0.0 Import Packages

In [46]:

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

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

1.0 Loading Data

In [2]:

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

In [3]:

```
history_data = pd.read_pickle(PICKLE_HISTORY_DIR)
train = pd.read_pickle(PICKLE_TRAIN_DIR)
```

2.0 Feature Engineering

In [4]:

```
mean_year = history_data.groupby(["id"]).mean().reset_index()
mean_6m = history_data[history_data["price_date"] > "2015-06-01"].groupby(["id"]).mean().r
mean_3m = history_data[history_data["price_date"] > "2015-10-01"].groupby(["id"]).mean().r
```

In [5]:

```
mean_year = mean_year.rename(index=str, columns={"price_p1_var": "mean_year_price_p1_var",
 "price_p2_var": "mean_year_price_p2_var",
"price_p3_var": "mean_year_price_p3_var",
"price_p1_fix": "mean_year_price_p1_fix",
"price_p2_fix": "mean_year_price_p2_fix",
"price_p3_fix": "mean_year_price_p3_fix",})
mean_year["mean_year_price_p1"] = mean_year["mean_year_price_p1_var"] + mean_year["mean_ye
mean_year["mean_year_price_p2"] = mean_year["mean_year_price_p2_var"] + mean_year["mean_ye
mean_year["mean_year_price_p3"] = mean_year["mean_year_price_p3_var"] + mean_year["mean_ye
```

In [6]:

```
mean_6m = mean_6m.rename(index=str, columns={"price_p1_var": "mean_6m_price_p1_var",
 "price_p2_var": "mean_6m_price_p2_var",
"price_p3_var": "mean_6m_price_p3_var",
"price_p1_fix": "mean_6m_price_p1_fix",
"price_p2_fix": "mean_6m_price_p2_fix",
"price_p3_fix": "mean_6m_price_p3_fix",})
mean_6m["mean_6m_price_p1"] = mean_6m["mean_6m_price_p1_var"] + mean_6m["mean_6m_price_p1_
mean_6m["mean_6m_price_p2"] = mean_6m["mean_6m_price_p2_var"] + mean_6m["mean_6m_price_p2_
mean_6m["mean_6m_price_p3"] = mean_6m["mean_6m_price_p3_var"] + mean_6m["mean_6m_price_p3_
```

In [7]:

```
mean_3m = mean_3m.rename(index=str, columns={"price_p1_var": "mean_3m_price_p1_var",
 "price_p2_var": "mean_3m_price_p2_var",
"price_p3_var": "mean_3m_price_p3_var",
"price_p1_fix": "mean_3m_price_p1_fix",
"price_p2_fix": "mean_3m_price_p2_fix",
"price_p3_fix": "mean_3m_price_p3_fix",})
mean_3m["mean_3m_price_p1"] = mean_3m["mean_3m_price_p1_var"] + mean_3m["mean_3m_price_p1_
mean_3m["mean_3m_price_p2"] = mean_3m["mean_3m_price_p2_var"] + mean_3m["mean_3m_price_p2_
mean_3m["mean_3m_price_p3"] = mean_3m["mean_3m_price_p3_var"] + mean_3m["mean_3m_price_p3_
```

In [52]:

```
features = mean_year
```

In [8]:

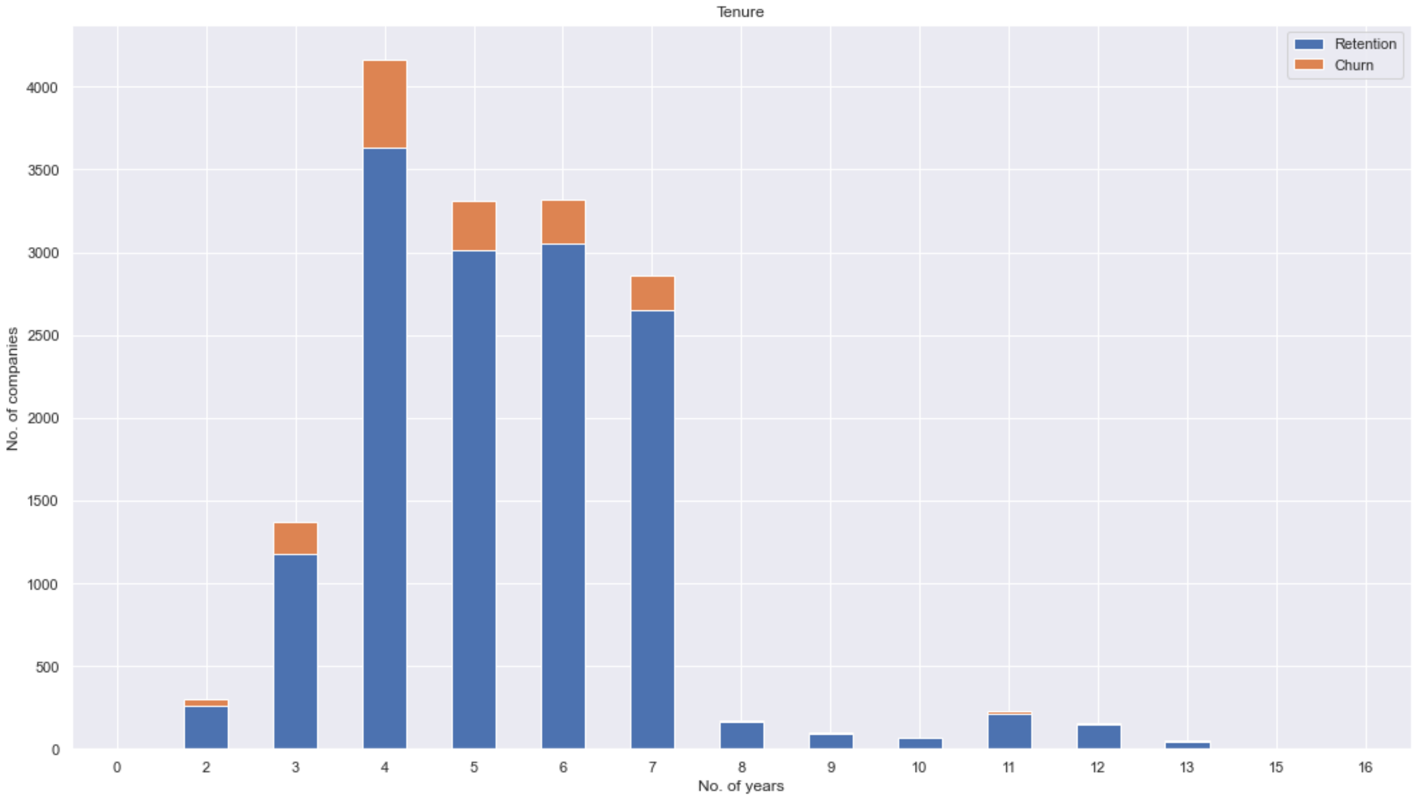
```
train["tenure"] = ((train["date_end"]-train["date_activ"])/ np.timedelta64(1, "Y")).astype
```

In [9]:

```
tenure = train[["tenure", "churn", "id"]].groupby(["tenure", "churn"])["id"].count().unsta
tenure_percentage = (tenure.div(tenure.sum(axis=1), axis=0)*100)
```

In [10]:

```
tenure.plot(kind="bar",
 figsize=(18,10),
 stacked=True,
rot=0,
 title= "Tenure")
# Rename legend
plt.legend(["Retention", "Churn"], loc="upper right")
# Labels
plt.ylabel("No. of companies")
plt.xlabel("No. of years")
plt.show()
```



In [11]:

```
def convert_months(reference_date, dataframe, column):
 """
 Input a column with timedeltas and return months
 """
 time_delta = REFERENCE_DATE - dataframe[column]
```

```
    months = (time_delta / np.timedelta64(1, "M")).astype(int)
    return months
```

In [12]:

```
REFERENCE_DATE = datetime.datetime(2016,1,1)
```

In [13]:

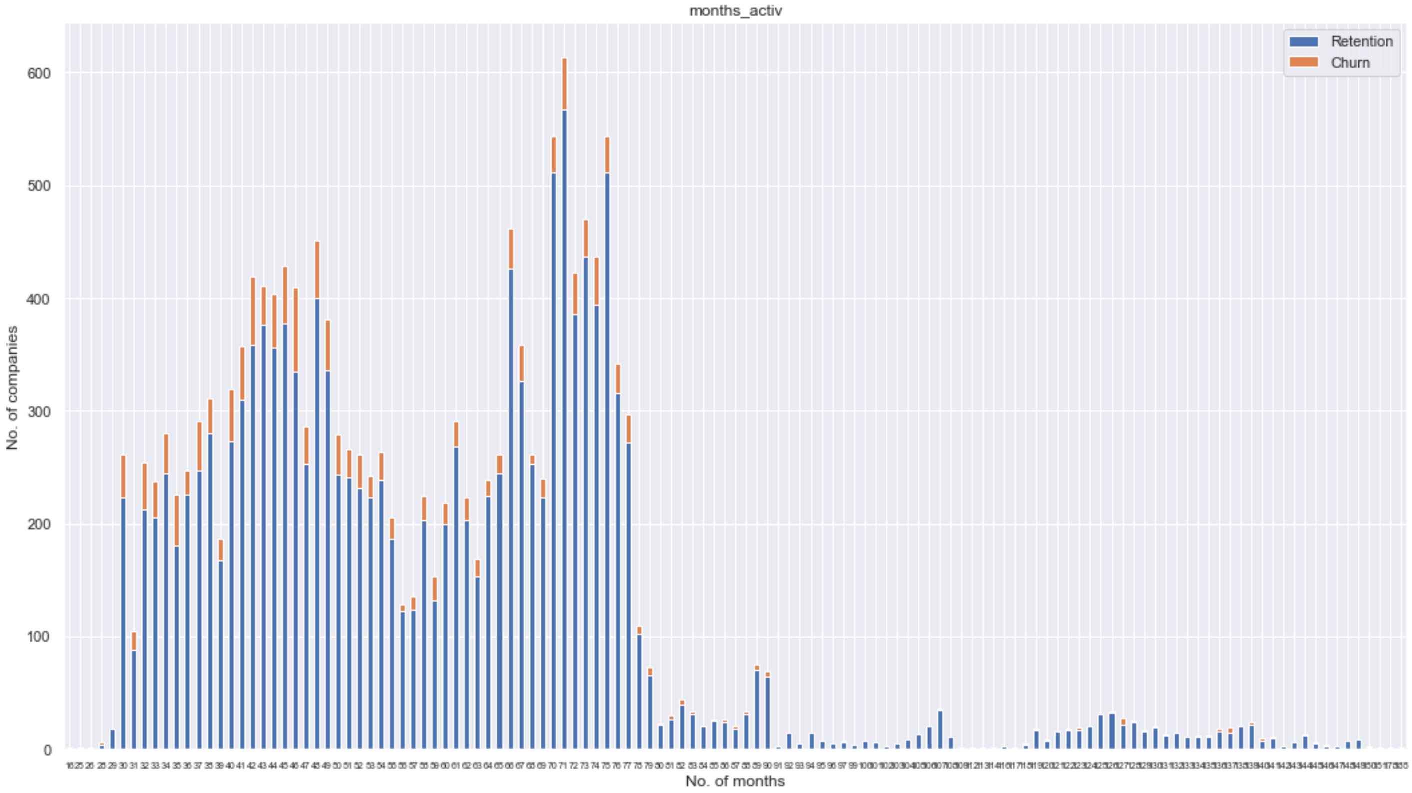
```
train["months_activ"] = convert_months(REFERENCE_DATE, train, "date_activ")
train["months_to_end"] = -convert_months(REFERENCE_DATE, train, "date_end")
train["months_modif_prod"] = convert_months(REFERENCE_DATE, train, "date_modif_prod")
train["months_renewal"] = convert_months(REFERENCE_DATE, train, "date_renewal")
```

In [14]:

```
def plot_churn_by_month(dataframe, column, fontsize_=11):
    """
    Plot churn distribution by monthly variable
    """
    temp = dataframe[[column, "churn", "id"]].groupby([column, "churn"])["id"].count().unstac
    temp.plot(kind="bar",
    figsize=(18,10),
    stacked=True,
    rot=0,
    title= column)
    # Rename legend
    plt.legend(["Retention", "Churn"], loc="upper right")
    # Labels
    plt.ylabel("No. of companies")
    plt.xlabel("No. of months")
    # Set xlabel fontsize
    plt.xticks(fontsize=fontsize_)
    plt.show()
```

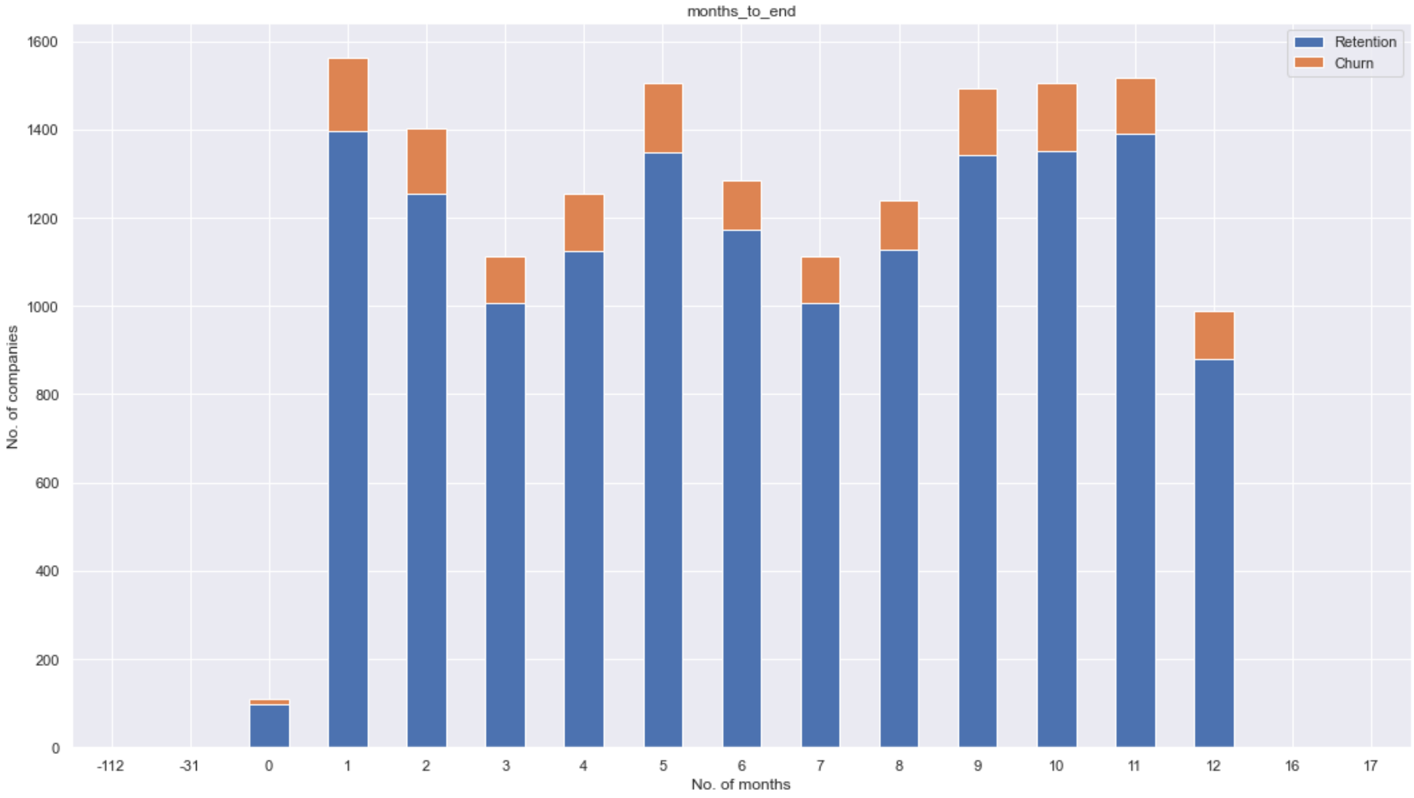
In [15]:

```
plot_churn_by_month(train, "months_activ", 7)
```

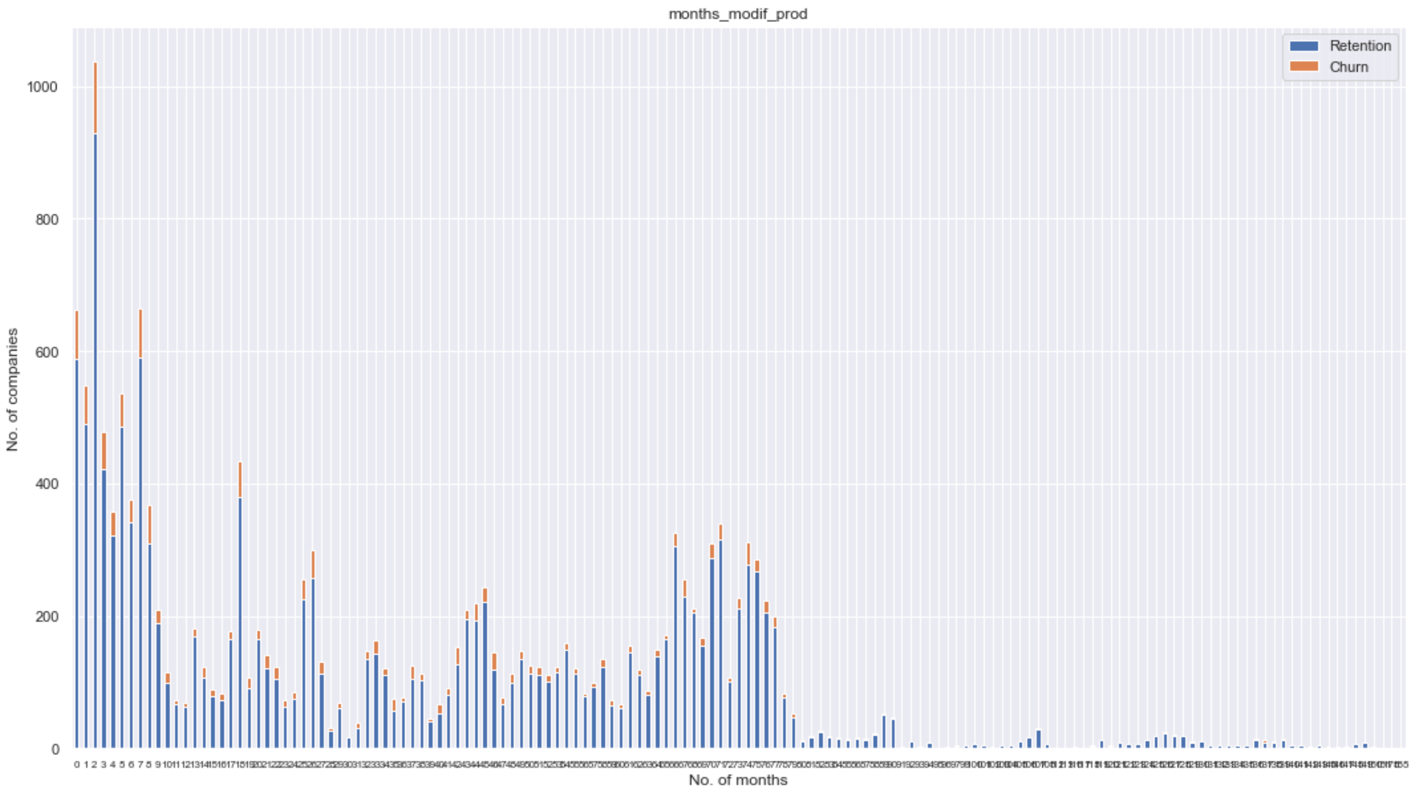


In [16]:

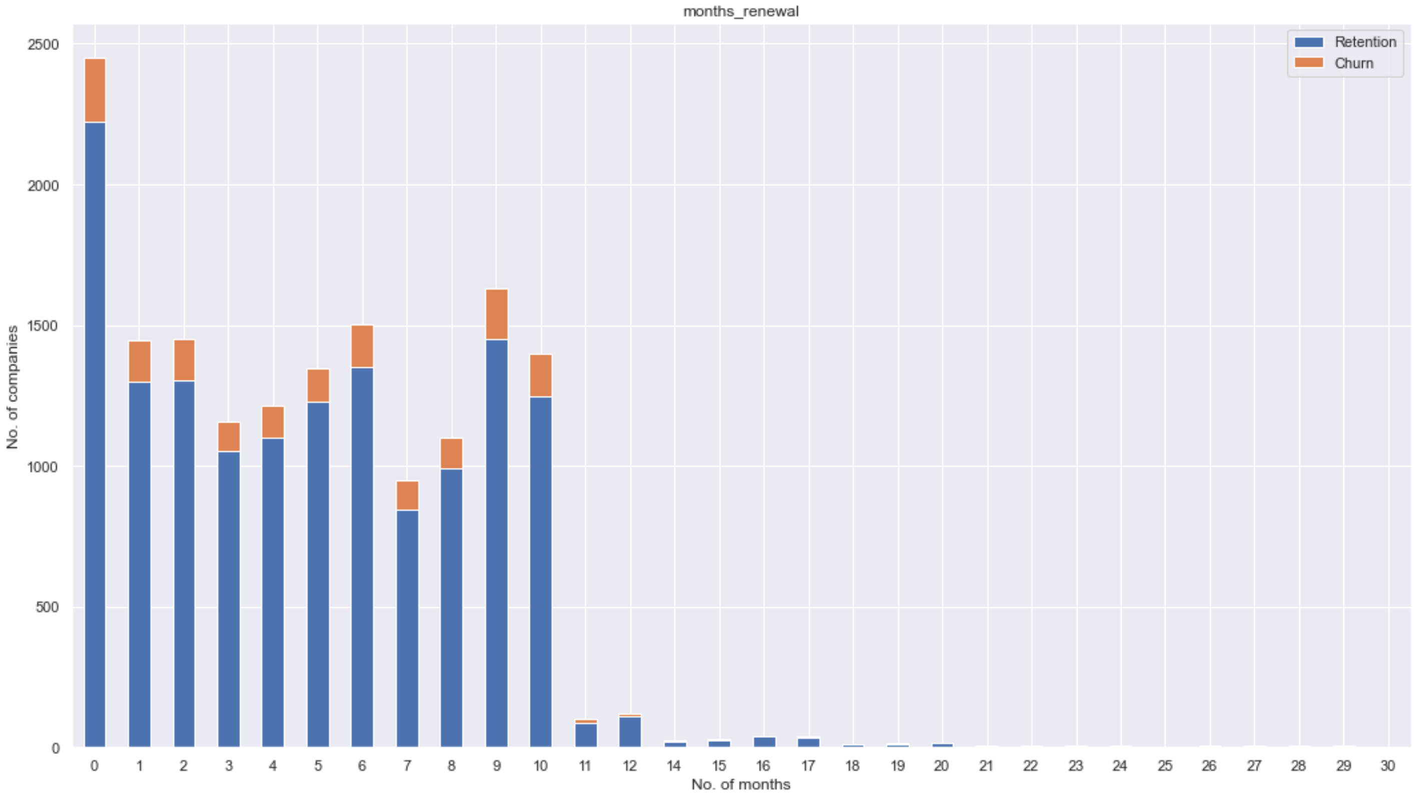
```
plot_churn_by_month(train, "months_to_end")
```



In [17]:
```
plot_churn_by_month(train, "months_modif_prod", 8)
```



In [18]:
```
plot_churn_by_month(train, "months_renewal")
```



Remove the date columns

In [19]:

```
train.drop(columns=["date_activ", "date_end", "date_modif_prod", "date_renewal"],inplace=T
```

2.1 Transforming boolean data

In [20]:

```
train["has_gas"]=train["has_gas"].replace(["t", "f"],[1,0])
```

2.2 Categorical data and dummy variables

In [21]:

```
train["channel_sales"] = train["channel_sales"].fillna("null_values_channel")
```

In [22]:

```
# Transform to categorical data type
train["channel_sales"] = train["channel_sales"].astype("category")
```

In [23]:

```
pd.DataFrame({"Samples in category": train["channel_sales"].value_counts()})
```

Out[23]:

	Samples in category
foosdfpfkusacimwkcsosbicdxkicaua	7377
null_values_channel	4218
lmkebamcaaclubfxadlmueccxoimlema	2073
usilxuppasemubllopkaafesmlibmsdf	1444
ewpakwlliwisiwduibdlfmalxowmwpci	966
sddiedcslfslkckwlfkdpoeeailfpeds	12
epumfxlbckeskwekxbiuasklxalciiuu	4
fixdbufsefwooaasfcxdxadsiekoceaa	2

In [24]:

```
categories_channel = pd.get_dummies(train["channel_sales"], prefix = "channel")
```

In [25]:

```
categories_channel.columns = [col_name[:11] for col_name in categories_channel.columns]
```

In [26]:

```
categories_channel.head(5)
```

Out[26]:

	channel_epu	channel_ewp	channel_fix	channel_foo	channel_lmk	channel_nul	channel_sdd	channel_us
0	0	0	0	0	1	0	0	
1	0	0	0	1	0	0	0	
2	0	0	0	0	0	1	0	
3	0	0	0	1	0	0	0	
4	0	0	0	0	1	0	0	

In [27]:

```
categories_channel.drop(columns=["channel_nul"],inplace=True)
```

In [28]:

```
train["origin_up"] = train["origin_up"].fillna("null_values_origin")
```

In [29]:

```
train["origin_up"] = train["origin_up"].astype("category")
```

In [30]:

```
pd.DataFrame({"Samples in category": train["origin_up"].value_counts()})
```

Out[30]:

	Samples in category
lxidpiddsbxsbosboudacockeimpuepw	7825
kamkkxfxxuwbdslkwifmmcsiusiuosws	4517
ldkssxwpmemidmecebumciepifcamkci	3664
null_values_origin	87
usapbepcfoloekilkwsdiboslwaxobdp	2
ewxeelcelemmiwuafmddpobolfuxioce	1

In [31]:

```
# Create dummy variables
categories_origin = pd.get_dummies(train["origin_up"], prefix = "origin")
# Rename columns for simplicity
categories_origin.columns = [col_name[:10] for col_name in categories_origin.columns]
```

In [32]:

```
categories_origin.head(5)
```

Out[32]:

	origin_ewx	origin_kam	origin_ldk	origin_lxi	origin_nul	origin_usa
0	0	0	1	0	0	0
1	0	0	0	1	0	0
2	0	1	0	0	0	0

	origin_ewx	origin_kam	origin_ldk	origin_lxi	origin_nul	origin_usa
3	0	1	0	0	0	0
4	0	1	0	0	0	0

In [33]:

```
categories_origin.drop(columns=["origin_nul"],inplace=True)
```

2.3 Categorical Data - Feature Engineering

In [34]:

```
train["activity_new"] = train["activity_new"].fillna("null_values_activity")
```

In [35]:

```
categories_activity = pd.DataFrame({"Activity samples":train["activity_new"].value_counts
categories_activity
```

Out[35]:

	Activity samples
null_values_activity	9545
apdekpcbwosbxepsfxclislboipuxpop	1577
kkklcdamwfafdcfwofuscwfwadblfmce	422
kwuslieomapmswolewpobpplkaooaaew	230
fmwdwsxillemwbbwelxsampiuwwpcdcb	219
...	...
iilxdefdkwudppkiekwlcexkdupeucla	1
kllldxcildwkssbmoabmsdffmawsafsf	1
wkwdccuiboaeaalcaawlwmldiwmpewma	1
ksukukiwxdxwbfwaapmuwippflemumlp	1
ewaupfkppoboxiuilledxxlwieawexel	1

420 rows × 1 columns

In [36]:

```
# Get the categories with less than 75 samples
to_replace = list(categories_activity[categories_activity["Activity samples"] <= 75].index
# Replace them with `null_values_categories`
train["activity_new"]=train["activity_new"].replace(to_replace,"null_values_activity")
```

In [37]:

```
# Create dummy variables
categories_activity = pd.get_dummies(train["activity_new"], prefix = "activity")
# Rename columns for simplicity
categories_activity.columns = [col_name[:12] for col_name in categories_activity.columns]
```

In [38]:

```
categories_activity.head(5)
```

Out[38]:

	activity_apd	activity_ckf	activity_clu	activity_cwo	activity_fmw	activity_kkk	activity_kwu	activity_nul
0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1

	activity_apd	activity_ckf	activity_clu	activity_cwo	activity_fmw	activity_kkk	activity_kwu	activity_nul
2	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	1

In [39]:

```
# Use common index to merge
train = pd.merge(train, categories_channel, left_index=True, right_index=True)
train = pd.merge(train, categories_origin, left_index=True, right_index=True)
train = pd.merge(train, categories_activity, left_index=True, right_index=True)
```

In [40]:

```
train.drop(columns=["channel_sales", "origin_up", "activity_new"],inplace=True)
```

In [41]:

```
train.describe()
```

Out[41]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_dis
count	1.609600e+04	1.609600e+04	1.609600e+04	16096.000000	16096.000000	
mean	1.948044e+05	3.191164e+04	1.946154e+04	2370.555949	1907.347229	
std	6.795151e+05	1.775885e+05	8.235676e+04	4035.085664	5257.364759	
min	-1.252760e+05	-3.037000e+03	-9.138600e+04	-16689.260000	-85627.000000	
25%	5.906250e+03	0.000000e+00	0.000000e+00	513.230000	0.000000	
50%	1.533250e+04	0.000000e+00	9.010000e+02	1179.160000	378.000000	
75%	5.022150e+04	0.000000e+00	4.127000e+03	2692.077500	1994.250000	
max	1.609711e+07	4.188440e+06	4.538720e+06	103801.930000	175375.000000	

In [43]:

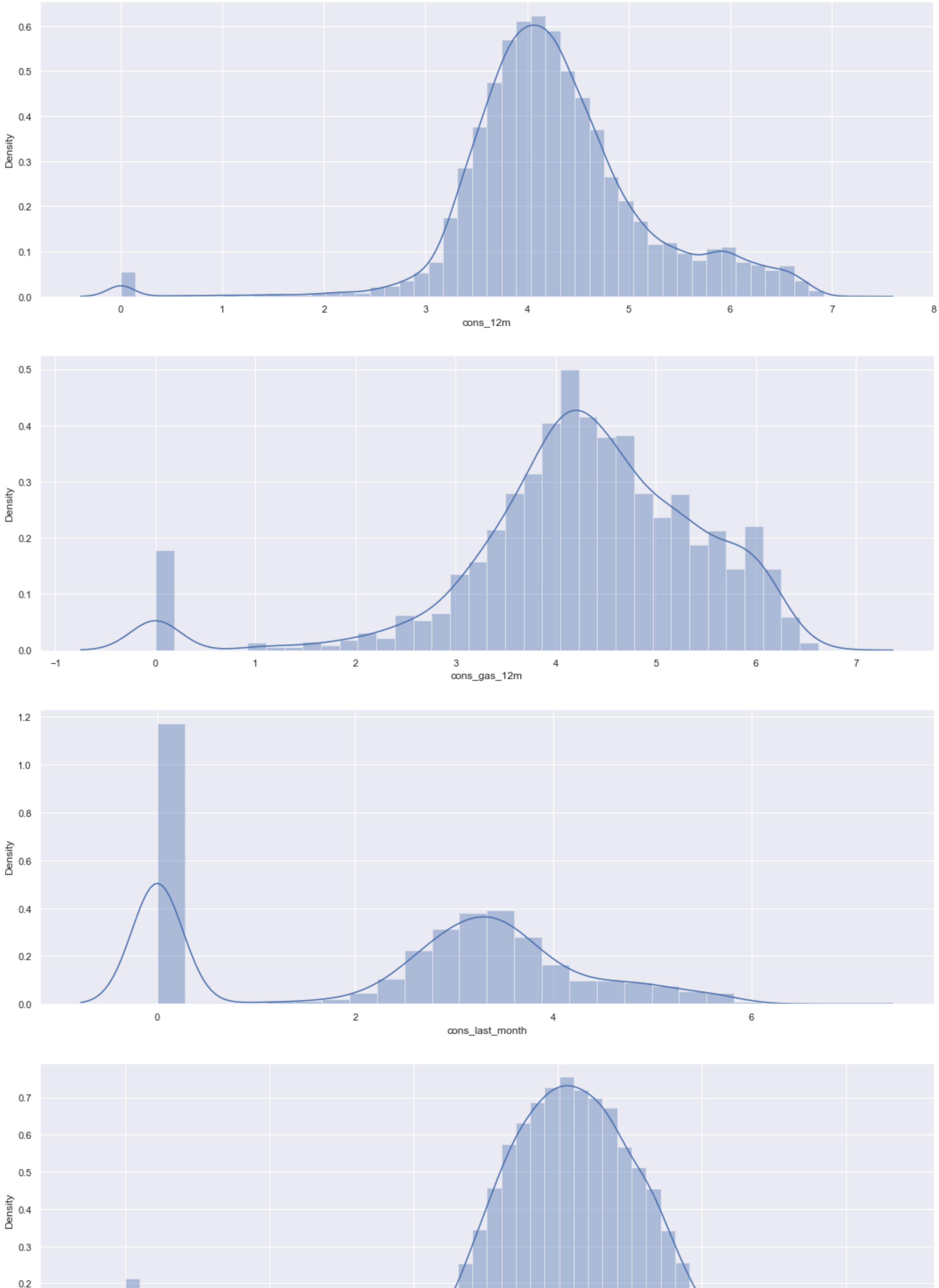
```
# Remove negative values
train.loc[train.cons_12m < 0,"cons_12m"] = np.nan
train.loc[train.cons_gas_12m < 0,"cons_gas_12m"] = np.nan
train.loc[train.cons_last_month < 0,"cons_last_month"] = np.nan
train.loc[train.forecast_cons_12m < 0,"forecast_cons_12m"] = np.nan
train.loc[train.forecast_cons_year < 0,"forecast_cons_year"] = np.nan
train.loc[train.forecast_meter_rent_12m < 0,"forecast_meter_rent_12m"] = np.nan
train.loc[train.imp_cons < 0,"imp_cons"] = np.nan
```

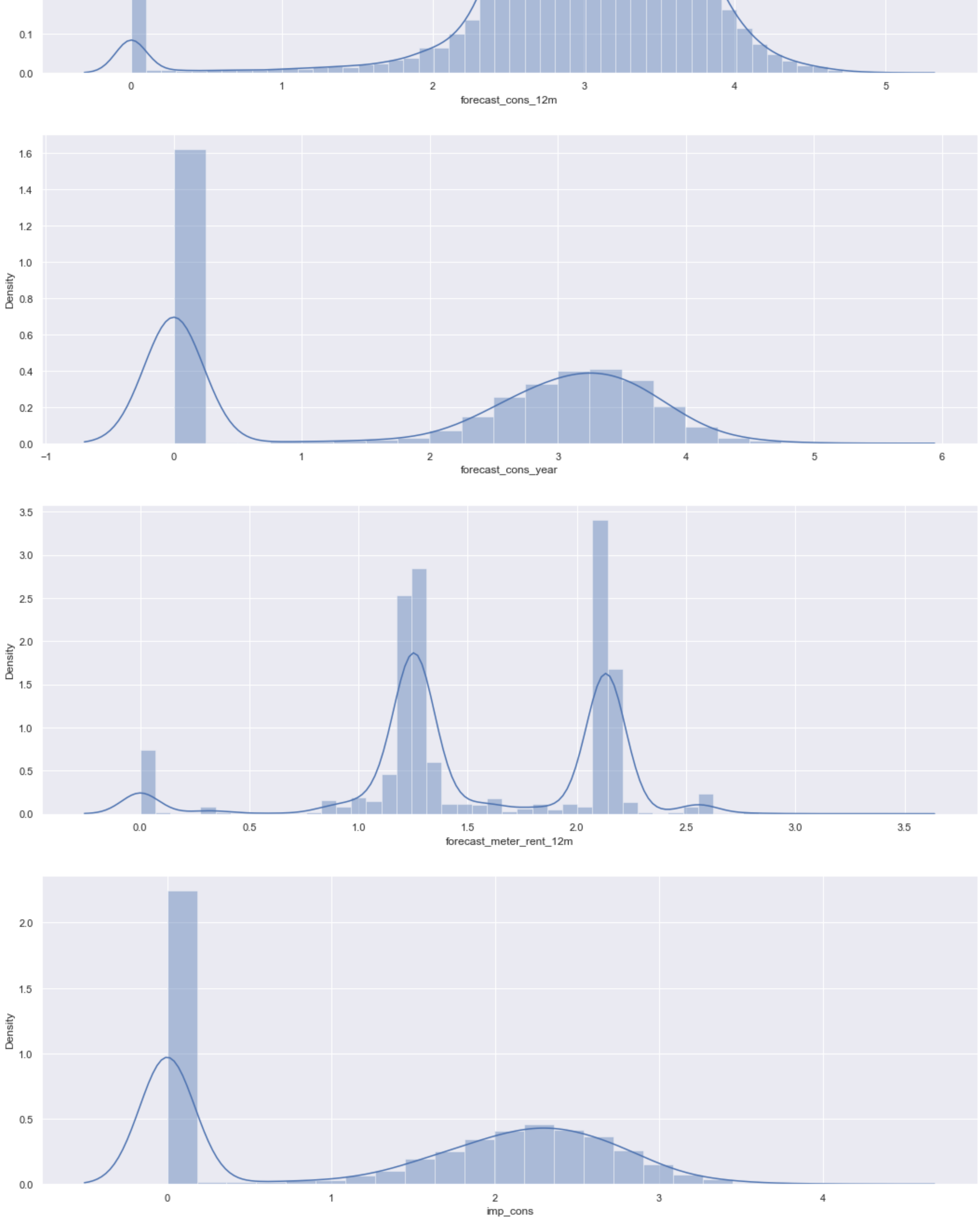
In [44]:

```
# Apply log10 transformation
train["cons_12m"] = np.log10(train["cons_12m"]+1)
train["cons_gas_12m"] = np.log10(train["cons_gas_12m"]+1)
train["cons_last_month"] = np.log10(train["cons_last_month"]+1)
train["forecast_cons_12m"] = np.log10(train["forecast_cons_12m"]+1)
train["forecast_cons_year"] = np.log10(train["forecast_cons_year"]+1)
train["forecast_meter_rent_12m"] = np.log10(train["forecast_meter_rent_12m"]+1)
train["imp_cons"] = np.log10(train["imp_cons"]+1)
```

In [47]:

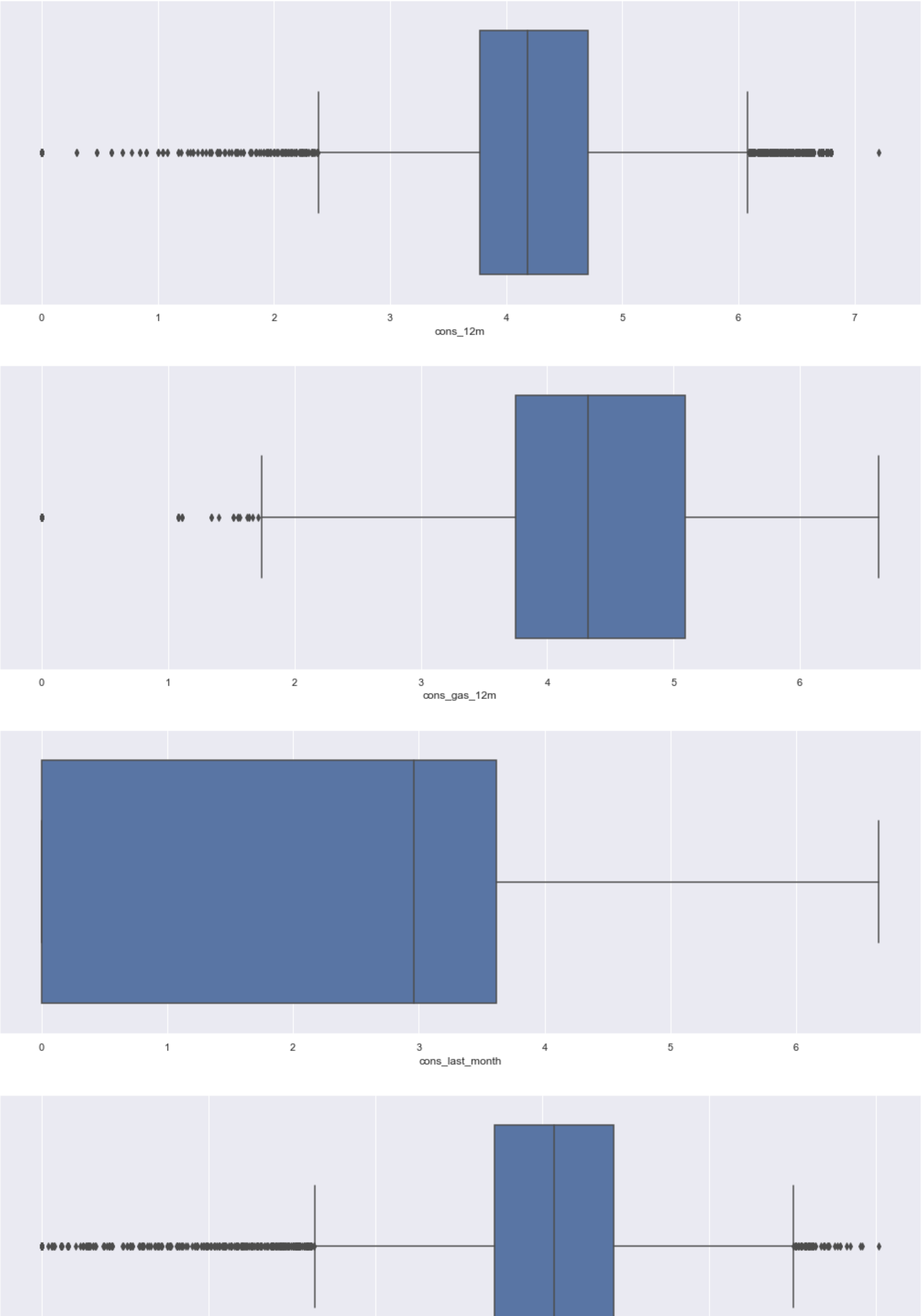
```
fig, axs = plt.subplots(nrows=7, figsize=(18,50))
# Plot histograms
sns.distplot((train["cons_12m"].dropna()), ax=axs[0])
sns.distplot((train[train["has_gas"]==1]["cons_gas_12m"].dropna()), ax=axs[1])
sns.distplot((train["cons_last_month"].dropna()), ax=axs[2])
sns.distplot((train["forecast_cons_12m"].dropna()),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

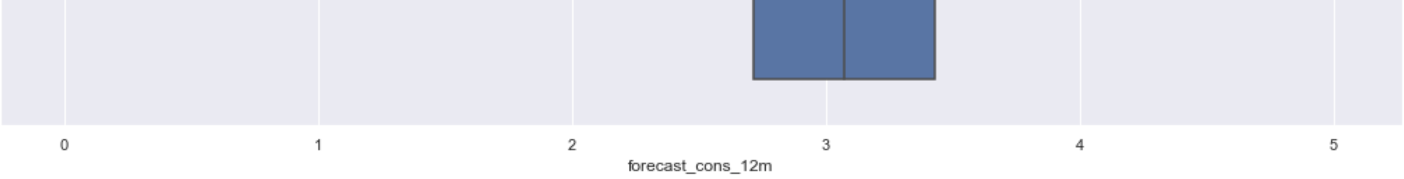


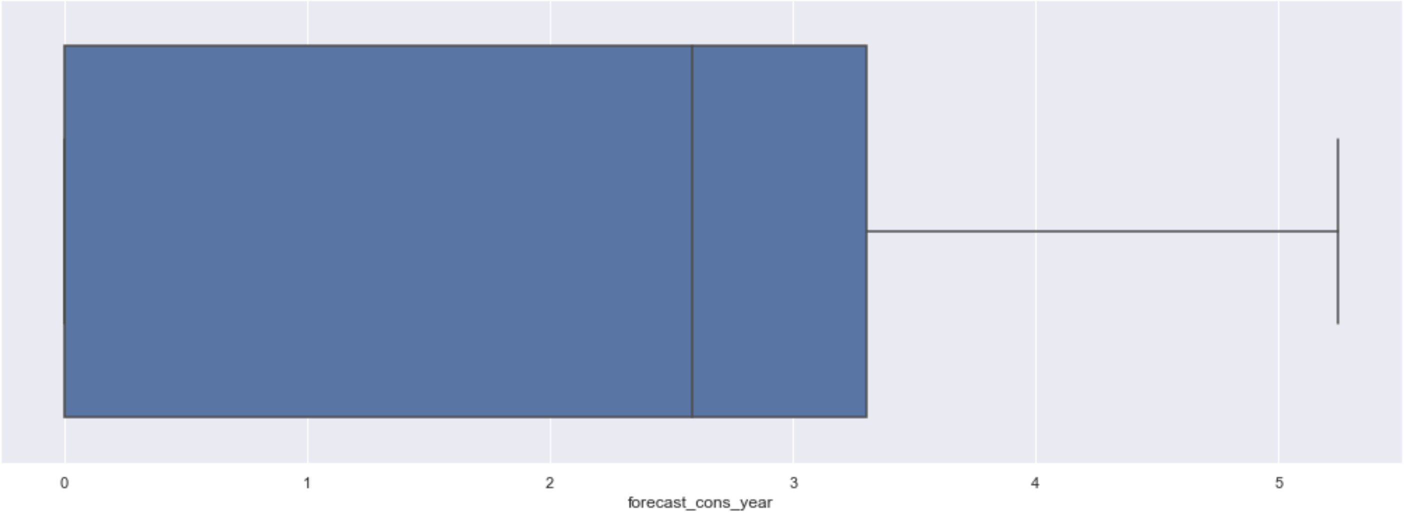


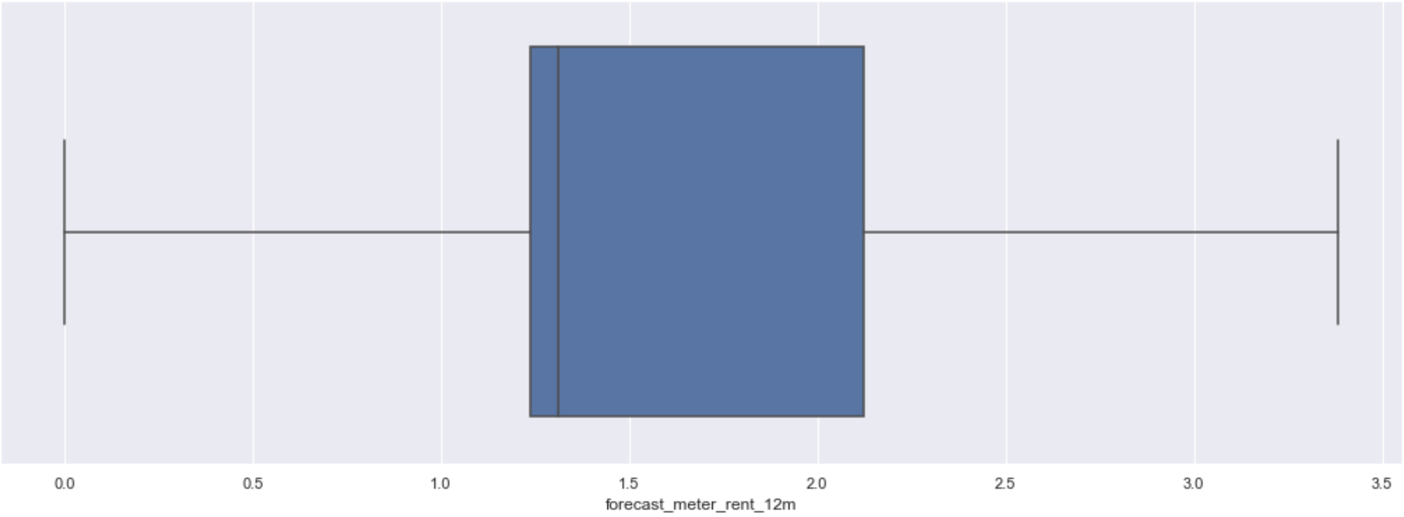
In [48]:

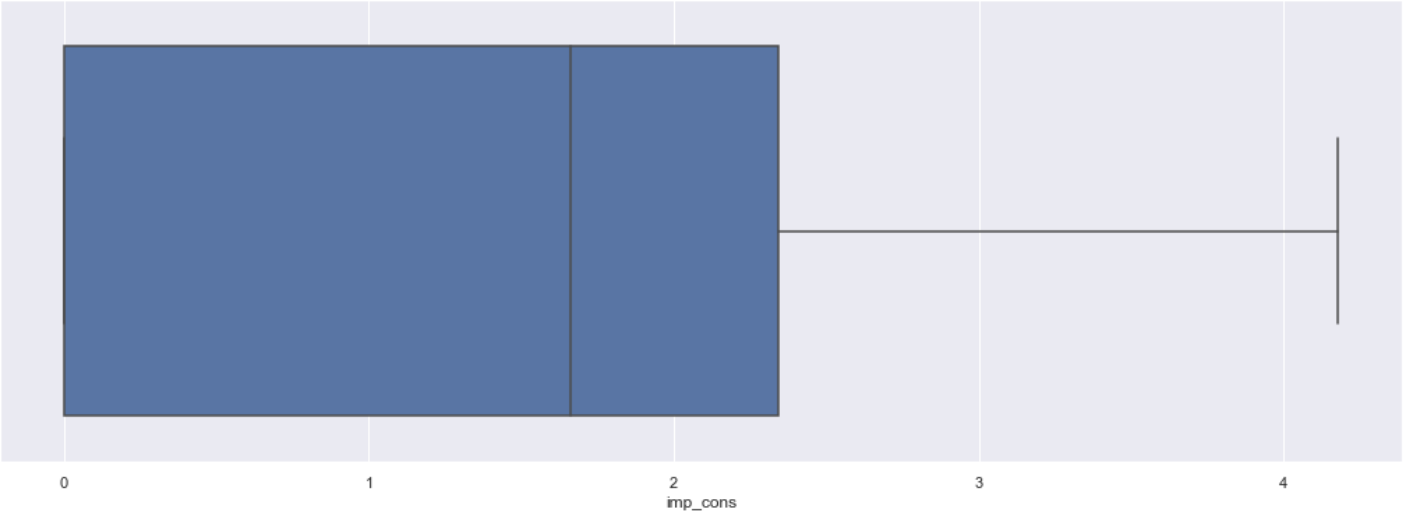
```
fig, axs = plt.subplots(nrows=7, figsize=(18,50))
# Plot boxplots
sns.boxplot((train["cons_12m"].dropna()), ax=axs[0])
sns.boxplot((train[train["has_gas"]==1]["cons_gas_12m"].dropna()), ax=axs[1])
sns.boxplot((train["cons_last_month"].dropna()), ax=axs[2])
sns.boxplot((train["forecast_cons_12m"].dropna()),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











In [50]:

```
train.describe()
```

Out[50]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_dis
count	16069.000000	16090.000000	16050.000000	16055.000000	16071.000000	

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_dis
mean	4.283812	0.800300	2.359281	3.006826	1.869956	
std	0.915265	1.748833	1.789067	0.709778	1.612963	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	3.773786	0.000000	0.000000	2.713952	0.000000	
50%	4.187408	0.000000	2.959041	3.073579	2.583199	
75%	4.701508	0.000000	3.617000	3.430950	3.301030	
max	7.206748	6.622052	6.656933	5.016210	5.243970	

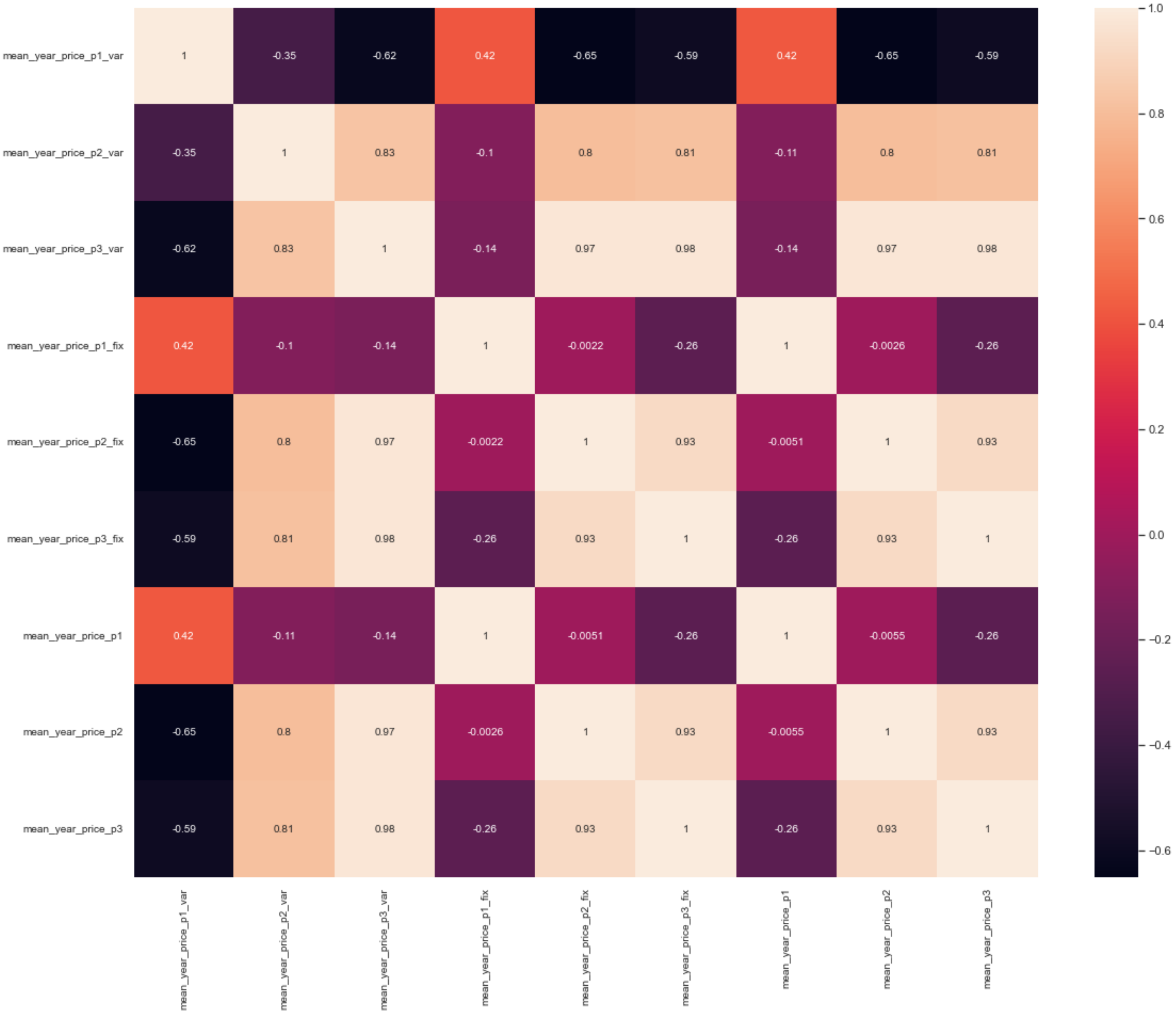
3.0 High correlation variables

In [53]:

```
# Calculate correlation of variables
correlation = features.corr()
```

In [54]:

```
 # Plot correlation
plt.figure(figsize=(19,15))
sns.heatmap(correlation, xticklabels=correlation.columns.values,
 yticklabels=correlation.columns.values, annot = True, annot_kws={'size':10})
# Axis ticks size
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

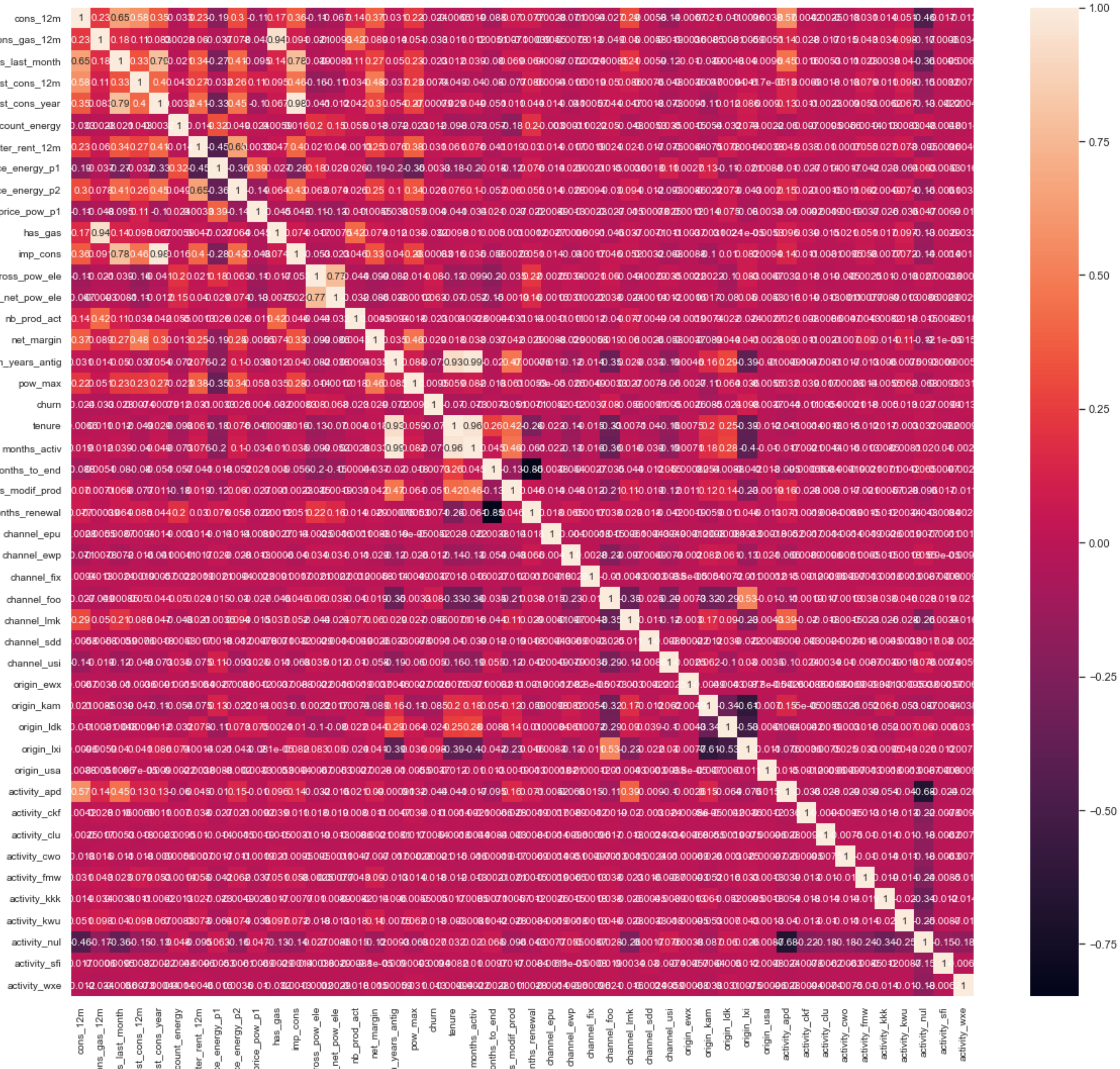


In [55]:

```
correlation = train.corr()
```

In [56]:

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


cons_12m
cons_gas_12m
cons_last_month
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
has_gas
imp_cons
margin_gross_pow_ele
margin_net_pow_ele
nb_prod_act
net_margin
num_years_antig
pow_max
churn
tenure
months_activ
months_to_end
months_modif_prod
months_renewal
channel_epu
channel_ewp
channel_fix
channel_foo
channel_lmk
channel_sdd
channel_usi
origin_ewx
origin_kam
origin_ldk
origin_lxi
origin_usa
activity_apd
activity_ckf
activity_clu
activity_cwo
activity_fmw
activity_kkk
activity_kwu
activity_nul
activity_sfi
activity_wxe
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75