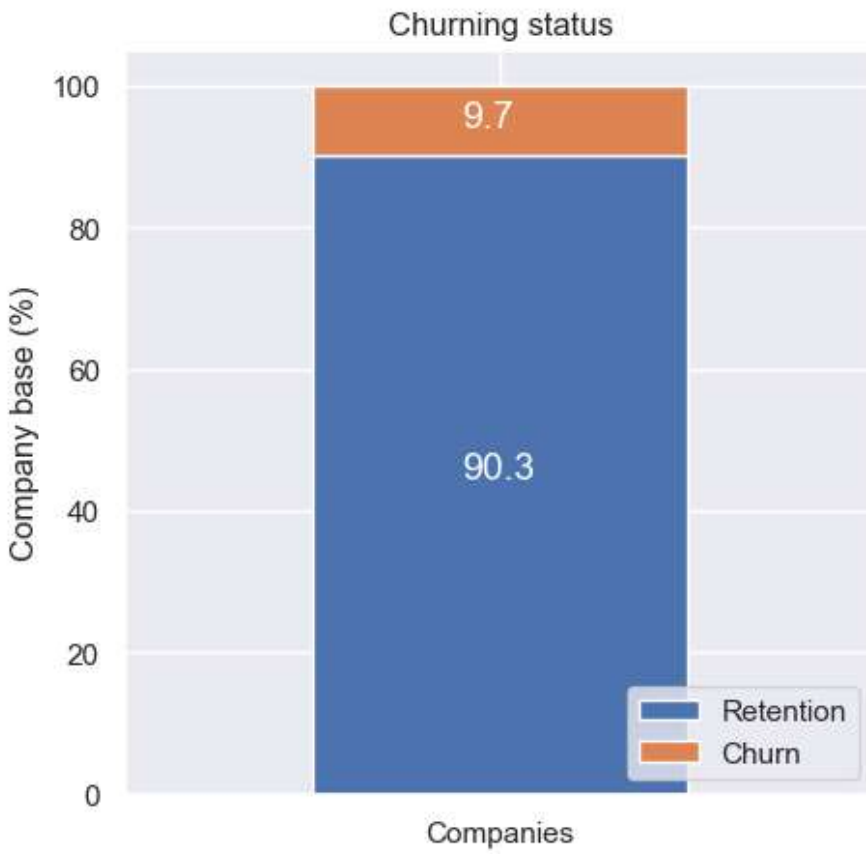
Below are some functions that you can use to get started with visualizations.

In [13]:
```
def plot_stacked_bars(dataframe, title_, size_=(18, 10), rot_=0, legend_="upper righ
    """
    Plot stacked bars with annotations
    """
    ax = dataframe.plot(
        kind="bar",
        stacked=True,
        figsize=size_,
        rot=rot_,
        title=title_
    )

    # Annotate bars
    annotate_stacked_bars(ax, textsize=14)
    # Rename legend
    plt.legend(["Retention", "Churn"], loc=legend_)
    # Labels
    plt.ylabel("Company base (%)")
    plt.show()

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

    # Iterate over the plotted rectanges/bars
    for p in ax.patches:

        # Calculate annotation
        value = str(round(p.get_height(),1))
        # If value is 0 do not annotate
        if value == '0.0':
            continue
        ax.annotate(
            value,
            ((p.get_x()+ p.get_width()/2)*pad-0.05, (p.get_y()+p.get_height()/2)*pad
            color=colour,
            size=textsize
        )

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

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

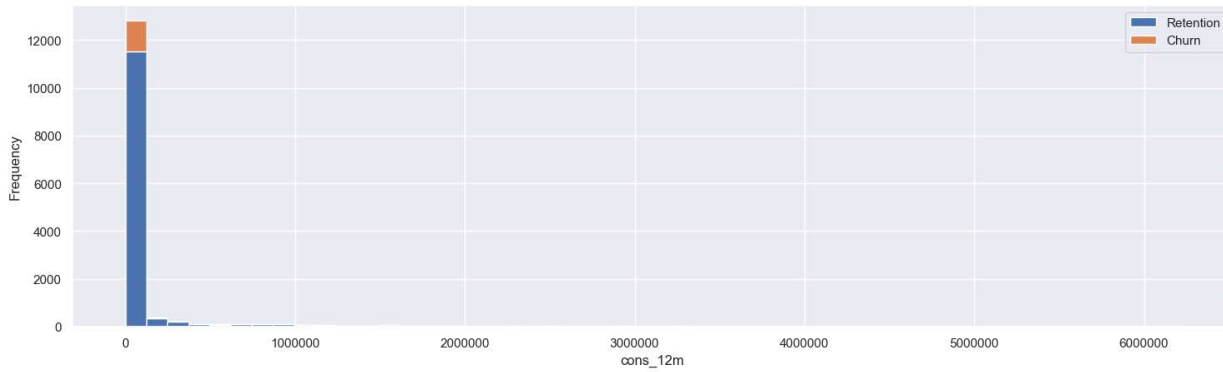
Churn

In [14]:
```
churn = client_df[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5), legend_="
```



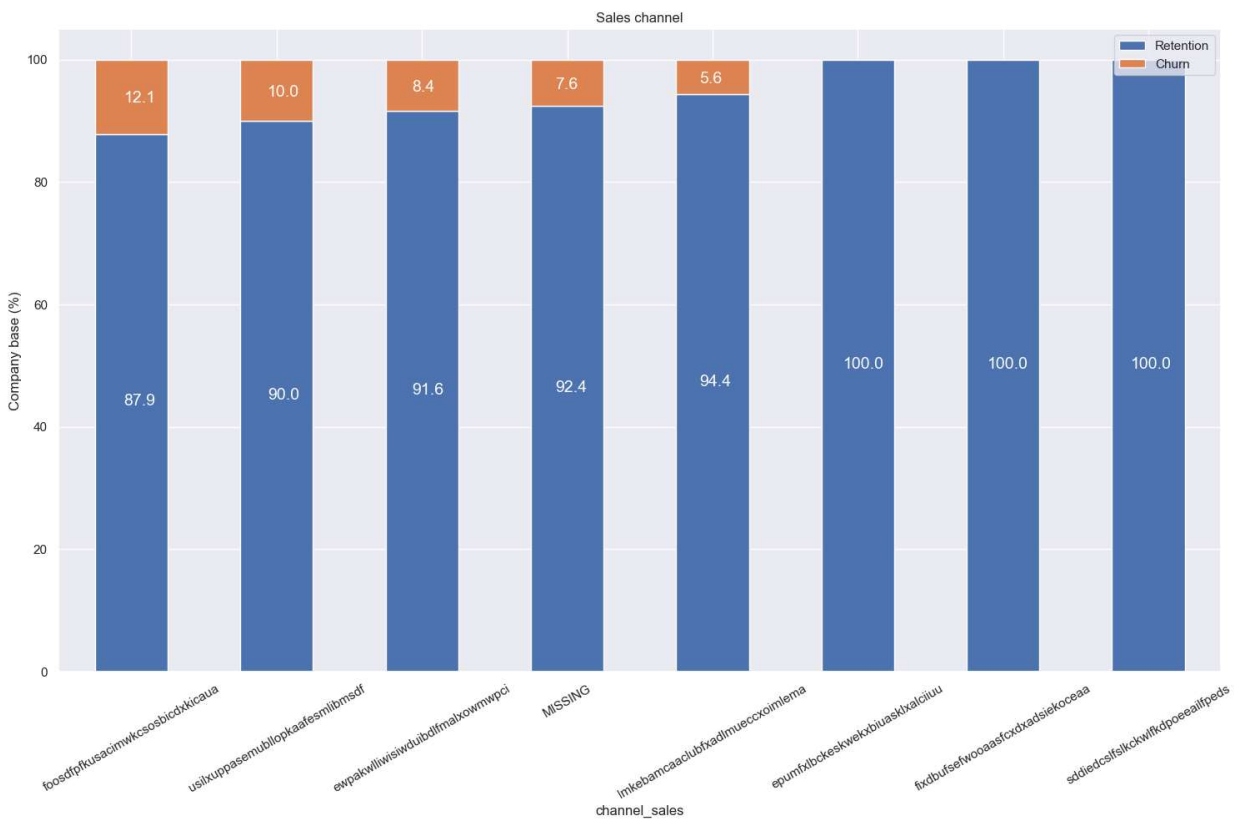
The second function `annotate_bars` is used by the first function, but the third function `plot_distribution` helps you to plot the distribution of a numeric column. An example of how it can be used is given below:

In [15]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_c

fig, axs = plt.subplots(nrows=1, figsize=(18, 5))

plot_distribution(consumption, 'cons_12m', axs)
```



Sales channel

In [16]:
```
channel = client_df[['id', 'channel_sales', 'churn']]
channel = channel.groupby([channel['channel_sales'], channel['churn']])['id'].count(
channel_churn = (channel.div(channel.sum(axis=1), axis=0) * 100).sort_values(by=[1],
plot_stacked_bars(channel_churn, 'Sales channel', rot_=30)
```



the churning customers are distributed over 5 different values for channel_sales. the value of MISSING has a churn rate of 7.6%. MISSING indicates a missing value and was added by the team when they were cleaning the dataset.

Consumption

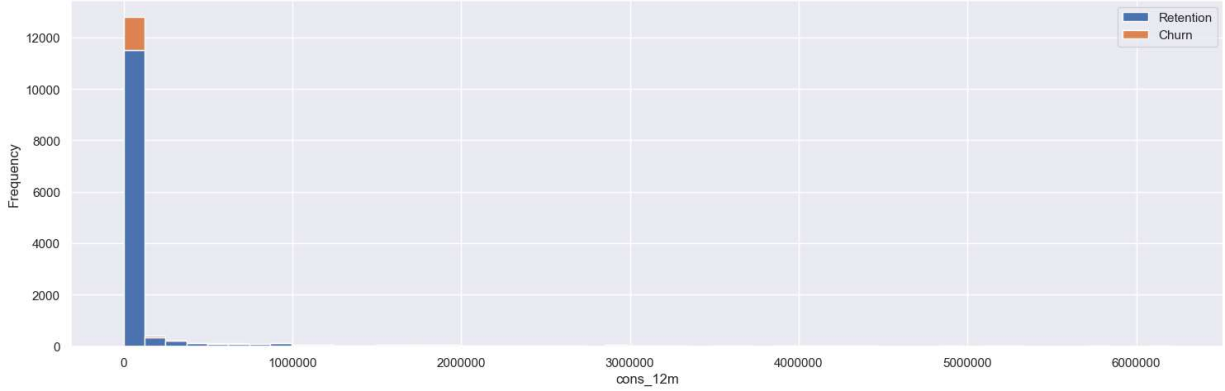
Let's see the distribution of the consumption in the last year and month. Since the consumption data is univariate, let's use histograms to visualize their distribution.

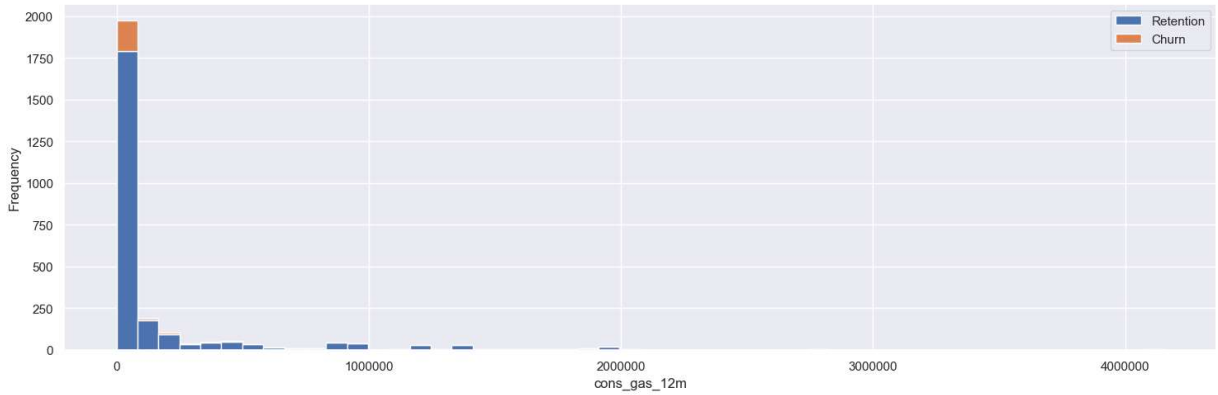
In [17]:
```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_c
consumption.head()
```

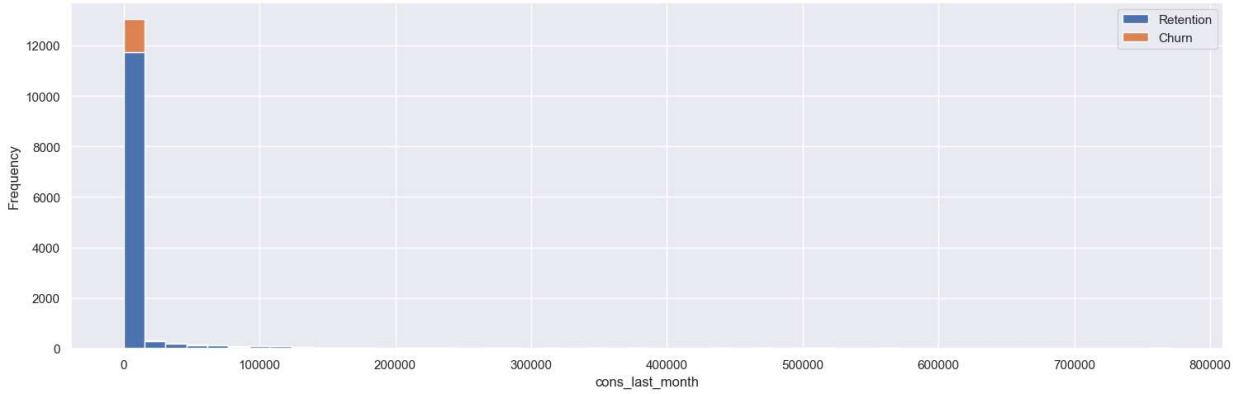
Out[17]:

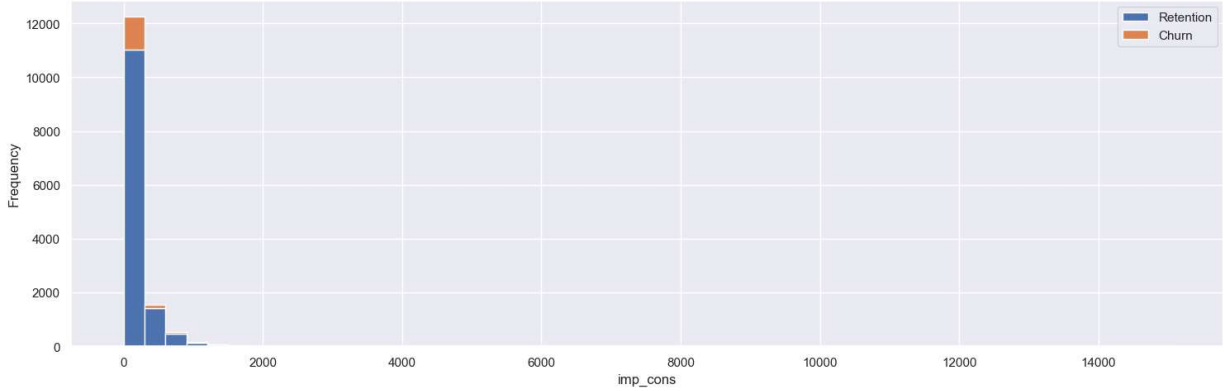
	id	cons_12m	cons_gas_12m	cons_last_month	imp_cons	has_gas
0	24011ae4ebbe3035111d65fa7c15bc57	0	54946	0	0.00	t
1	d29c2c54acc38ff3c0614d0a653813dd	4660	0	0	0.00	f
2	764c75f661154dac3a6c254cd082ea7d	544	0	0	0.00	f
3	bba03439a292a1e166f80264c16191cb	1584	0	0	0.00	f
4	149d57cf92fc41cf94415803a877cb4b	4425	0	526	52.32	f

In [18]:
```
def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distirbution in a stacked histogram of churned or retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "Churn":dataframe[dataframe["churn"]==1][column]})
    # Plot the histogram
    temp[["Retention","Churn"]].plot(kind='hist', bins=bins_, ax=ax, stacked=True)
    # X-axis label
    ax.set_xlabel(column)
    # Change the x-axis to plain style
    ax.ticklabel_format(style='plain', axis='x')
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])

plt.savefig("Distribution of the consumption.png", bbox_inches="tight")
```









The consumption data is clearly significantly positively skewed and has a very long right tail toward the upper values of the distribution. Outliers are most likely to be found in the values at the upper and lower ends of the distribution.

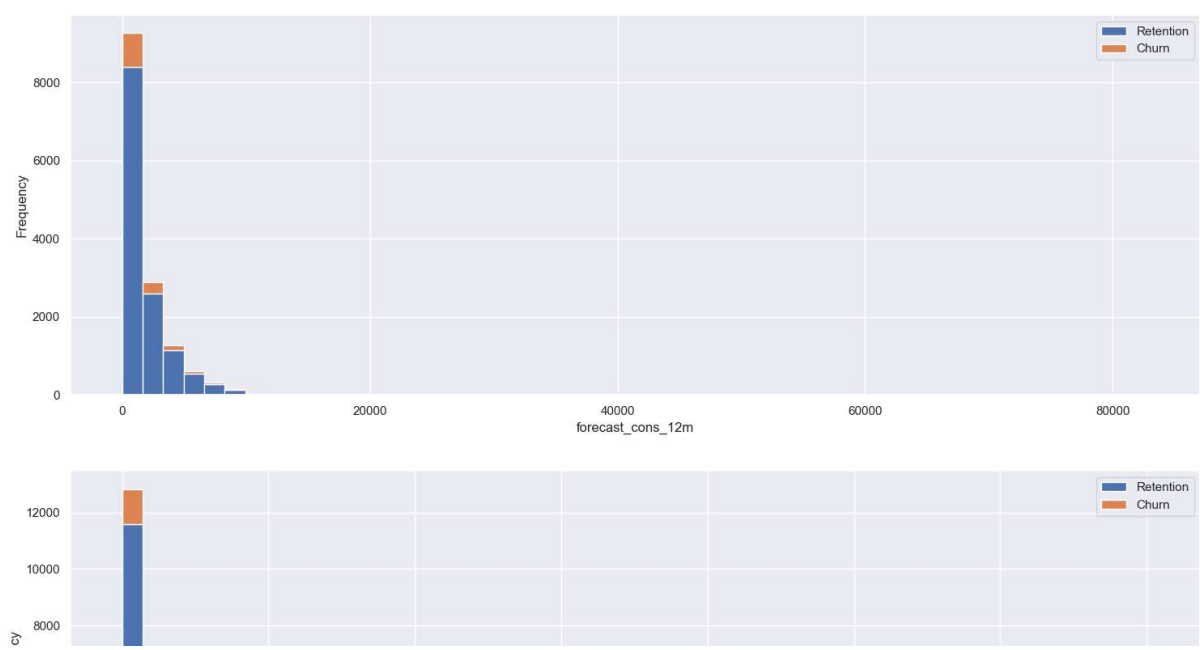
Forecast

In [19]:
```
forecast = client_df[["id", "forecast_cons_12m", "forecast_cons_year","forecast_disc
    "forecast_price_energy_off_peak","forecast_price_energy_peak", "forecast_price_p

forecast.head()
```

Out[19]:

	id	forecast_cons_12m	forecast_cons_year	forecast_discount_energy
0	24011ae4ebbe3035111d65fa7c15bc57	0.00	0	0.0
1	d29c2c54acc38ff3c0614d0a653813dd	189.95	0	0.0
2	764c75f661154dac3a6c254cd082ea7d	47.96	0	0.0
3	bba03439a292a1e166f80264c16191cb	240.04	0	0.0
4	149d57cf92fc41cf94415803a877cb4b	445.75	526	0.0

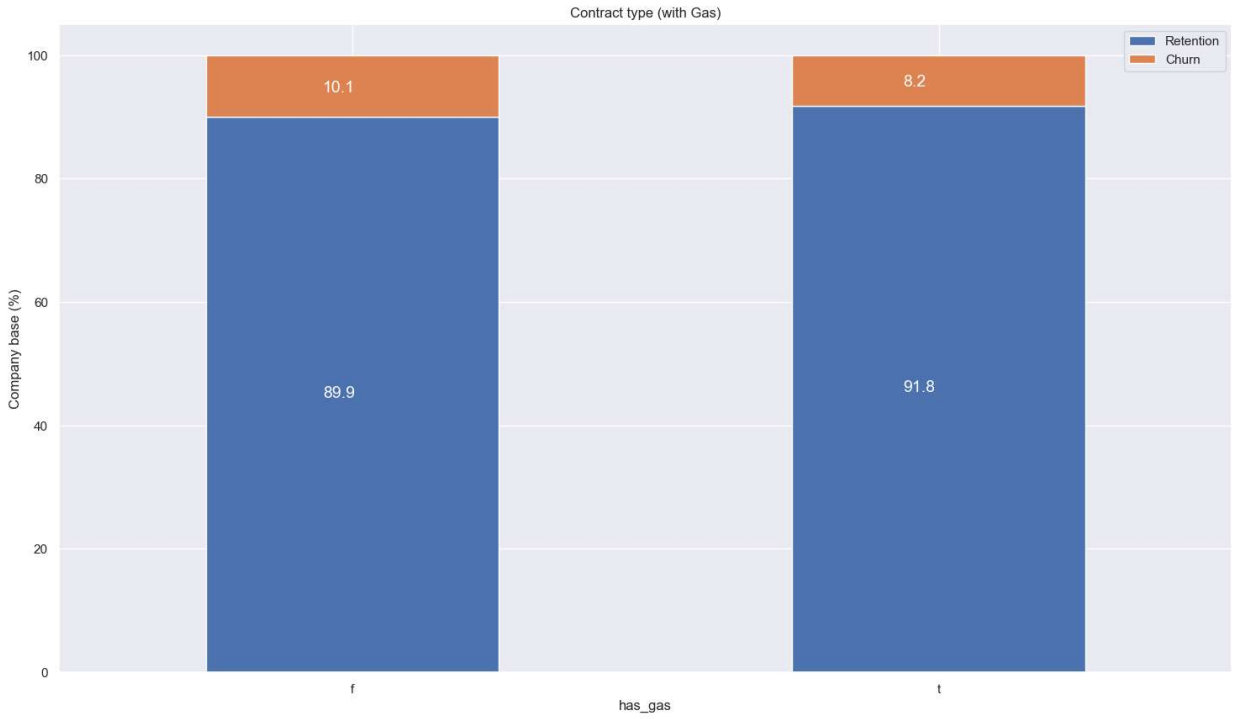
In [20]:
```
fig, axs = plt.subplots(nrows=7, figsize=(18,50))

# 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])

plt.savefig("View of Forecast.png", bbox_inches="tight")
```



We can see that several of the variables are significantly positively skewed, giving a very long tail for the higher values, similar to the consumption plots.

Contract type

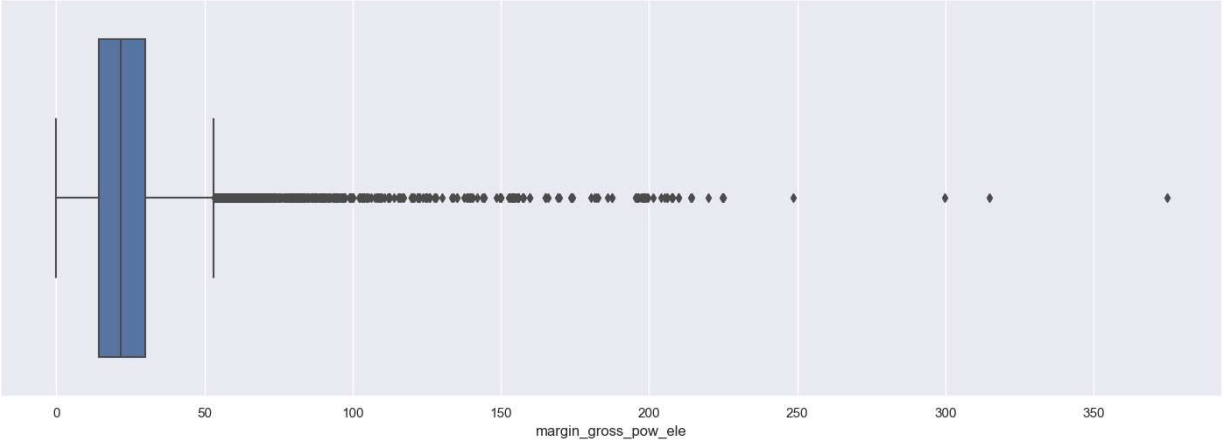
In [21]:
```
contract_type = client_df[['id', 'has_gas', 'churn']]
contract = contract_type.groupby([contract_type['churn'], contract_type['has_gas']])
contract_percentage = (contract.div(contract.sum(axis=1), axis=0) * 100).sort_values
plot_stacked_bars(contract_percentage, 'Contract type (with Gas)')
```

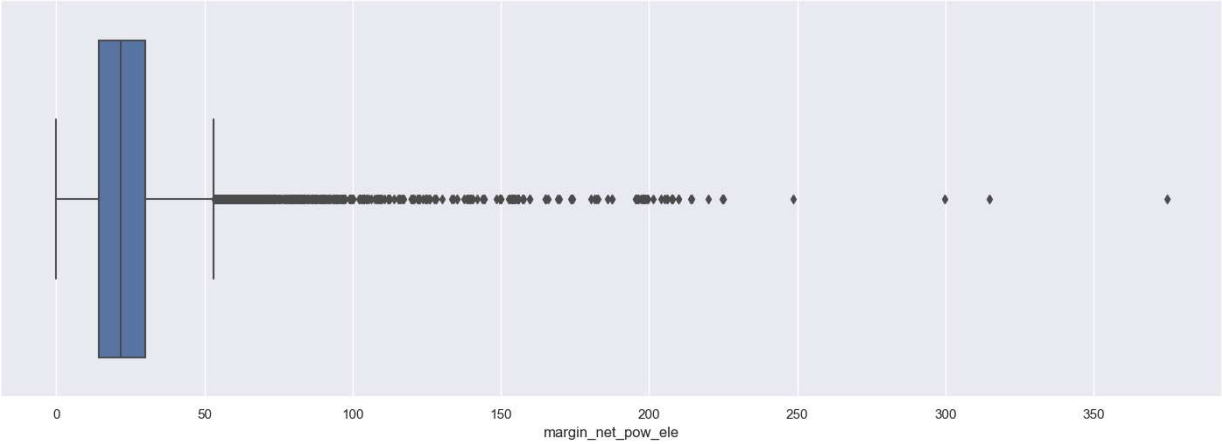


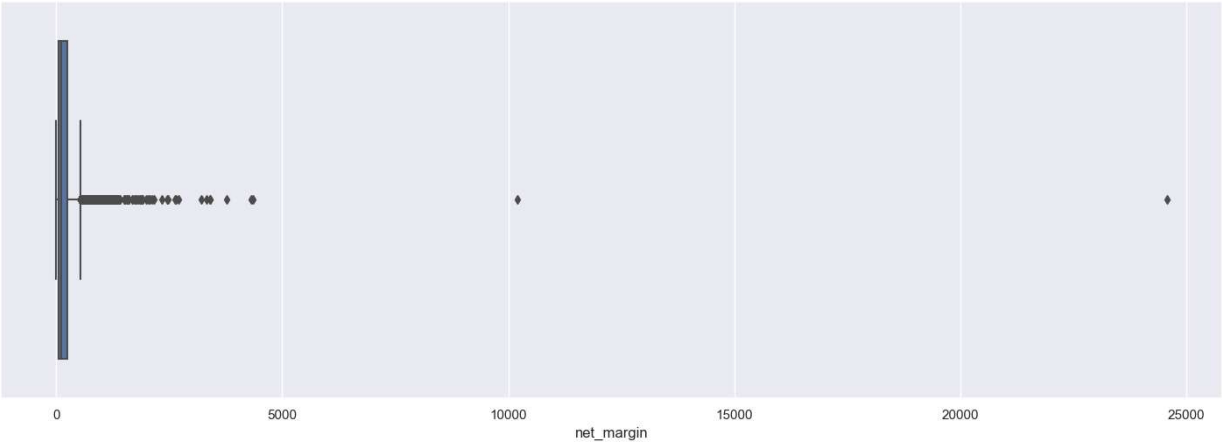
Margins

In [22]:
```
margin = client_df[['id', 'margin_gross_pow_ele', 'margin_net_pow_ele', 'net_margin'
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])
# 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.savefig("View of Margins.png", bbox_inches="tight")

plt.show()
```

```
C:\Users\lavender\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the only v
alid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\lavender\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the only v
alid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\lavender\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the only v
alid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```

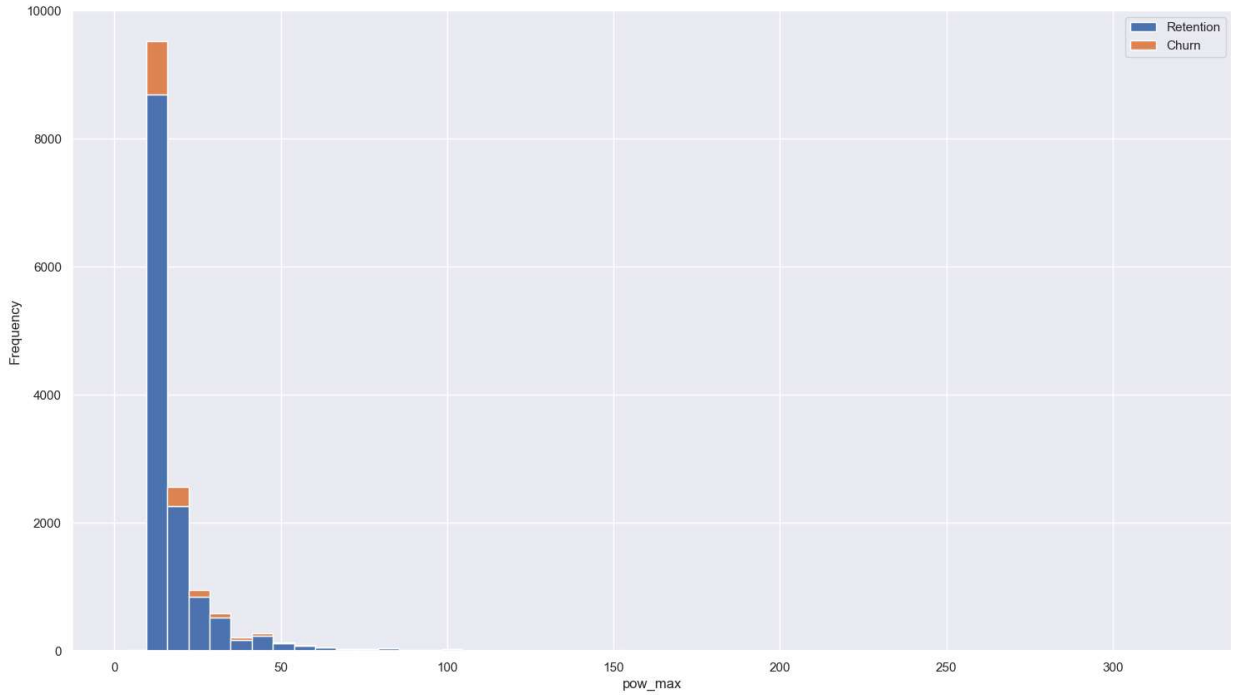

0
50
100
150
200
250
300
350
margin_gross_pow_ele


0
50
100
150
200
250
300
350
margin_net_pow_ele


0
5000
10000
15000
20000
25000
net_margin

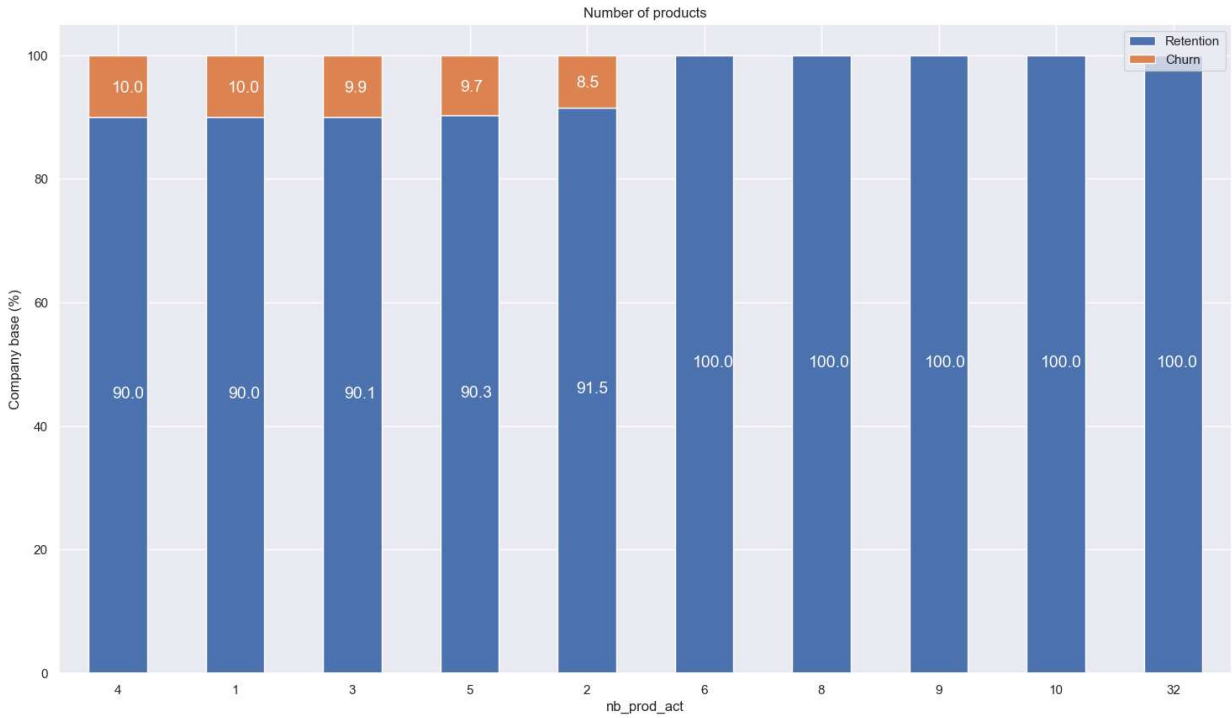
Subscribed power

In [23]:
```
power = client_df[['id', 'pow_max', 'churn']]
fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)

plt.savefig("View of Subscribed power.png", bbox_inches="tight")
```

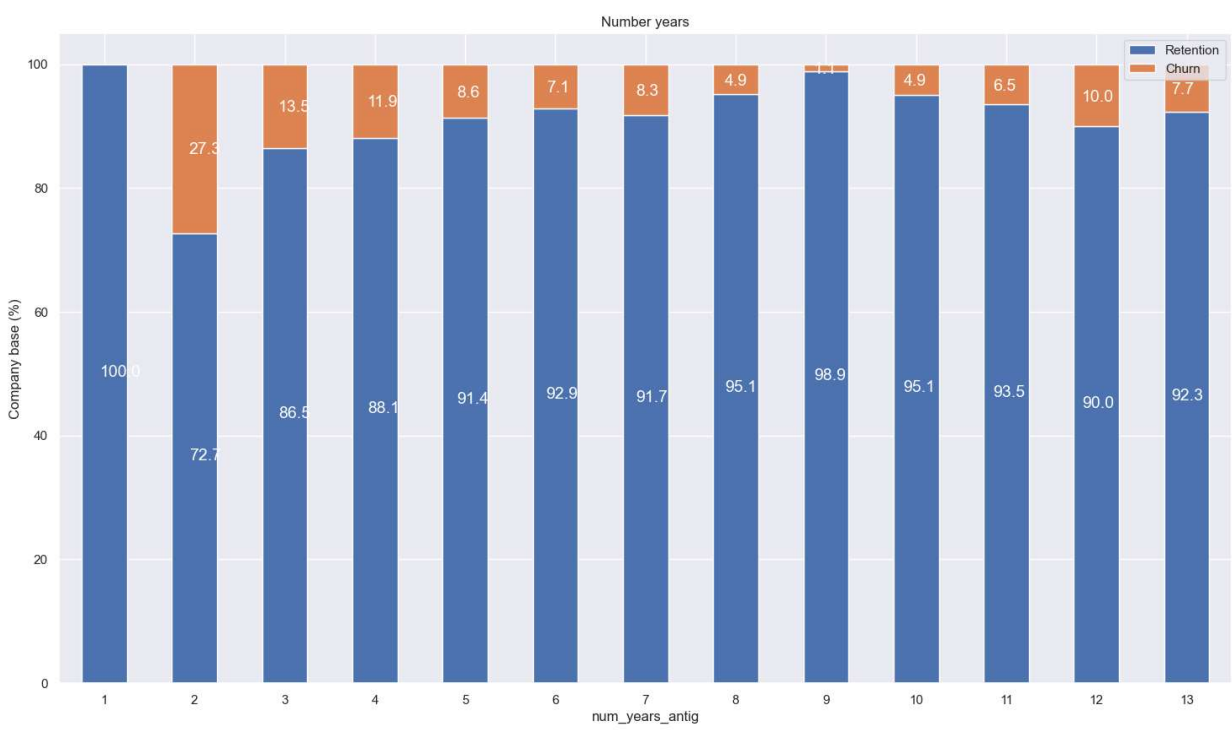


Other Columns

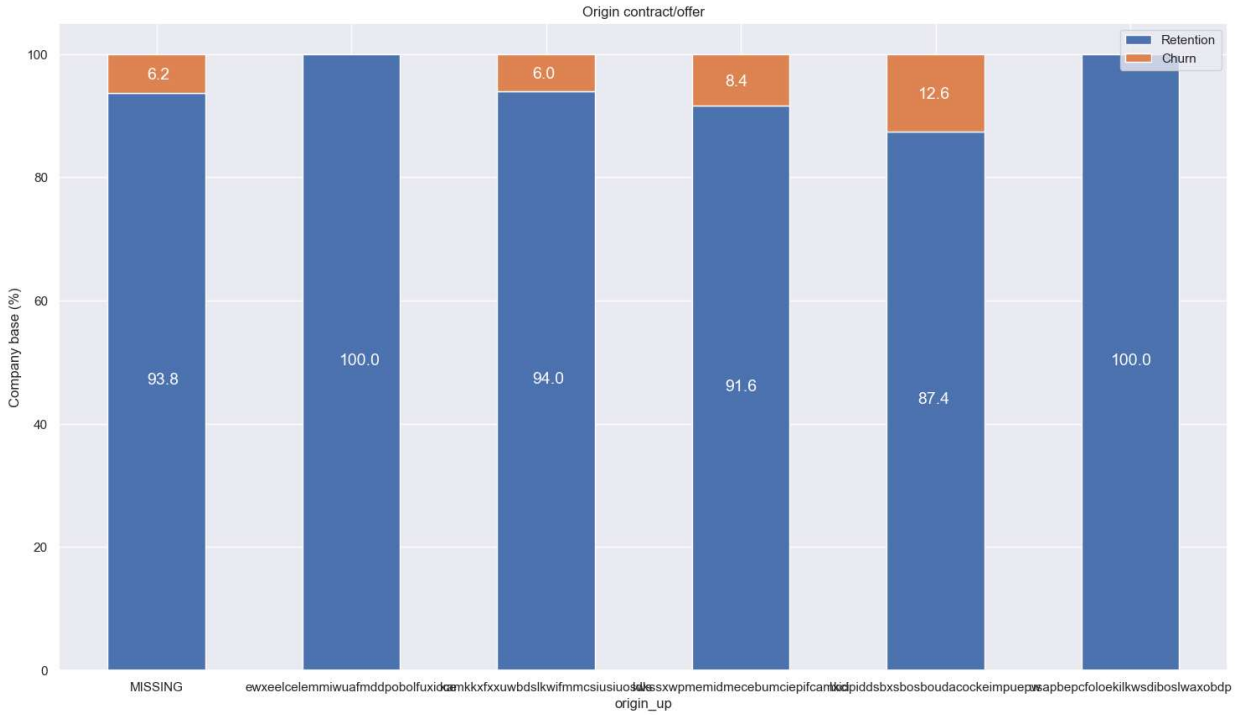
In [24]:
```
other_cols = client_df[['id', 'nb_prod_act', 'num_years_antig', 'origin_up', 'churn'
products = other_cols.groupby([other_cols["nb_prod_act"], other_cols["churn"]])["id"
products_percentage = (products.div(products.sum(axis=1), axis=0)*100).sort_values(b
plot_stacked_bars(products_percentage, "Number of products")
```



In [25]:
```
years_antig = other_cols.groupby([other_cols["num_years_antig"], other_cols["churn"]
years_antig_percentage = (years_antig.div(years_antig.sum(axis=1), axis=0)*100)
plot_stacked_bars(years_antig_percentage, "Number years")
```



In [26]:
```
origin = other_cols.groupby([other_cols["origin_up"], other_cols["churn"]])["id"].co
origin_percentage = (origin.div(origin.sum(axis=1), axis=0)*100)
plot_stacked_bars(origin_percentage, "Origin contract/offer")
```



Hypothesis Investigation

Since we have investigated the information, now is the ideal time to explore whether cost responsiveness has some effect on stir. First we want to characterize precisely exact thing is cost responsiveness.

In [27]:
```
# Transform date columns to datetime type
client_df["date_activ"] = pd.to_datetime(client_df["date_activ"], format='%Y-%m-%d')
client_df["date_end"] = pd.to_datetime(client_df["date_end"], format='%Y-%m-%d')
client_df["date_modif_prod"] = pd.to_datetime(client_df["date_modif_prod"], format='
client_df["date_renewal"] = pd.to_datetime(client_df["date_renewal"], format='%Y-%m-

price_df['price_date'] = pd.to_datetime(price_df['price_date'], format='%Y-%m-%d')

# Create mean average data
mean_year = price_df.groupby(['id']).mean().reset_index()
mean_6m = price_df[price_df['price_date'] > '2015-06-01'].groupby(['id']).mean().res
mean_3m = price_df[price_df['price_date'] > '2015-10-01'].groupby(['id']).mean().res

# Comnbine into single dataframe
mean_year = mean_year.rename(index=str, columns={"price_off_peak_var": "mean_year_pr
                                                 "price_peak_var": "mean_year_price_
                                                 "price_mid_peak_var": "mean_year_pr
                                                 "price_off_peak_fix": "mean_year_pr
                                                 "price_peak_fix": "mean_year_price_
                                                 "price_mid_peak_fix": "mean_year_pr
mean_year["mean_year_price_off_peak"] = mean_year["mean_year_price_off_peak_var"] + 
mean_year["mean_year_price_peak"] = mean_year["mean_year_price_peak_var"] + mean_yea
mean_year["mean_year_price_mid_peak"] = mean_year["mean_year_price_mid_peak_var"] + 

mean_6m = mean_6m.rename(index=str, columns={"price_off_peak_var": "mean_6m_price_of
                                             "price_peak_var": "mean_6m_price_peak_v
                                             "price_mid_peak_var": "mean_6m_price_mi
                                             "price_off_peak_fix": "mean_6m_price_of
                                             "price_peak_fix": "mean_6m_price_peak_f
                                             "price_mid_peak_fix": "mean_6m_price_mi

mean_6m["mean_6m_price_off_peak"] = mean_6m["mean_6m_price_off_peak_var"] + mean_6m[
mean_6m["mean_6m_price_peak"] = mean_6m["mean_6m_price_peak_var"] + mean_6m["mean_6m
mean_6m["mean_6m_price_mid_peak"] = mean_6m["mean_6m_price_mid_peak_var"] + mean_6m[

mean_3m = mean_3m.rename(index=str, columns={"price_off_peak_var": "mean_3m_price_of
                                             "price_peak_var": "mean_3m_price_peak_v
                                             "price_mid_peak_var": "mean_3m_price_mi
                                             "price_off_peak_fix": "mean_3m_price_of
                                             "price_peak_fix": "mean_3m_price_peak_f
                                             "price_mid_peak_fix": "mean_3m_price_mi

mean_3m["mean_3m_price_off_peak"] = mean_3m["mean_3m_price_off_peak_var"] + mean_3m[
mean_3m["mean_3m_price_peak"] = mean_3m["mean_3m_price_peak_var"] + mean_3m["mean_3m
mean_3m["mean_3m_price_mid_peak"] = mean_3m["mean_3m_price_mid_peak_var"] + mean_3m[

# Merge into 1 dataframe
price_features = pd.merge(mean_year, mean_6m, on='id')
price_features = pd.merge(price_features, mean_3m, on='id')
price_features.head()
```

Out[27]:

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

5 rows × 28 columns

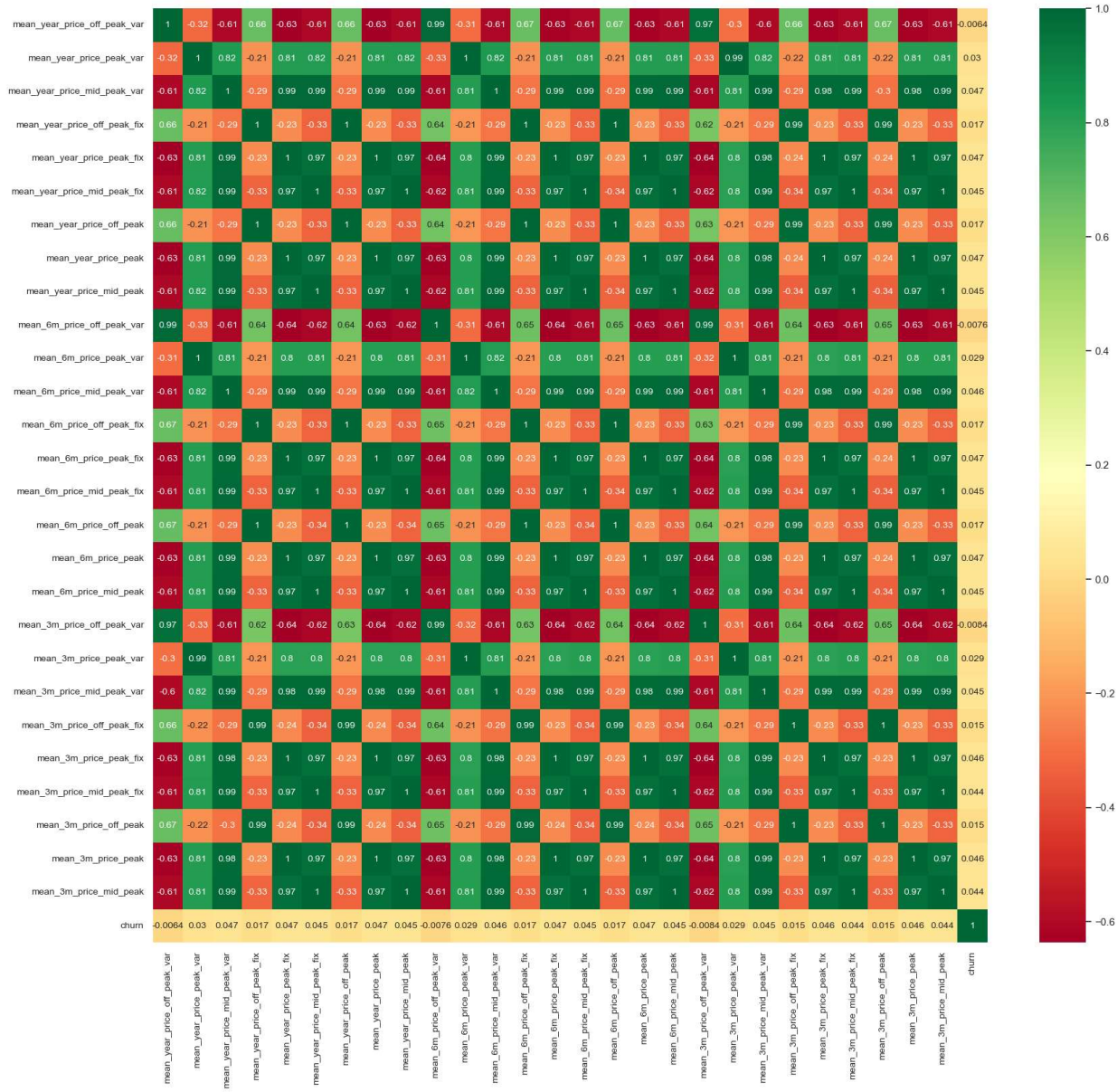
In [28]:
```
price_analysis = pd.merge(price_features, client_df[['id', 'churn']], on='id')
price_analysis.head()
```

Out[28]:

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

5 rows × 29 columns

In [29]:
```
corr = price_analysis.corr()
# Plot correlation
plt.figure(figsize=(20,18))
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values,
            annot = True, cmap='RdYlGn', annot_kws={'size':10})

# Axis ticks size
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.savefig("View of Correlation.png", bbox_inches="tight")
plt.show()
```



the correlation plot reveals a higher degree of correlation with other price sensitivity variables. This suggests that price sensitivity and churn have a shaky linear relationship. This shows that we might need to develop the feature differently if we want price sensivity to be a significant factor in predicting churn.

In [30]:
```
merged_data = pd.merge(client_df.drop(columns=['churn']), price_analysis, on='id')
merged_data.head()
```

Out[30]:

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

5 rows × 53 columns

In [ ]: