eda_model_answer

October 28, 2021

1 Exploratory Data Analysis

---

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

1.1 1. Import packages

[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)
```

1.2 2. 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

[3]:
```
client_df = pd.read_csv('./client_data.csv')
price_df = pd.read_csv('./price_data.csv')
```

Let's look at the first 3 rows of both dataframes to see what the data looks like

[4]:
```
client_df.head(3)
```

[4]:
```
                                 id                     channel_sales  \
0  24011ae4ebbe3035111d65fa7c15bc57  foosdfpfkusacimwkcsosbicdxkicaua
```

```
1  d29c2c54acc38ff3c0614d0a653813dd                           MISSING
2  764c75f661154dac3a6c254cd082ea7d  foosdfpfkusacimwkcsosbicdxkicaua

   cons_12m  cons_gas_12m  cons_last_month  date_activ    date_end  \
0         0         54946                0  2013-06-15  2016-06-15
1      4660             0                0  2009-08-21  2016-08-30
2       544             0                0  2010-04-16  2016-04-16

  date_modif_prod date_renewal  forecast_cons_12m  …  has_gas  imp_cons  \
0      2015-11-01   2015-06-23               0.00  …        t       0.0
1      2009-08-21   2015-08-31             189.95  …        f       0.0
2      2010-04-16   2015-04-17              47.96  …        f       0.0

   margin_gross_pow_ele  margin_net_pow_ele  nb_prod_act  net_margin  \
0                 25.44               25.44            2      678.99
1                 16.38               16.38            1       18.89
2                 28.60               28.60            1        6.60

  num_years_antig                       origin_up  pow_max  churn
0               3  lxidpiddsbxsbosboudacockeimpuepw   43.648      1
1               6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.800      0
2               6  kamkkxfxxuwbdslkwifmmcsiusiuosws   13.856      0

[3 rows x 26 columns]
```

With the client data, we have a mix of numeric and categorical data, which we will need to transform before modelling later

[5]:
```python
price_df.head(3)
```

[5]:
```
                                 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

   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
```

With the price data, it is purely numeric data but we can see a lot of zeros

---

1.3 3. Descriptive statistics of data

1.3.1 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.

[6]:
```
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_p1  14606 non-null  float64
 14  forecast_price_energy_p2  14606 non-null  float64
 15  forecast_price_pow_p1     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
```

[7]:
```
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_p1_var  193002 non-null  float64
 3   price_p2_var  193002 non-null  float64
 4   price_p3_var  193002 non-null  float64
```

```
 5   price_p1_fix  193002 non-null  float64
 6   price_p2_fix  193002 non-null  float64
 7   price_p3_fix  193002 non-null  float64
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

You can see that all of the `datetime` related columns are not currently in datetime format. We will need to convert these later.

1.3.2 Statistics

Now let's look at some statistics about the datasets

[8]:
```
client_df.describe()
```

[8]:
```
           cons_12m  cons_gas_12m  cons_last_month  forecast_cons_12m  \
count  1.460600e+04  1.460600e+04     14606.000000       14606.000000
mean   1.592203e+05  2.809238e+04     16090.269752        1868.614880
std    5.734653e+05  1.629731e+05     64364.196422        2387.571531
min    0.000000e+00  0.000000e+00         0.000000           0.000000
25%    5.674750e+03  0.000000e+00         0.000000         494.995000
50%    1.411550e+04  0.000000e+00       792.500000        1112.875000
75%    4.076375e+04  0.000000e+00      3383.000000        2401.790000
max    6.207104e+06  4.154590e+06    771203.000000       82902.830000

       forecast_cons_year  forecast_discount_energy  forecast_meter_rent_12m  \
count        14606.000000              14606.000000             14606.000000
mean          1399.762906                  0.966726                63.086871
std           3247.786255                  5.108289                66.165783
min              0.000000                  0.000000                 0.000000
25%              0.000000                  0.000000                16.180000
50%            314.000000                  0.000000                18.795000
75%           1745.750000                  0.000000               131.030000
max         175375.000000                 30.000000               599.310000

       forecast_price_energy_p1  forecast_price_energy_p2  \
count              14606.000000              14606.000000
mean                   0.137283                  0.050491
std                    0.024623                  0.049037
min                    0.000000                  0.000000
25%                    0.116340                  0.000000
50%                    0.143166                  0.084138
75%                    0.146348                  0.098837
max                    0.273963                  0.195975

       forecast_price_pow_p1      imp_cons  margin_gross_pow_ele  \
count           14606.000000  14606.000000          14606.000000
mean               43.130056    152.786896             24.565121
```

std	4.485988	341.369366	20.231172
min	0.000000	0.000000	0.000000
25%	40.606701	0.000000	14.280000
50%	44.311378	37.395000	21.640000
75%	44.311378	193.980000	29.880000
max	59.266378	15042.790000	374.640000

	margin_net_pow_ele	nb_prod_act	net_margin	num_years_antig \
count	14606.000000	14606.000000	14606.000000	14606.000000
mean	24.562517	1.292346	189.264522	4.997809
std	20.230280	0.709774	311.798130	1.611749
min	0.000000	1.000000	0.000000	1.000000
25%	14.280000	1.000000	50.712500	4.000000
50%	21.640000	1.000000	112.530000	5.000000
75%	29.880000	1.000000	243.097500	6.000000
max	374.640000	32.000000	24570.650000	13.000000

	pow_max	churn
count	14606.000000	14606.000000
mean	18.135136	0.097152
std	13.534743	0.296175
min	3.300000	0.000000
25%	12.500000	0.000000
50%	13.856000	0.000000
75%	19.172500	0.000000
max	320.000000	1.000000

The describe method gives us a lot of information about the client data. The key point to take away from this is that we have highly skewed data, as exhibited by the percentile values.

[9]:
```
price_df.describe()
```

[9]:

	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
```

Overall the price data looks good.

1.4 3. Data visualization

Now let's dive a bit deeper into the dataframes

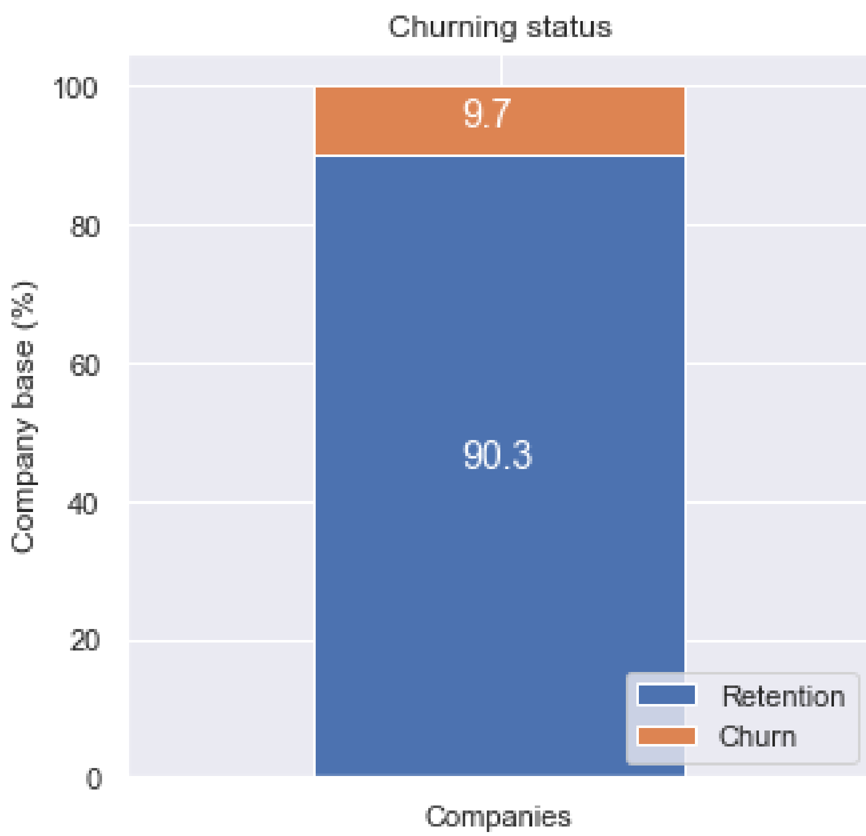
[10]:
```
def plot_stacked_bars(dataframe, title_, size_=(18, 10), rot_=0, legend_="upper␣
↪right"):
    """
    Plot stacked bars with annotations
    """
    ax = dataframe.plot(
        kind="bar",
        stacked=True,
        figsize=size_,
        rot=rot_,
        title=title_
    )

    # 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
        )
```

1.4.1 Churn

[11]:
```
churn = client_df[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
```

[12]:
```
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5),␣
↪legend_="lower right")
```

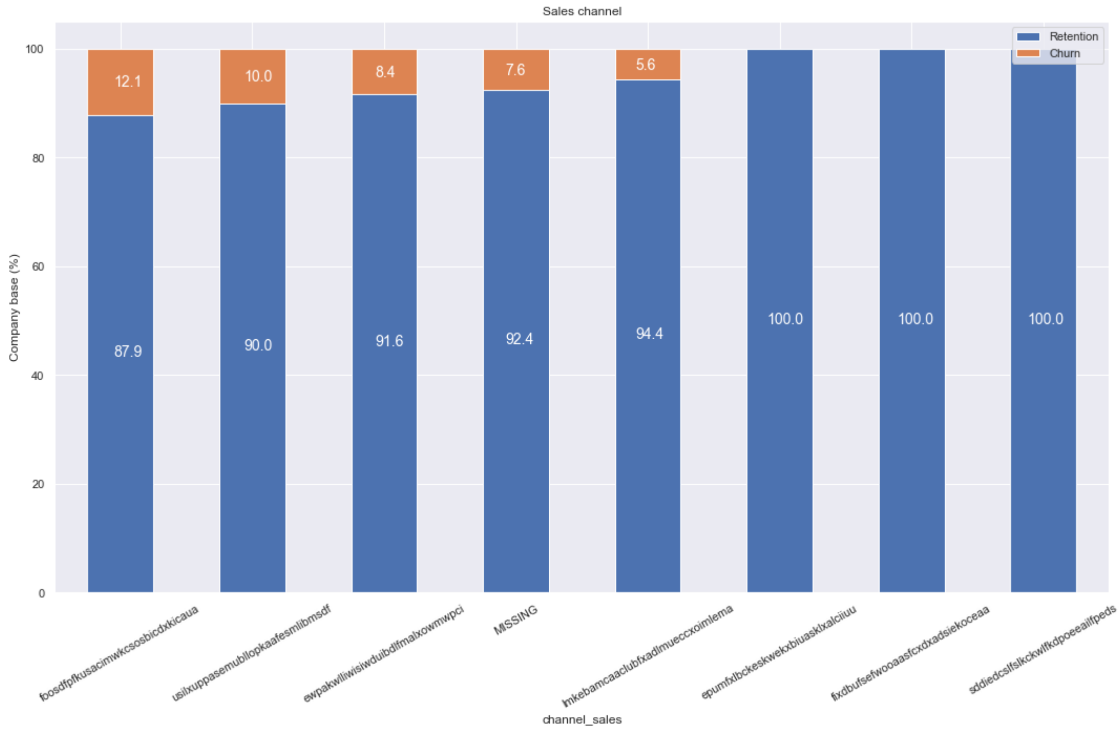


About 10% of the total customers have churned. (This sounds about right)

1.4.2 Sales channel

[13]:
```python
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)
```

[14]:
```python
plot_stacked_bars(channel_churn, 'Sales channel', rot_=30)
```



Interestingly, the churning customers are distributed over 5 different values for `channel_sales`. As well as this, 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. This feature could be an important feature when it comes to building our model.

1.4.3 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.

[15]:
```python
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month',␣
↪'imp_cons', 'has_gas', 'churn']]
```

[16]:
```python
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')
```

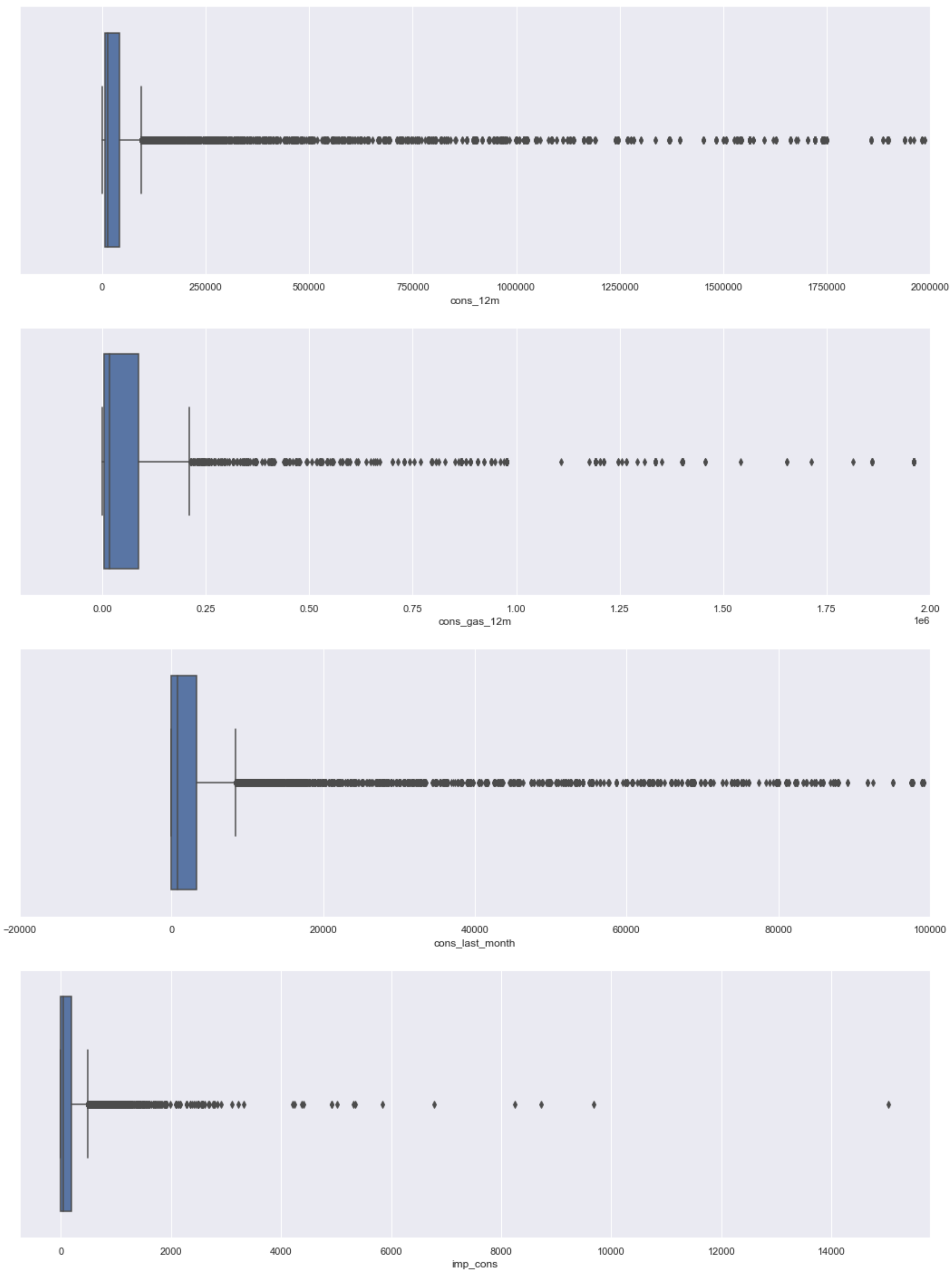
[17]:
```
fig, axs = plt.subplots(nrows=4, figsize=(18, 25))

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])
```

Clearly, the consumption data is highly positively skewed, presenting a very long right-tail towards the higher values of the distribution. 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.

[18]:
```
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()
```

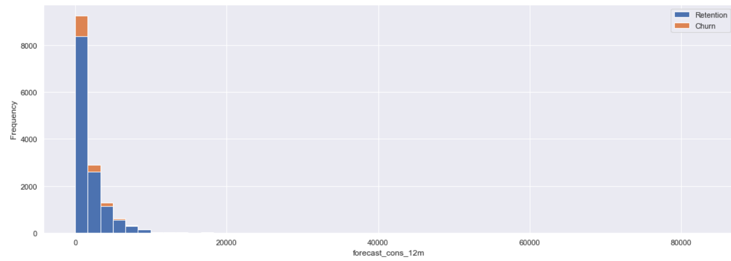


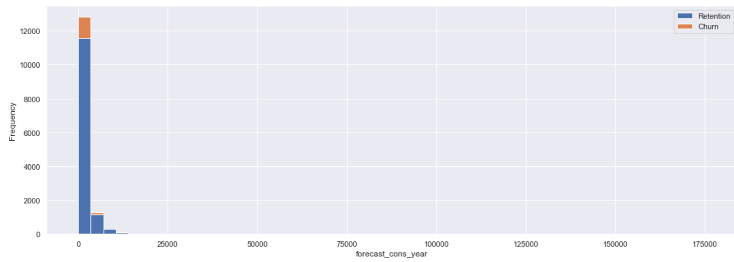
We will deal with skewness and outliers during feature engineering in the next exercise.

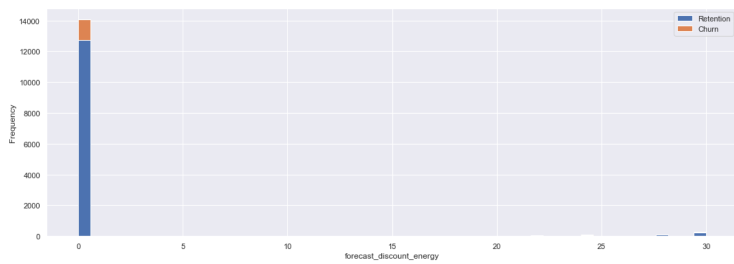
1.4.4 Forecast

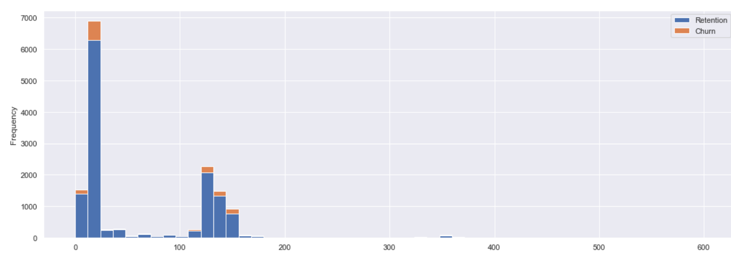
[19]:
```
forecast = client_df[
    ["id", "forecast_cons_12m",
    "forecast_cons_year","forecast_discount_energy","forecast_meter_rent_12m",
    "forecast_price_energy_p1","forecast_price_energy_p2",
    "forecast_price_pow_p1","churn"
    ]
]
```

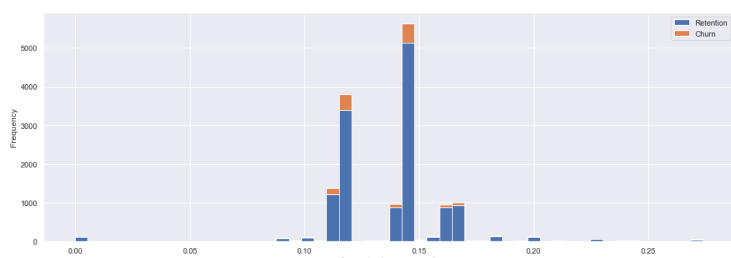
[20]:
```
fig, axs = plt.subplots(nrows=7, figsize=(18,50))

# 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_p1", axs[4])
plot_distribution(client_df, "forecast_price_energy_p2", axs[5])
plot_distribution(client_df, "forecast_price_pow_p1", axs[6])
```

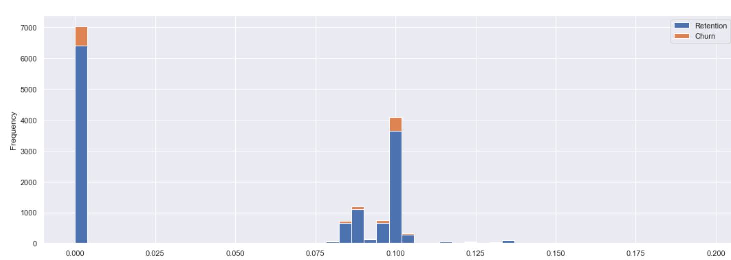

Retention
Churn
Frequency
0 2000 4000 6000 8000
0 20000 40000 60000 80000
forecast_cons_12m

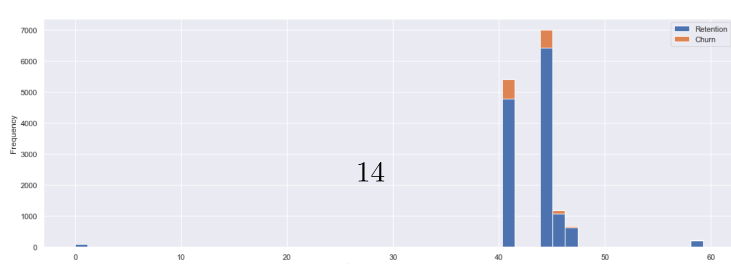

Retention
Churn
Frequency
0 2000 4000 6000 8000 10000 12000
0 25000 50000 75000 100000 125000 150000 175000
forecast_cons_year


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


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


Retention
Churn
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

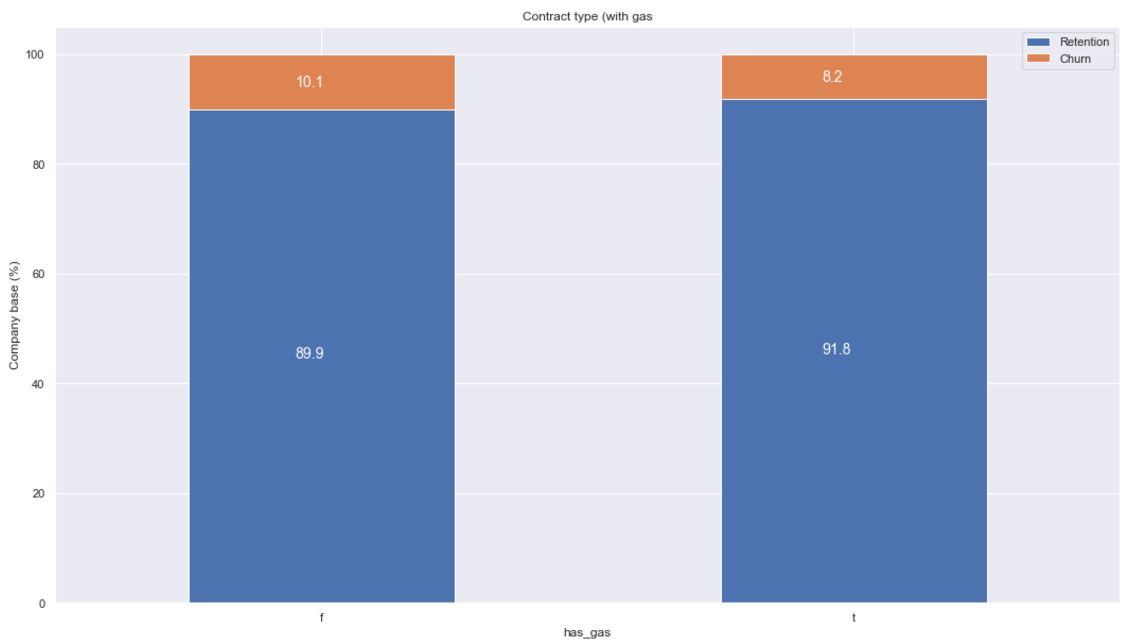

Retention
Churn
Frequency
0 1000 2000 3000 4000 5000 6000 7000
0 10 20 30 40 50 60
forecast_price_pow_p1

Similarly to the consumption plots, we can observe that a lot of the variables are highly positively skewed, creating a very long tail for the higher values. We will make some transformations during the next exercise to correct for this skewness.

1.4.5 Contract type

[21]:
```
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)
```

[22]:
```
plot_stacked_bars(contract_percentage, 'Contract type (with gas')
```

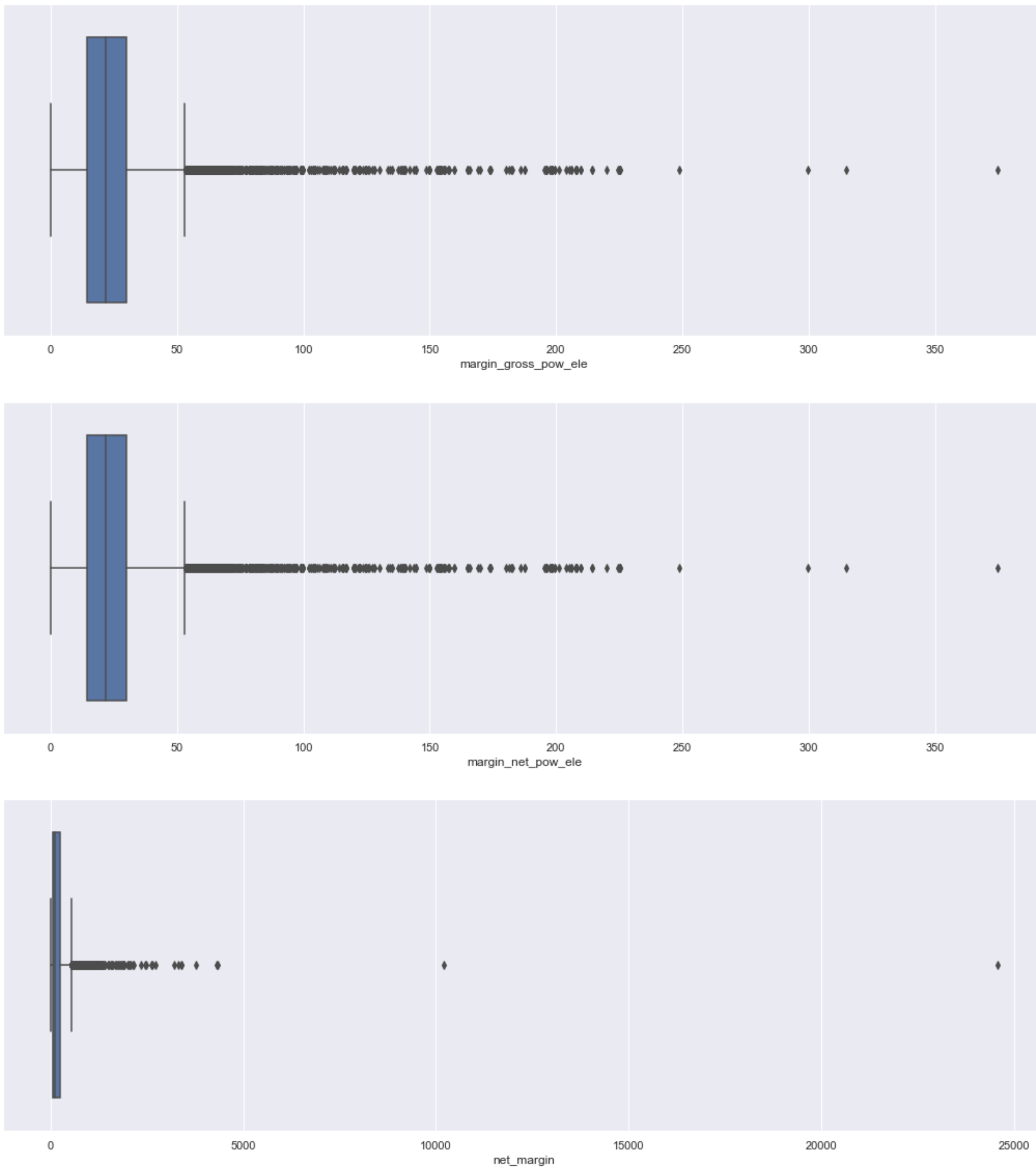


1.4.6 Margins

[23]:
```
margin = client_df[['id', 'margin_gross_pow_ele', 'margin_net_pow_ele',␣
↪'net_margin']]
```

[24]:
```
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.show()
```

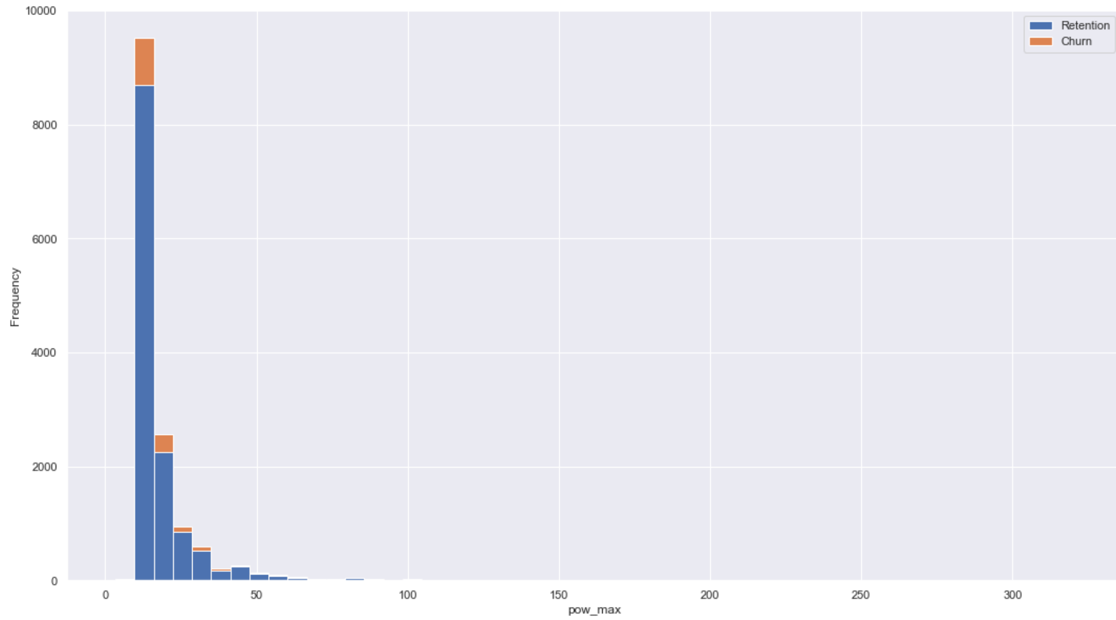


We can see some outliers here as well which we will deal with in the next exercise.

1.4.7 Subscribed power

[25]:
```
power = client_df[['id', 'pow_max', 'churn']]
```

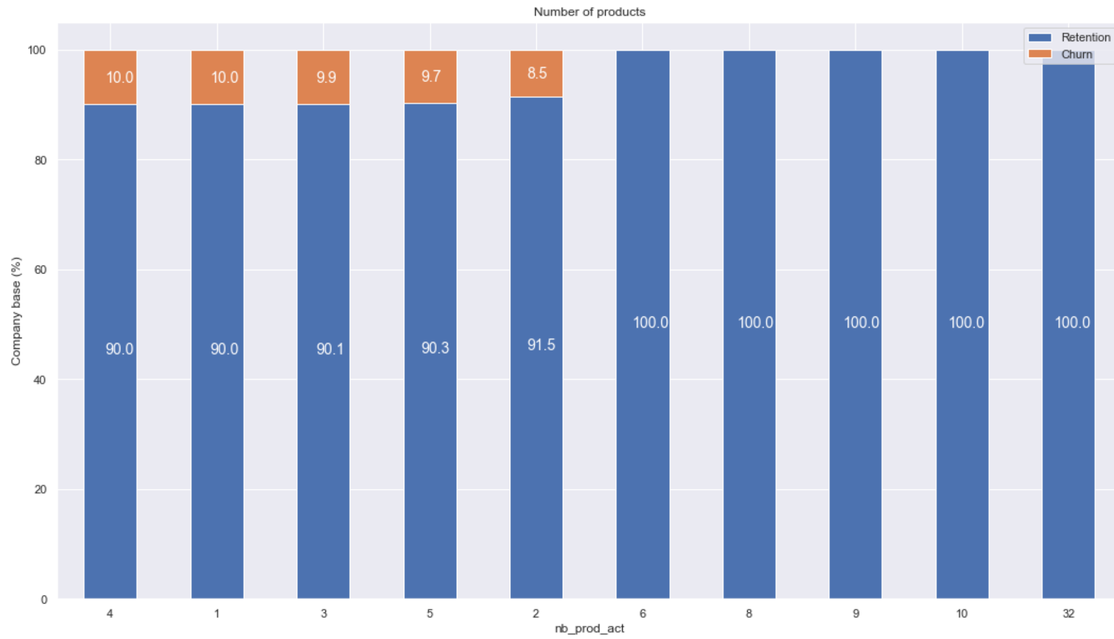
[26]:
```
fig, axs = plt.subplots(nrows=1, figsize=(18, 10))
plot_distribution(power, 'pow_max', axs)
```



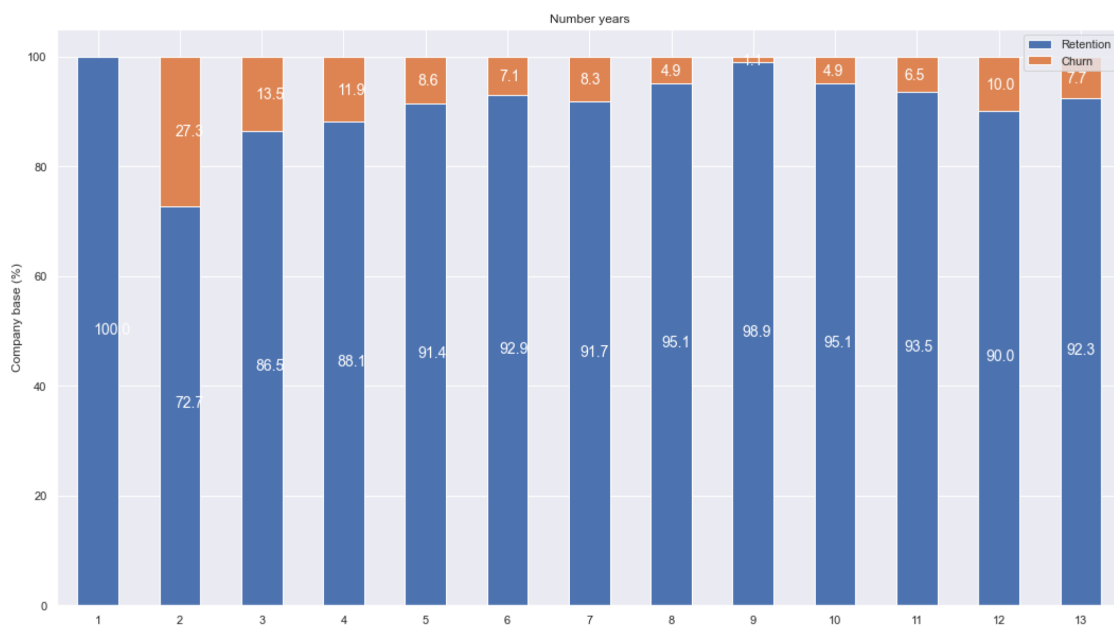
1.4.8 Other columns

[27]:
```
others = client_df[['id', 'nb_prod_act', 'num_years_antig', 'origin_up',␣
↪'churn']]
products = others.groupby([others["nb_prod_act"],others["churn"]])["id"].
↪count().unstack(level=1)
products_percentage = (products.div(products.sum(axis=1), axis=0)*100).
↪sort_values(by=[1], ascending=False)
```

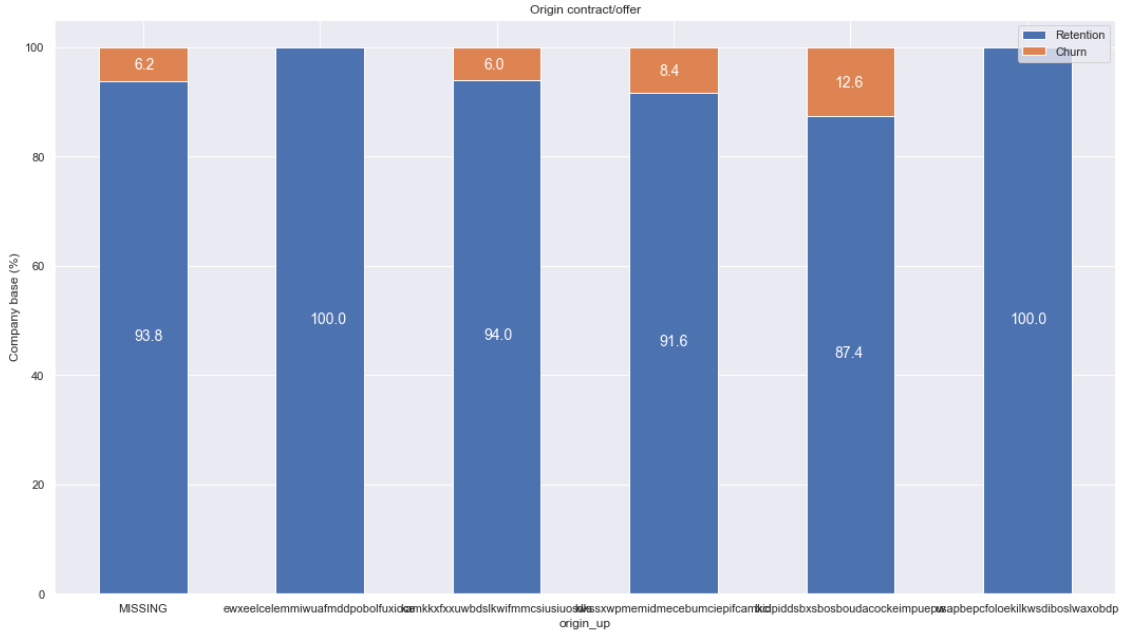
[28]:
```
plot_stacked_bars(products_percentage, "Number of products")
```



[29]:
```
years_antig = others.groupby([others["num_years_antig"],others["churn"]])["id"].
    ↪count().unstack(level=1)
years_antig_percentage = (years_antig.div(years_antig.sum(axis=1), axis=0)*100)
plot_stacked_bars(years_antig_percentage, "Number years")
```



```
[30]: origin = others.groupby([others["origin_up"],others["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")
```



1.5 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 d`

```
[81]: # 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='%Y-%m-%d')
      client_df["date_renewal"] = pd.to_datetime(client_df["date_renewal"],␣
      ↪format='%Y-%m-%d')
      price_df['price_date'] = pd.to_datetime(price_df['price_date'],␣
      ↪format='%Y-%m-%d')
```

```
[82]: # Create yearly sensitivity features
      var_year = price_df.groupby(['id', 'price_date']).mean().groupby(['id']).var().
      ↪reset_index()
```

```
# Create last 6 months sensitivity features
var_6m = price_df[
    price_df['price_date'] > '2015-06-01'
].groupby(['id', 'price_date']).mean().groupby(['id']).var().reset_index()

# Rename columns
var_year = var_year.rename(
    columns={
        "price_p1_var": "var_year_price_p1_var",
        "price_p2_var": "var_year_price_p2_var",
        "price_p3_var": "var_year_price_p3_var",
        "price_p1_fix": "var_year_price_p1_fix",
        "price_p2_fix": "var_year_price_p2_fix",
        "price_p3_fix": "var_year_price_p3_fix"
    }
)
var_year["var_year_price_p1"] = var_year["var_year_price_p1_var"] +␣
↪var_year["var_year_price_p1_fix"]
var_year["var_year_price_p2"] = var_year["var_year_price_p2_var"] +␣
↪var_year["var_year_price_p2_fix"]
var_year["var_year_price_p3"] = var_year["var_year_price_p3_var"] +␣
↪var_year["var_year_price_p3_fix"]

var_6m = var_6m.rename(
    columns={
        "price_p1_var": "var_6m_price_p1_var",
        "price_p2_var": "var_6m_price_p2_var",
        "price_p3_var": "var_6m_price_p3_var",
        "price_p1_fix": "var_6m_price_p1_fix",
        "price_p2_fix": "var_6m_price_p2_fix",
        "price_p3_fix": "var_6m_price_p3_fix"
    }
)
var_6m["var_6m_price_p1"] = var_6m["var_6m_price_p1_var"] +␣
↪var_6m["var_6m_price_p1_fix"]
var_6m["var_6m_price_p2"] = var_6m["var_6m_price_p2_var"] +␣
↪var_6m["var_6m_price_p2_fix"]
var_6m["var_6m_price_p3"] = var_6m["var_6m_price_p3_var"] +␣
↪var_6m["var_6m_price_p3_fix"]

# Merge into 1 dataframe
price_features = pd.merge(var_year, var_6m, on='id')
```

[83]:
```
price_features.head()
```

[83]:

```
                                 id  var_year_price_p1_var  \
0  0002203ffbb812588b632b9e628cc38d              0.000016
1  0004351ebdd665e6ee664792efc4fd13              0.000005
2  0010bcc39e42b3c2131ed2ce55246e3c              0.000676
3  0010ee3855fdea87602a5b7aba8e42de              0.000025
4  00114d74e963e47177db89bc70108537              0.000005

   var_year_price_p2_var  var_year_price_p3_var  var_year_price_p1_fix  \
0               0.000004           1.871602e-06           4.021438e-03
1               0.000000           0.000000e+00           7.661891e-03
2               0.000000           0.000000e+00           5.965909e-01
3               0.000007           1.627620e-07           7.238536e-03
4               0.000000           0.000000e+00           3.490909e-13

   var_year_price_p2_fix  var_year_price_p3_fix  var_year_price_p1  \
0               0.001448               0.000643           0.004037
1               0.000000               0.000000           0.007667
2               0.000000               0.000000           0.597267
3               0.002606               0.001158           0.007264
4               0.000000               0.000000           0.000005

   var_year_price_p2  var_year_price_p3  var_6m_price_p1_var  \
0           0.001452           0.000645             0.000011
1           0.000000           0.000000             0.000003
2           0.000000           0.000000             0.000003
3           0.002613           0.001158             0.000011
4           0.000000           0.000000             0.000003

   var_6m_price_p2_var  var_6m_price_p3_var  var_6m_price_p1_fix  \
0             0.000003         4.860000e-10                  0.0
1             0.000000         0.000000e+00                  0.0
2             0.000000         0.000000e+00                  0.0
3             0.000003         4.860000e-10                  0.0
4             0.000000         0.000000e+00                  0.0

   var_6m_price_p2_fix  var_6m_price_p3_fix  var_6m_price_p1  var_6m_price_p2  \
0                  0.0                  0.0         0.000011         0.000003
1                  0.0                  0.0         0.000003         0.000000
2                  0.0                  0.0         0.000003         0.000000
3                  0.0                  0.0         0.000011         0.000003
4                  0.0                  0.0         0.000003         0.000000

   var_6m_price_p3
0     4.860000e-10
1     0.000000e+00
2     0.000000e+00
3     4.860000e-10
```

```
4     0.000000e+00
```

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

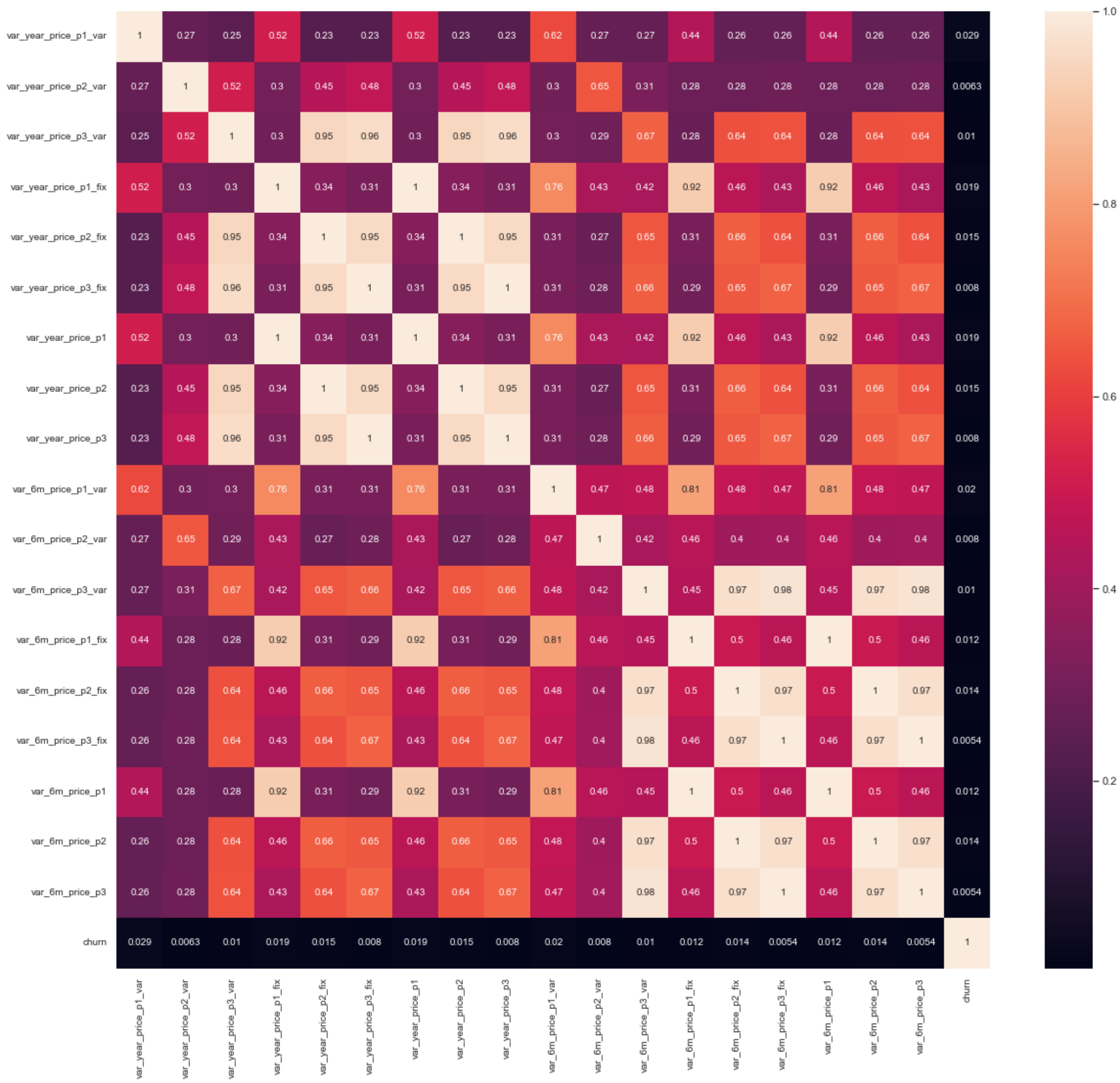
[84]:
```python
price_analysis = pd.merge(price_features, client_df[['id', 'churn']], on='id')
price_analysis.head()
```

[84]:
```
                                 id  var_year_price_p1_var  \
0  0002203ffbb812588b632b9e628cc38d               0.000016
1  0004351ebdd665e6ee664792efc4fd13               0.000005
2  0010bcc39e42b3c2131ed2ce55246e3c               0.000676
3  00114d74e963e47177db89bc70108537               0.000005
4  0013f326a839a2f6ad87a1859952d227               0.000016

   var_year_price_p2_var  var_year_price_p3_var  var_year_price_p1_fix  \
0               0.000004               0.000002           4.021438e-03
1               0.000000               0.000000           7.661891e-03
2               0.000000               0.000000           5.965909e-01
3               0.000000               0.000000           3.490909e-13
4               0.000004               0.000002           0.000000e+00

   var_year_price_p2_fix  var_year_price_p3_fix  var_year_price_p1  \
0               0.001448               0.000643           0.004037
1               0.000000               0.000000           0.007667
2               0.000000               0.000000           0.597267
3               0.000000               0.000000           0.000005
4               0.000000               0.000000           0.000016

   var_year_price_p2  var_year_price_p3  var_6m_price_p1_var  \
0           0.001452           0.000645             0.000011
1           0.000000           0.000000             0.000003
2           0.000000           0.000000             0.000003
3           0.000000           0.000000             0.000003
4           0.000004           0.000002             0.000011

   var_6m_price_p2_var  var_6m_price_p3_var  var_6m_price_p1_fix  \
0             0.000003         4.860000e-10                  0.0
1             0.000000         0.000000e+00                  0.0
2             0.000000         0.000000e+00                  0.0
3             0.000000         0.000000e+00                  0.0
4             0.000003         4.860000e-10                  0.0

   var_6m_price_p2_fix  var_6m_price_p3_fix  var_6m_price_p1  var_6m_price_p2  \
0                  0.0                  0.0         0.000011         0.000003
1                  0.0                  0.0         0.000003         0.000000
2                  0.0                  0.0         0.000003         0.000000
3                  0.0                  0.0         0.000003         0.000000
4                  0.0                  0.0         0.000011         0.000003
```

```
   var_6m_price_p3  churn
0     4.860000e-10      0
1     0.000000e+00      0
2     0.000000e+00      0
3     0.000000e+00      0
4     4.860000e-10      0
```

[85]:

```
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()
```



From the correlation plot, it shows that the price sensitivity features a high inter-correlation with each other, but 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 features differently.

[86]:
```
merged_data = pd.merge(client_df.drop(columns=['churn']), price_analysis,␣
↪on='id')
```

[87]:
```
merged_data.head()
```

[87]:
```
                                 id                     channel_sales  \
0  24011ae4ebbe3035111d65fa7c15bc57  foosdfpfkusacimwkcsosbicdxkicaua
1  d29c2c54acc38ff3c0614d0a653813dd                           MISSING
2  764c75f661154dac3a6c254cd082ea7d  foosdfpfkusacimwkcsosbicdxkicaua
```

```
3  bba03439a292a1e166f80264c16191cb  lmkebamcaaclubfxadlmueccxoimlema
4  149d57cf92fc41cf94415803a877cb4b                           MISSING

   cons_12m  cons_gas_12m  cons_last_month date_activ   date_end  \
0         0         54946                0 2013-06-15 2016-06-15
1      4660             0                0 2009-08-21 2016-08-30
2       544             0                0 2010-04-16 2016-04-16
3      1584             0                0 2010-03-30 2016-03-30
4      4425             0              526 2010-01-13 2016-03-07

  date_modif_prod date_renewal  forecast_cons_12m  …  var_6m_price_p1_var  \
0      2015-11-01   2015-06-23               0.00  …             0.000131
1      2009-08-21   2015-08-31             189.95  …             0.000003
2      2010-04-16   2015-04-17              47.96  …             0.000004
3      2010-03-30   2015-03-31             240.04  …             0.000003
4      2010-01-13   2015-03-09             445.75  …             0.000011

   var_6m_price_p2_var  var_6m_price_p3_var  var_6m_price_p1_fix  \
0         4.100838e-05         9.084737e-04             2.086294
1         1.217891e-03         0.000000e+00             0.009482
2         9.450150e-08         0.000000e+00             0.000000
3         0.000000e+00         0.000000e+00             0.000000
4         2.896760e-06         4.860000e-10             0.000000

   var_6m_price_p2_fix  var_6m_price_p3_fix var_6m_price_p1  var_6m_price_p2  \
0            99.530517            44.235794        2.086425     9.953056e+01
1             0.000000             0.000000        0.009485     1.217891e-03
2             0.000000             0.000000        0.000004     9.450150e-08
3             0.000000             0.000000        0.000003     0.000000e+00
4             0.000000             0.000000        0.000011     2.896760e-06

   var_6m_price_p3  churn
0     4.423670e+01      1
1     0.000000e+00      0
2     0.000000e+00      0
3     0.000000e+00      0
4     4.860000e-10      0

[5 rows x 44 columns]
```

[88]:
```
merged_data.to_csv('clean_data_after_eda.csv')
```