Feature Engineering

- By - Gautam Sharma

I. Import all require libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import datetime
import pickle
import os
```

In [2]:

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

In [3]:

```
# Set maximum number of columns to be displayed
pd.set_option('display.max_columns', 100)
```

In [ ]:

```

```

In [ ]:

```

```

Load data into DataFrame

In [4]:

```
data_hist = pd.read_csv('ml_history_processed.csv')
data_train = pd.read_csv('train.csv')
```

In [5]:

```
data_hist.head(5)
```

Out[5]:

	id	price_date	price_p1_var	price_p2_var	price_p3_var
0	038af19179925da21a25619c5a24b745	1/1/2015	0.151367	0.0	0.0
1	038af19179925da21a25619c5a24b745	2/1/2015	0.151367	0.0	0.0
2	038af19179925da21a25619c5a24b745	3/1/2015	0.151367	0.0	0.0
3	038af19179925da21a25619c5a24b745	4/1/2015	0.149626	0.0	0.0
4	038af19179925da21a25619c5a24b745	5/1/2015	0.149626	0.0	0.0

Feature Engineering

We will create new features last six months , and the last 3 months our model.

In [6]:

```
mean_year = data_hist.groupby(['id']).mean().reset_index()
```

In [7]:

```
mean_6m = data_hist[data_hist['price_date']> '2015-06-01'].groupby(['id']).mean
().reset_index()
```

In [8]:

```
mean_3m = data_hist[data_hist['price_date'] > '2015-10-01'].groupby(['id']).mean
().reset_index()
```

Combine all of them in single DataFrame

In [9]:

```
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',

})
```

In [10]:

```
mean_year['mean_year_price_p1'] = mean_year['mean_year_price_p1_var'] + mean_yea
r['mean_year_price_p1_fix']
mean_year['mean_year_price_p2'] = mean_year['mean_year_price_p2_var'] + mean_yea
r['mean_year_price_p2_fix']
mean_year['mean_year_price_p3'] = mean_year['mean_year_price_p3_var'] + mean_yea
r['mean_year_price_p3_fix']
```

In [11]:

```
# For 6 months mean_6m
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',

})
```

In [12]:

```
mean_6m['mean_6m_price_p1'] = mean_6m['mean_6m_price_p1_var'] + mean_6m['mean_6m
_price_p1_fix']
mean_6m['mean_6m_price_p2'] = mean_6m['mean_6m_price_p2_var'] + mean_6m['mean_6m
_price_p2_fix']
mean_6m['mean_6m_price_p3'] = mean_6m['mean_6m_price_p3_var'] + mean_6m['mean_6m
_price_p3_fix']
```

In [13]:

```
# For 3 months mean_3m
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',

})
```

In [14]:

```
mean_3m['mean_3m_price_p1'] = mean_3m['mean_3m_price_p1_var'] + mean_3m['mean_3m
_price_p1_fix']
mean_3m['mean_3m_price_p2'] = mean_3m['mean_3m_price_p2_var'] + mean_3m['mean_3m
_price_p2_fix']
mean_3m['mean_3m_price_p3'] = mean_3m['mean_3m_price_p3_var'] + mean_3m['mean_3m
_price_p3_fix']
```

- Now we will mwrge them into a single dataframe

In [15]:

```
features = mean_year
```

Feature Engineering

In [16]:

```
data_train.head(5)
```

Out[16]:

	Unnamed: 0	id	activity_new	campaign_
0	0	48ada52261e7cf58715202705a0451c9	esoiiifxdlbkcsluxmfuacbdckommixw	
1	1	24011ae4ebbe3035111d65fa7c15bc57	NaN	
2	2	d29c2c54acc38ff3c0614d0a653813dd	NaN	
3	3	764c75f661154dac3a6c254cd082ea7d	NaN	
4	4	bba03439a292a1e166f80264c16191cb	NaN	

Transforming boolean data

In [17]:

```
data_train['has_gas'] = data_train['has_gas'].replace(['t','f'],[1,0])
```

Categorical Data And Dummy Variables

In [18]:

```
data_train['channel_sales'] = data_train['channel_sales'].fillna('null_values_channel')
```

In [19]:

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

We want to see how many categories we will end up with

In [20]:

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

Out[20]:

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

In [21]:

```
# Create dummy variables
categories_channel = pd.get_dummies(data_train['channel_sales'],prefix='channel'
)
```

In [22]:

```
# Rename columns for simplicity
categories_channel.columns = [col_name[:11] for col_name in categories_channel.c
olumns]
```

In [23]:

```
categories_channel.head()
```

Out[23]:

	channel_epu	channel_ewp	channel_fix	channel_foo	channel_lmk	channel_nul	channel_
0	0	0	0	0	1	0	
1	0	0	0	1	0	0	
2	0	0	0	0	0	1	
3	0	0	0	1	0	0	
4	0	0	0	0	1	0	

In [24]:

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

Categorical Data

- First of all let's replace the NAN values with a string called null_values_origin

In [25]:

```
data_train['origin_up'] = data_train['origin_up'].fillna('null_values_origin')
```

Now transform the origin_up column into categorical data type

In [26]:

```
data_train['origin_up'] = data_train['origin_up'].astype('category')
```

How many categories

In [27]:

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

Out[27]:

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

In [28]:

```
# Create dummy variables
categories_origin = pd.get_dummies(data_train['origin_up'], prefix = 'origin')
```

In [29]:

```
categories_origin.columns = [col_name[:10] for col_name in categories_origin.col
umns]
```

In [30]:

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

Out[30]:

	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
3	0	1	0	0	0	0
4	0	1	0	0	0	0

Finally remove one column to avoid the dummy variable trap

Categorical data - Feature Engineering

First of all let's replce the NAN values with a sting called null_values_activity

In [31]:

```
categories_activity = pd.DataFrame({'Activity samples':data_train['activity_new'
].value_counts()})
```

In [32]:

```
categories_activity
```

Out[32]:

	Activity samples
apdekpcbwosbxepsfxclislboipuxpop	1577
kkklcdamwfafdcfwofuscwfwadblfmce	422
kwuslieomapmswolewpobpplkaooaaew	230
fmwdwsxillemwbbwelxsampiuwwpcdcb	219
ckfxocssowaeipxueikxcmaxdmcduxsa	189
...	...
exmccxcauwolkacaceedipbcmodfedfl	1
akakmkfwoesfipbpaodfippfklpkuxdd	1
beplffiwdfsmiuodulsfscelscscbdix	1
dbklukmppmseoekmmxfolmfbdidmawls	1
lblmkbemolxxlkicccsucmoapesxsplx	1

419 rows × 1 columns

As we can see below there are too many categories with vary few number of samples. So we will replace any category with less than 75 null_values_category

In [33]:

```
# 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`
data_train["activity_new"]=data_train["activity_new"].replace(to_replace,"null_v
alues_activity")
```

In [34]:

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

In [35]:

```
categories_activity.head(3)
```

Out[35]:

	activity_apd	activity_ckf	activity_clu	activity_cwo	activity_fmw	activity_kkk	activity_kwu
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0

Finally remove one column to avoid the dummy variable trap

In [36]:

```
categories_activity.drop(columns=['activity_nul'], inplace=True)
```

Merge dummy variables to main dataframe

In [37]:

```
# Use common index to merge
train_new = pd.merge(data_train, categories_channel, left_index=True, right_inde
x=True)
train_new = pd.merge(data_train, categories_origin, left_index=True, right_index
=True)
train_new = pd.merge(data_train, categories_activity, left_index=True, right_ind
ex=True)
```

In [38]:

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

In [39]:

```
train_new.describe()
```

Out[39]:

	Unnamed: 0	campaign_disc_ele	cons_12m	cons_gas_12m	cons_last_month	fo
count	16096.000000	0.0	1.609600e+04	1.609600e+04	1.609600e+04	
mean	8047.500000	NaN	1.948044e+05	3.191164e+04	1.946154e+04	
std	4646.659302	NaN	6.795151e+05	1.775885e+05	8.235676e+04	
min	0.000000	NaN	-1.252760e+05	-3.037000e+03	-9.138600e+04	
25%	4023.750000	NaN	5.906250e+03	0.000000e+00	0.000000e+00	
50%	8047.500000	NaN	1.533250e+04	0.000000e+00	9.010000e+02	
75%	12071.250000	NaN	5.022150e+04	0.000000e+00	4.127000e+03	
max	16095.000000	NaN	1.609711e+07	4.188440e+06	4.538720e+06	

Remove the negative values

In [40]:

```
train_new.loc[train_new.cons_12m < 0,"cons_12m"] = np.nan
train_new.loc[train_new.cons_gas_12m < 0,"cons_gas_12m"] = np.nan
train_new.loc[train_new.cons_last_month < 0,"cons_last_month"] = np.nan
train_new.loc[train_new.forecast_cons_12m < 0,"forecast_cons_12m"] = np.nan
train_new.loc[train_new.forecast_cons_year < 0,"forecast_cons_year"] = np.nan
train_new.loc[train_new.forecast_meter_rent_12m < 0,"forecast_meter_rent_12m"] = np.nan
train_new.loc[train_new.imp_cons < 0,"imp_cons"] = np.nan
```

Apply the Log10 Transformation

In [41]:

```
train_new["cons_12m"] = np.log10(train_new["cons_12m"]+1)
train_new["cons_gas_12m"] = np.log10(train_new["cons_gas_12m"]+1)
train_new["cons_last_month"] = np.log10(train_new["cons_last_month"]+1)
train_new["forecast_cons_12m"] = np.log10(train_new["forecast_cons_12m"]+1)
train_new["forecast_cons_year"] = np.log10(train_new["forecast_cons_year"]+1)
train_new["forecast_meter_rent_12m"] = np.log10(train_new["forecast_meter_rent_12m"]+1)
train_new["imp_cons"] = np.log10(train_new["imp_cons"]+1)
```

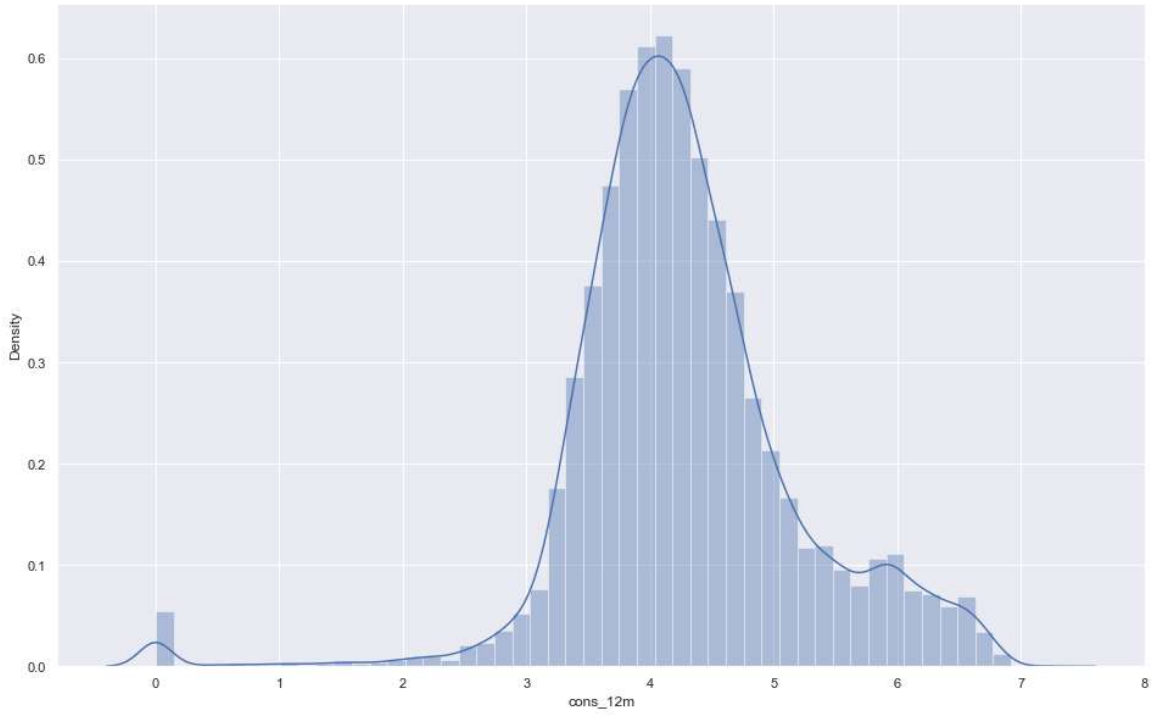
In [42]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new["cons_12m"].dropna()))
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7cbd8eb48>
```



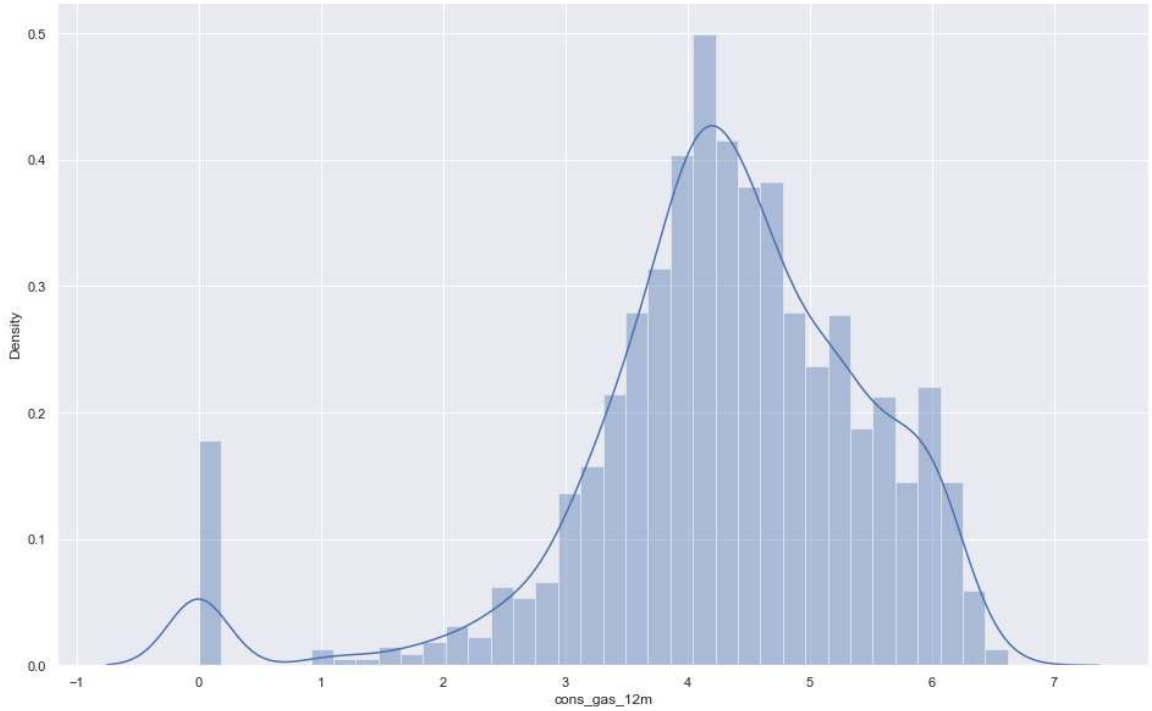
In [43]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new[train_new["has_gas"]==1]["cons_gas_12m"].dropna()))
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7cd779d48>
```



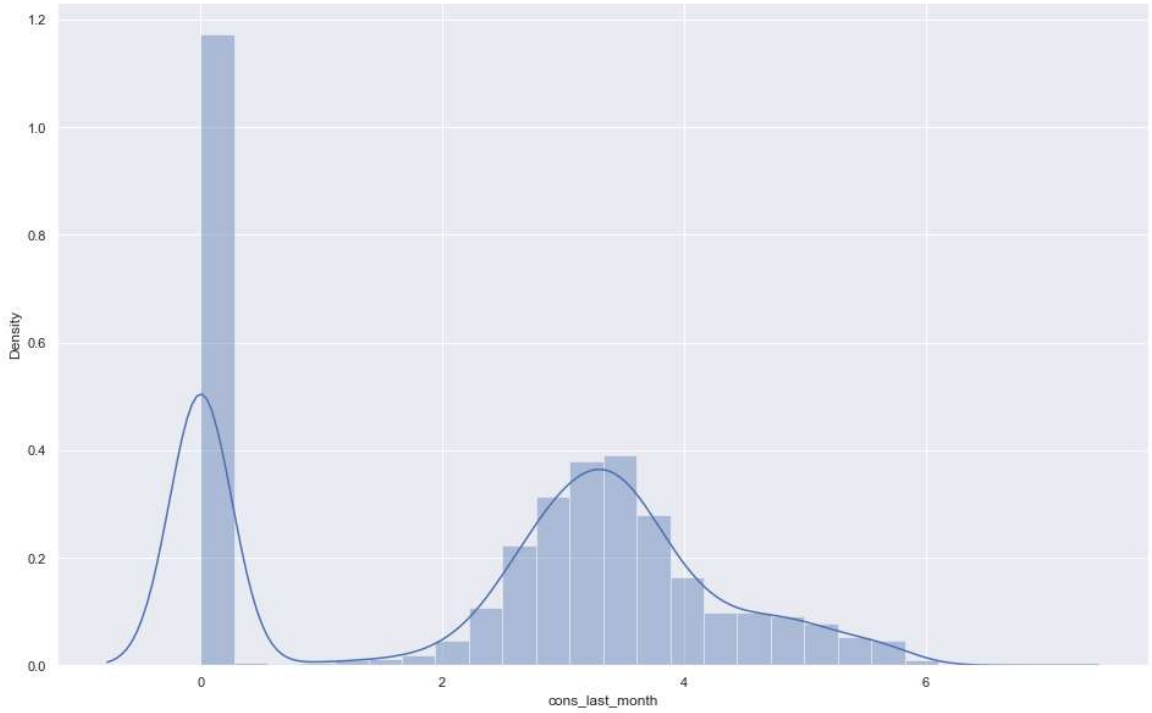
In [44]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new["cons_last_month"].dropna()))
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7ccdb2488>
```



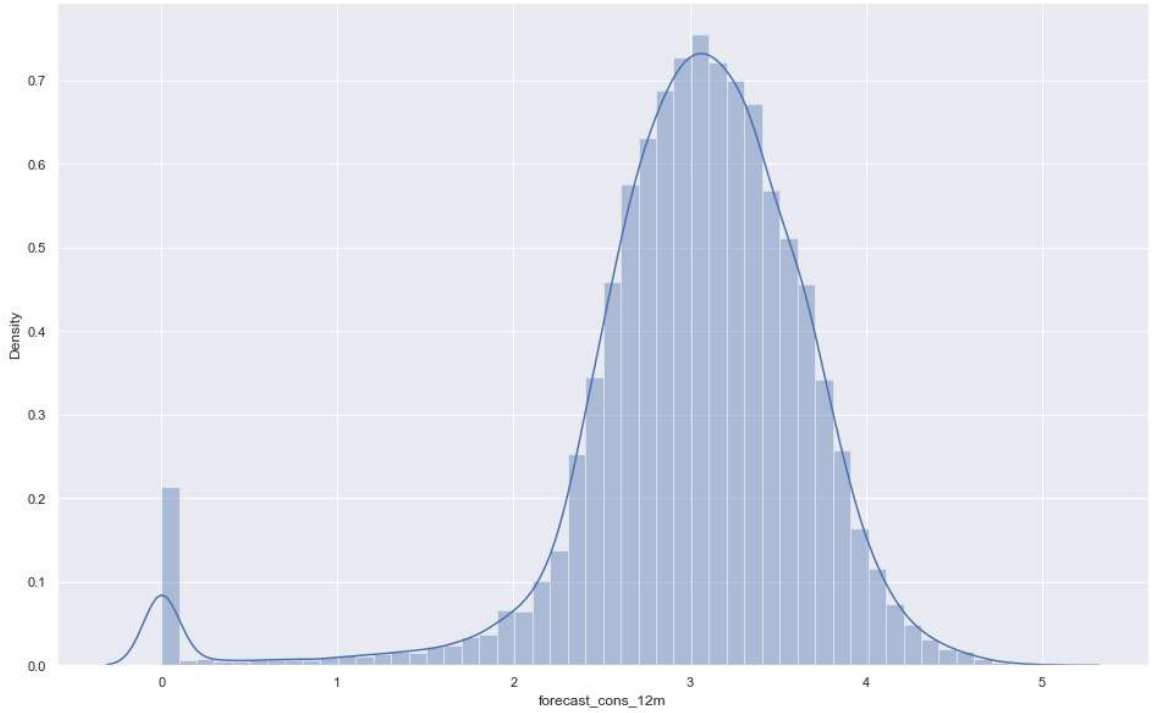
In [45]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new["forecast_cons_12m"].dropna()))
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7ccec7fc8>
```



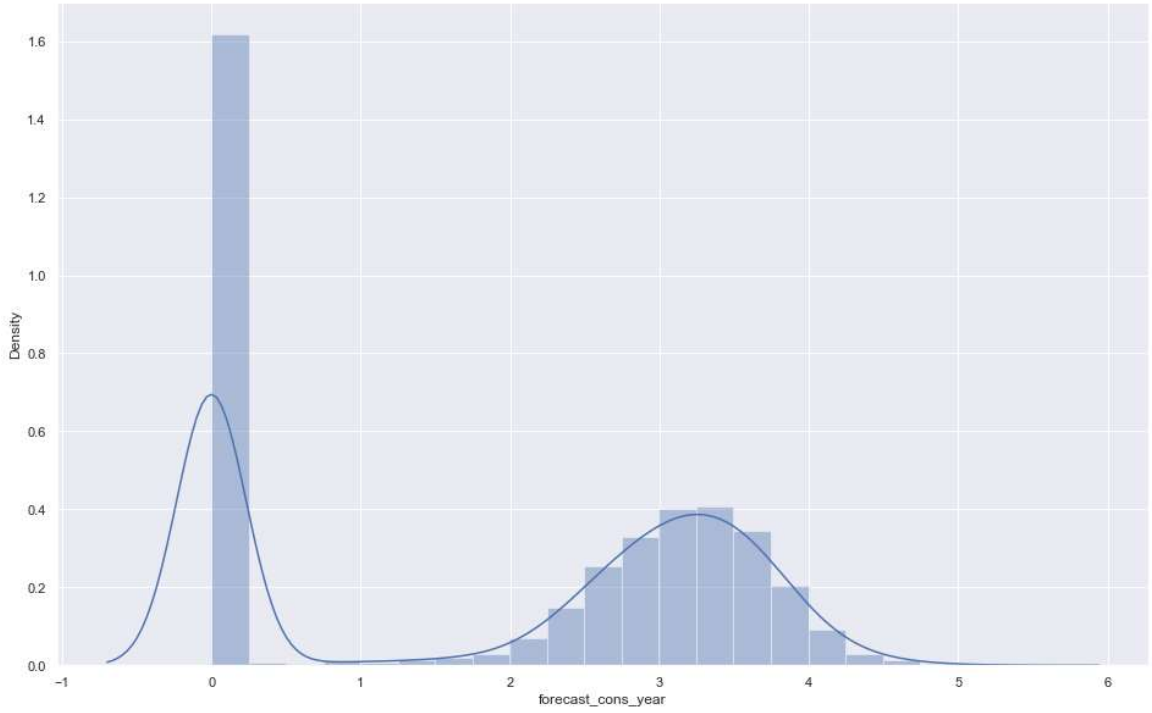
In [46]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new["forecast_cons_year"].dropna()))
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7cd7c74c8>
```



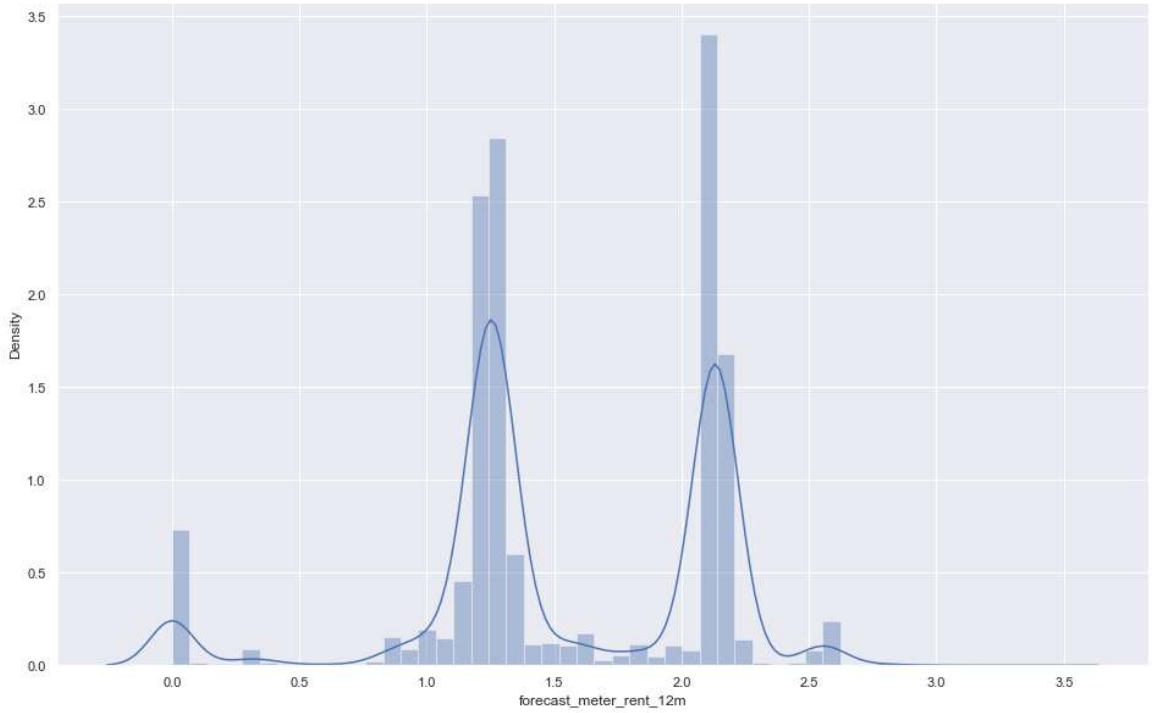
In [47]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new["forecast_meter_rent_12m"].dropna()))
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[47]:

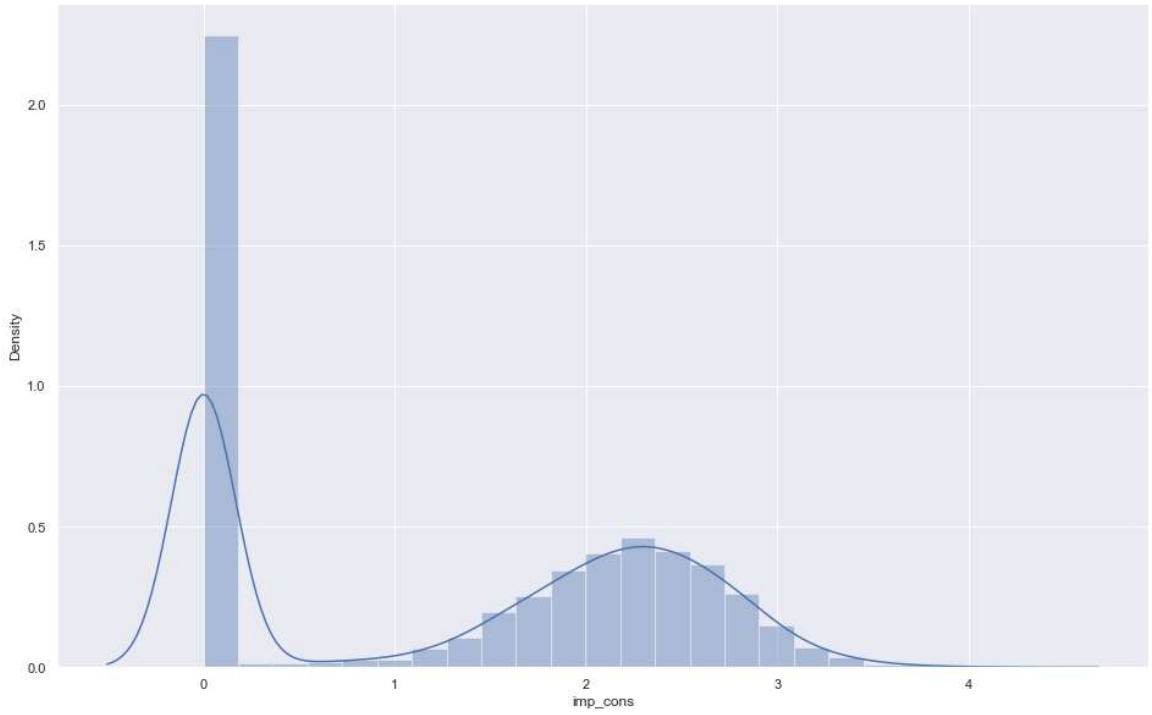
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7cdac5088>
```



In [48]:

```
fig = plt.subplots(figsize=(16,10))
sns.distplot((train_new["imp_cons"].dropna()))
plt.show()
```

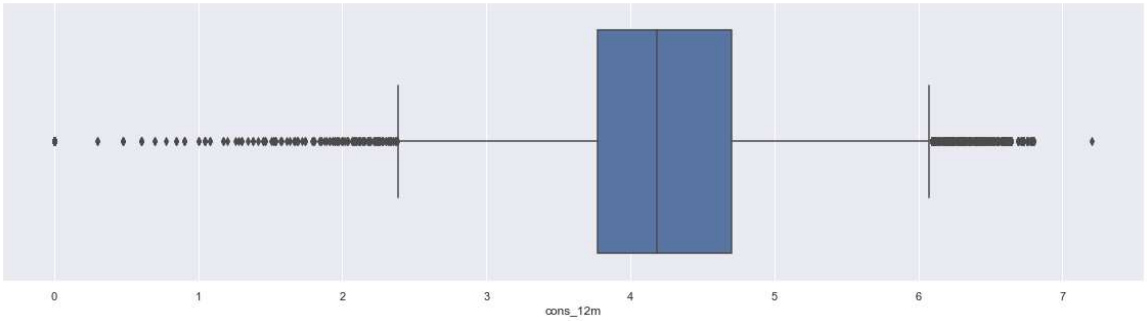
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `histp
lot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [49]:

```
fig = plt.subplots( figsize=(20,5))
# Plot boxplots
sns.boxplot((train_new["cons_12m"].dropna()))
plt.show()
```

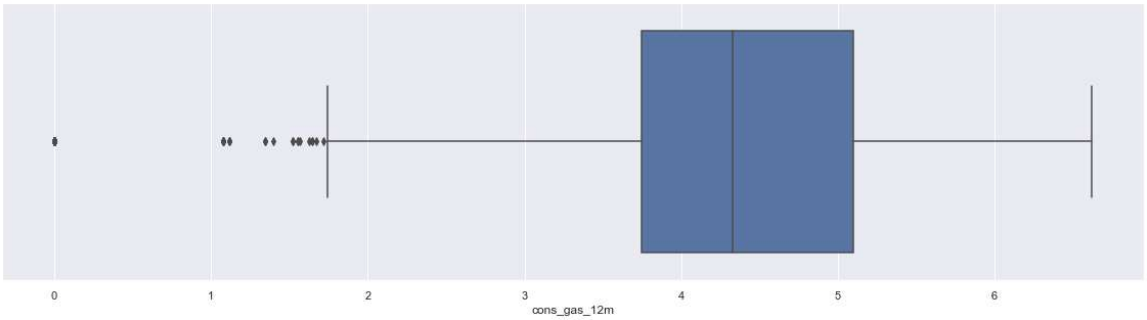
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:4
3: FutureWarning: Pass the following variable as a keyword arg: x. F
rom version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```



In [50]:

```
fig = plt.subplots( figsize=(20,5))
sns.boxplot((train_new[train_new["has_gas"]==1]["cons_gas_12m"].dropna()))
plt.show()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:4
3: FutureWarning: Pass the following variable as a keyword arg: x. F
rom version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```

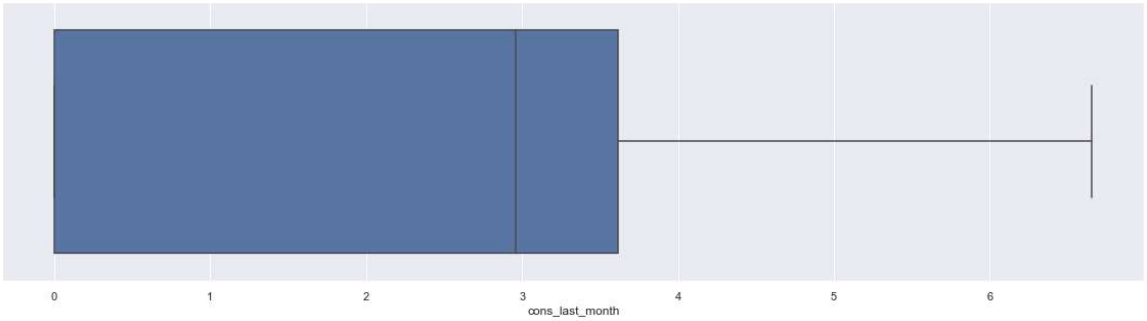


In [51]:

```
fig = plt.subplots( figsize=(20,5))
sns.boxplot((train_new["cons_last_month"].dropna()))

plt.show()
```

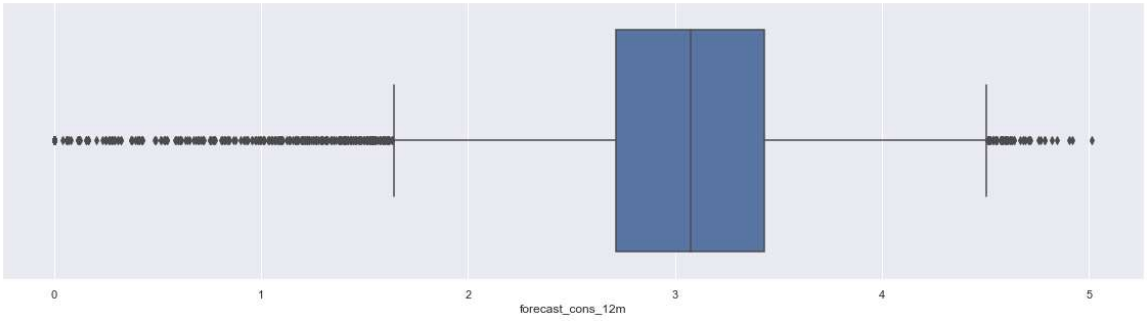
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [52]:

```
fig = plt.subplots( figsize=(20,5))
sns.boxplot((train_new["forecast_cons_12m"].dropna()))
plt.show()
```

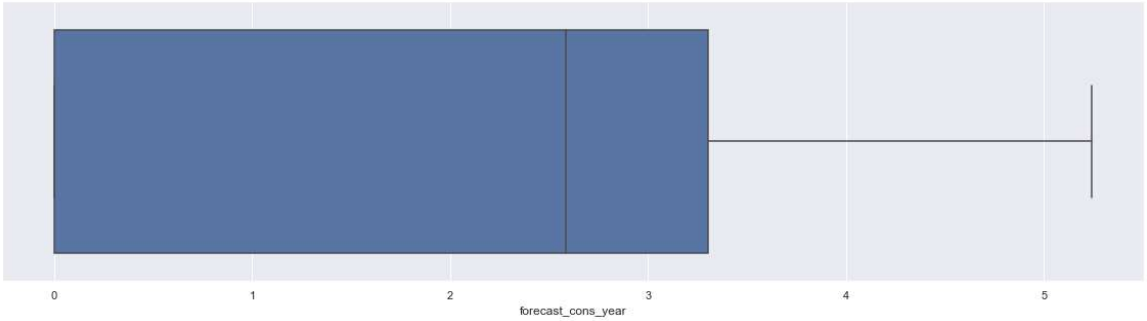
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [53]:

```
fig = plt.subplots( figsize=(20,5))
sns.boxplot((train_new["forecast_cons_year"].dropna()))
plt.show()
```

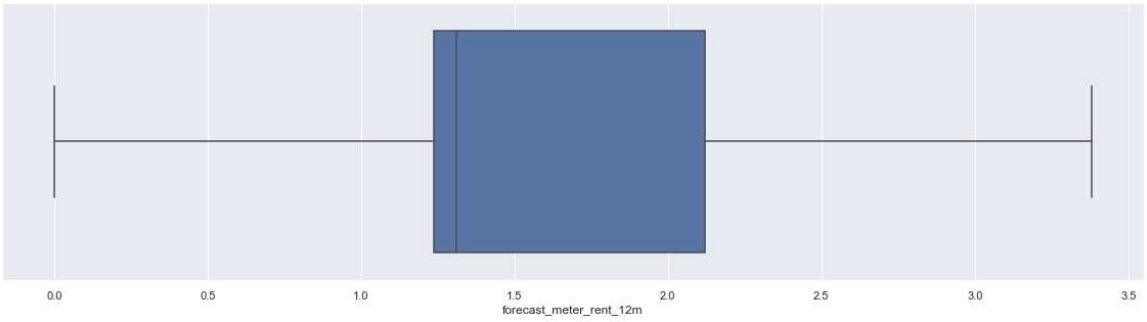
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [54]:

```
fig = plt.subplots( figsize=(20,5))
sns.boxplot((train_new["forecast_meter_rent_12m"].dropna()))
plt.show()
```

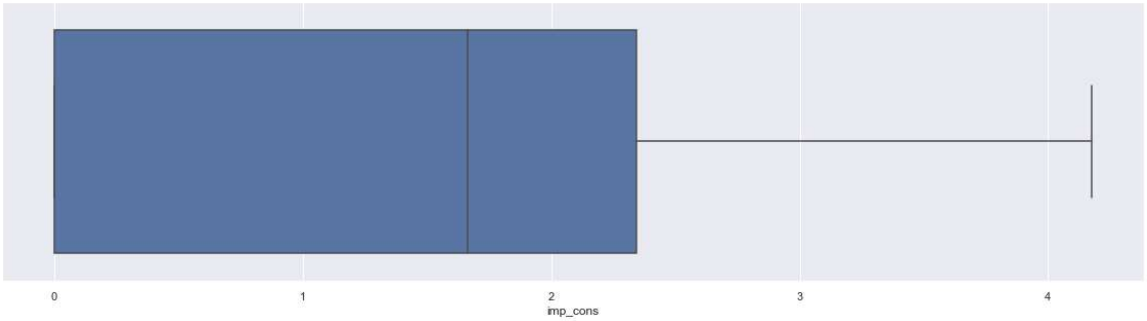
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [55]:

```
fig = plt.subplots( figsize=(20,5))
sns.boxplot((train_new["imp_cons"].dropna()))
plt.show()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [56]:

```
train_new.describe()
```

Out[56]:

	Unnamed: 0	campaign_disc_ele	cons_12m	cons_gas_12m	cons_last_month	fore
count	16096.000000	0.0	16069.000000	16090.000000	16050.000000	
mean	8047.500000	NaN	4.283812	0.800300	2.359281	
std	4646.659302	NaN	0.915265	1.748833	1.789067	
min	0.000000	NaN	0.000000	0.000000	0.000000	
25%	4023.750000	NaN	3.773786	0.000000	0.000000	
50%	8047.500000	NaN	4.187408	0.000000	2.959041	
75%	12071.250000	NaN	4.701508	0.000000	3.617000	
max	16095.000000	NaN	7.206748	6.622052	6.656933	

High correlation Variables

In [57]:

```
correlation = features.corr()
```

In [58]:

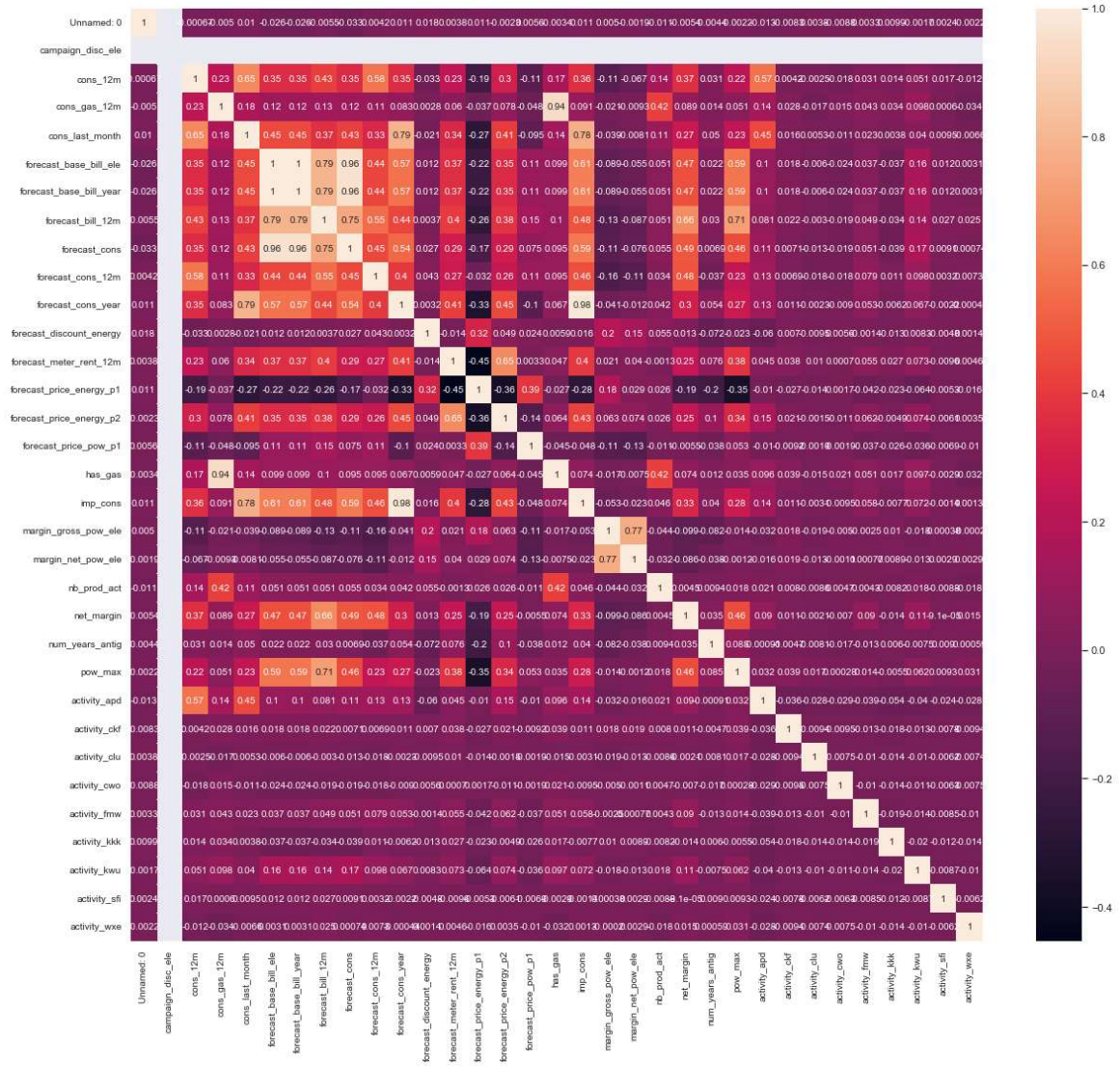
```
# Plot correlation
plt.figure(figsize=(19,15))
sns.heatmap(correlation, xticklabels=correlation.columns.values,
 yticklabels=correlation.columns.values, annot = True, annot_kws={'size':10})
# Axis ticks size
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

In [59]:

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

In [60]:

```
# 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()
```



In [61]:

```
train_new.drop(columns=["num_years_antig", "forecast_cons_year"],inplace=True)
```

Removing Outliers

In [62]:

```
def replace_outliers_z_score(dataframe, column, Z=3):
    """
    Replace outliers with the mean values using the Z score.
    Nan values are also replaced with the mean values.
    Parameters
    ----------
    dataframe : pandas dataframe
        Contains the data where the outliers are to be found
    column : str
        Usually a string with the name of the column

    Returns
    -------
    Dataframe
    With outliers under the lower and above the upper bound removed
    """
    from scipy.stats import zscore

    df = dataframe.copy(deep=True)
    df.dropna(inplace=True, subset=[column])

    # Calculate mean without outliers
    df["zscore"] = zscore(df[column])
    mean_ = df[(df["zscore"] > -Z) & (df["zscore"] < Z)][column].mean()

    # Replace with mean values
    dataframe[column] = dataframe[column].fillna(mean_)
    dataframe["zscore"] = zscore(dataframe[column])
    no_outliers = dataframe[(dataframe["zscore"] < -Z) | (dataframe["zscore"] >
Z)].shape[0]
    dataframe.loc[(dataframe["zscore"] < -Z) | (dataframe["zscore"] > Z),column]
= mean_

    # Print message
    print("Replaced:", no_outliers, " outliers in ", column)
    return dataframe.drop(columns="zscore")
```

In [63]:

```
for c in features.columns:
    if c != "id":
        features = replace_outliers_z_score(features,c)
```

```
Replaced: 276  outliers in  mean_year_price_p1_var
Replaced: 0  outliers in  mean_year_price_p2_var
Replaced: 0  outliers in  mean_year_price_p3_var
Replaced: 120  outliers in  mean_year_price_p1_fix
Replaced: 0  outliers in  mean_year_price_p2_fix
Replaced: 0  outliers in  mean_year_price_p3_fix
Replaced: 122  outliers in  mean_year_price_p1
Replaced: 0  outliers in  mean_year_price_p2
Replaced: 0  outliers in  mean_year_price_p3
```

In [64]:

```
features.reset_index(drop=True, inplace=True)
```

In [65]:

```
def _find_outliers_iqr(dataframe, column):
    """
    Find outliers using the 1.5*IQR rule.
    Parameters
    ----------
    dataframe : pandas dataframe
        Contains the data where the outliers are to be found
    column : str
        Usually a string with the name of the column

    Returns
    -------
    Dict
        With the values of the iqr, lower_bound and upper_bound
    """
    col = sorted(dataframe[column])
    q1, q3= np.percentile(col,[25,75])
    iqr = q3 - q1
    lower_bound = q1 -(1.5 * iqr)
    upper_bound = q3 +(1.5 * iqr)
    results = {"iqr": iqr, "lower_bound":lower_bound, "upper_bound":upper_bound}
    return results
```

In [66]:

```
def remove_outliers_iqr(dataframe, column):
    """
    Remove outliers using the 1.5*IQR rule.
    Parameters
    ----------
    dataframe : pandas dataframe
        Contains the data where the outliers are to be found
    column : str
        Usually a string with the name of the column

    Returns
    -------
    Dataframe
    With outliers under the lower and above the upper bound removed
    """

    outliers = _find_outliers_iqr(dataframe, column)
    removed = dataframe[(dataframe[column] < outliers["lower_bound"]) | (datafra
me[column] > outliers["upper_bound"])].shape

    dataframe = dataframe[(dataframe[column] > outliers["lower_bound"]) & (dataf
rame[column] < outliers["upper_bound"])]
    print("Removed:", removed[0], " outliers")
    return dataframe
```

In [67]:

```
def remove_outliers_z_score(dataframe, column, Z=3):
    """
    Remove outliers using the Z score. Values with more than 3 are removed.
    Parameters
    ----------
    dataframe : pandas dataframe
        Contains the data where the outliers are to be found
    column : str
        Usually a string with the name of the column

    Returns
    -------
    Dataframe
    With outliers under the lower and above the upper bound removed
    """
    from scipy.stats import zscore

    dataframe["zscore"] = zscore(dataframe[column])

    removed = dataframe[(dataframe["zscore"] < -Z) | (dataframe["zscore"] > Z)].
shape

    dataframe = dataframe[(dataframe["zscore"] > -Z) & (dataframe["zscore"] < Z
)]

    print("Removed:", removed[0], " outliers of ", column)

    return dataframe.drop(columns="zscore")
```

In [68]:

```
def replace_outliers_z_score(dataframe, column, Z=3):
    """
    Replace outliers with the mean values using the Z score.
    Nan values are also replaced with the mean values.
    Parameters
    ----------
    dataframe : pandas dataframe
        Contains the data where the outliers are to be found
    column : str
        Usually a string with the name of the column

    Returns
    -------
    Dataframe
    With outliers under the lower and above the upper bound removed
    """
    from scipy.stats import zscore

    df = dataframe.copy(deep=True)
    df.dropna(inplace=True, subset=[column])

    # Calculate mean without outliers
    df["zscore"] = zscore(df[column])
    mean_ = df[(df["zscore"] > -Z) & (df["zscore"] < Z)][column].mean()

    # Replace with mean values
    no_outliers = dataframe[column].isnull().sum()
    dataframe[column] = dataframe[column].fillna(mean_)
    dataframe["zscore"] = zscore(dataframe[column])
    dataframe.loc[(dataframe["zscore"] < -Z) | (dataframe["zscore"] > Z),column]
= mean_

    # Print message
    print("Replaced:", no_outliers, " outliers in ", column)
    return dataframe.drop(columns="zscore")
```

In [ ]:

```

```

In [69]:

```
train = replace_outliers_z_score(train_new,"cons_12m")
train = replace_outliers_z_score(train_new,"cons_gas_12m")
train = replace_outliers_z_score(train_new,"cons_last_month")
train = replace_outliers_z_score(train_new,"forecast_cons_12m")
#train = replace_outliers_z_score(train,"forecast_cons_year")
train = replace_outliers_z_score(train_new,"forecast_discount_energy")
train = replace_outliers_z_score(train_new,"forecast_meter_rent_12m")
train = replace_outliers_z_score(train_new,"forecast_price_energy_p1")
train = replace_outliers_z_score(train_new,"forecast_price_energy_p2")
train = replace_outliers_z_score(train_new,"forecast_price_pow_p1")
train = replace_outliers_z_score(train_new,"imp_cons")
train = replace_outliers_z_score(train_new,"margin_gross_pow_ele")
train = replace_outliers_z_score(train_new,"margin_net_pow_ele")
train = replace_outliers_z_score(train_new,"net_margin")
train = replace_outliers_z_score(train_new,"pow_max")
```

```
Replaced: 27  outliers in  cons_12m
Replaced: 6  outliers in  cons_gas_12m
Replaced: 46  outliers in  cons_last_month
Replaced: 41  outliers in  forecast_cons_12m
Replaced: 126  outliers in  forecast_discount_energy
Replaced: 4  outliers in  forecast_meter_rent_12m
Replaced: 126  outliers in  forecast_price_energy_p1
Replaced: 126  outliers in  forecast_price_energy_p2
Replaced: 126  outliers in  forecast_price_pow_p1
Replaced: 27  outliers in  imp_cons
Replaced: 13  outliers in  margin_gross_pow_ele
Replaced: 13  outliers in  margin_net_pow_ele
Replaced: 15  outliers in  net_margin
Replaced: 3  outliers in  pow_max
```

Pickling

In [ ]:

In [ ]:

In [ ]: