BCG Task 2

Arnav Singh

Exploratory Data Analysis&Data Cleaning

1. Gathering Data
2. Assessing Data
3. Cleaning Data

Gathering data

```
#Import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
import pickle
```

```
#Loading data
train_data=pd.read_csv('ml_case_training_data.csv')
history_data=pd.read_csv('ml_case_training_hist_data.csv')
churn_data=pd.read_csv('ml_case_training_output.csv')
```

```
#Show the first 5 rows of data
train_data.head()
```

	id	activity_new	campaign_disc_ele	
0	48ada52261e7cf58715202705a0451c9	esoiiifxdlbkcsluxmfuacbdckommixw	NaN	lmkeba
1	24011ae4ebbe3035111d65fa7c15bc57	NaN	NaN	foos
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	NaN	
3	764c75f661154dac3a6c254cd082ea7d	NaN	NaN	foos
4	bba03439a292a1e166f80264c16191cb	NaN	NaN	lmkeba

5 rows × 32 columns

```
history_data.head()
```

	id	price_date	price_p1_var	price_p2_var	price_p3_var	pı
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.0	
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.0	
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.0	
3	038af19179925da21a25619c5a24b745	2015-04-01	0.149626	0.0	0.0	
4	038af19179925da21a25619c5a24b745	2015-05-01	0.149626	0.0	0.0	

```
churn_data.head()
```

	id	churn
0	48ada52261e7cf58715202705a0451c9	0

```
#merge the train_data and churn_data into one dataframe
train=pd.merge(train_data,churn_data, on="id")
train.head()
```

	id	activity_new	campaign_disc_ele	
0	48ada52261e7cf58715202705a0451c9	esoiiifxdlbkcsluxmfuacbdckommixw	NaN	lmkeba
1	24011ae4ebbe3035111d65fa7c15bc57	NaN	NaN	foos
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	NaN	
3	764c75f661154dac3a6c254cd082ea7d	NaN	NaN	foos
4	bba03439a292a1e166f80264c16191cb	NaN	NaN	lmkeba

5 rows × 33 columns

Accessing Data

```
#See the datatype of train data
train.dtypes
```

```
id                           object
activity_new                 object
campaign_disc_ele           float64
channel_sales                object
cons_12m                      int64
cons_gas_12m                  int64
cons_last_month               int64
date_activ                   object
date_end                     object
date_first_activ             object
date_modif_prod              object
date_renewal                 object
forecast_base_bill_ele      float64
forecast_base_bill_year     float64
forecast_bill_12m           float64
forecast_cons               float64
forecast_cons_12m           float64
forecast_cons_year            int64
forecast_discount_energy    float64
forecast_meter_rent_12m     float64
forecast_price_energy_p1    float64
forecast_price_energy_p2    float64
forecast_price_pow_p1       float64
has_gas                      object
imp_cons                    float64
margin_gross_pow_ele        float64
margin_net_pow_ele          float64
nb_prod_act                   int64
net_margin                  float64
num_years_antig               int64
origin_up                    object
pow_max                     float64
churn                         int64
dtype: object
```

```
history_data.dtypes
```

```
id               object
price_date       object
price_p1_var    float64
price_p2_var    float64
price_p3_var    float64
price_p1_fix    float64
price_p2_fix    float64
price_p3_fix    float64
dtype: object
```

```
#See the shape of dataset
train.shape
```

```
(16096, 33)
```

```
history_data.shape
```

```
(193002, 8)
```

```
#See the general descriptive statistics of data
train.describe()
```

	campaign_disc_ele	cons_12m	cons_gas_12m	cons_last_month	forecast_base_bill_el
count	0.0	1.609600e+04	1.609600e+04	1.609600e+04	3508.0000(
mean	NaN	1.948044e+05	3.191164e+04	1.946154e+04	335.84385
std	NaN	6.795151e+05	1.775885e+05	8.235676e+04	649.4060(
min	NaN	-1.252760e+05	-3.037000e+03	-9.138600e+04	-364.9400(
25%	NaN	5.906250e+03	0.000000e+00	0.000000e+00	0.0000(
50%	NaN	1.533250e+04	0.000000e+00	9.010000e+02	162.9550(
75%	NaN	5.022150e+04	0.000000e+00	4.127000e+03	396.1850(
max	NaN	1.609711e+07	4.188440e+06	4.538720e+06	12566.0800(

8 rows × 23 columns

It's seems that the campaign_disc_lel is an empty column

```
history_data.describe()
```

	price_p1_var	price_p2_var	price_p3_var	price_p1_fix	price_p2_fix	price_p3_fi
count	191643.000000	191643.000000	191643.000000	191643.000000	191643.000000	191643.00000(
mean	0.140991	0.054412	0.030712	43.325546	10.698201	6.45543(
std	0.025117	0.050033	0.036335	5.437952	12.856046	7.78227(
min	0.000000	0.000000	0.000000	-0.177779	-0.097752	-0.06517
25%	0.125976	0.000000	0.000000	40.728885	0.000000	0.00000(
50%	0.146033	0.085483	0.000000	44.266930	0.000000	0.00000(
75%	0.151635	0.101780	0.072558	44.444710	24.339581	16.22638(
max	0.280700	0.229788	0.114102	59.444710	36.490692	17.45822

```
#See The missing data of train
train.isnull().sum()/train.shape[0]
```

```
id                          0.000000
activity_new                0.593004
campaign_disc_ele           1.000000
channel_sales               0.262053
cons_12m                    0.000000
cons_gas_12m                0.000000
cons_last_month             0.000000
date_activ                  0.000000
date_end                    0.000124
date_first_activ            0.782058
date_modif_prod             0.009754
date_renewal                0.002485
forecast_base_bill_ele      0.782058
forecast_base_bill_year     0.782058
forecast_bill_12m           0.782058
forecast_cons               0.782058
forecast_cons_12m           0.000000
forecast_cons_year          0.000000
forecast_discount_energy    0.007828
forecast_meter_rent_12m     0.000000
forecast_price_energy_p1    0.007828
forecast_price_energy_p2    0.007828
forecast_price_pow_p1       0.007828
has_gas                     0.000000
imp_cons                    0.000000
margin_gross_pow_ele        0.000808
margin_net_pow_ele          0.000808
nb_prod_act                 0.000000
net_margin                  0.000932
num_years_antig             0.000000
origin_up                   0.005405
pow_max                     0.000186
```

```
churn                    0.000000
dtype: float64
```

As we can see that some of columns have missing data over 50%, we need to clean them in the later

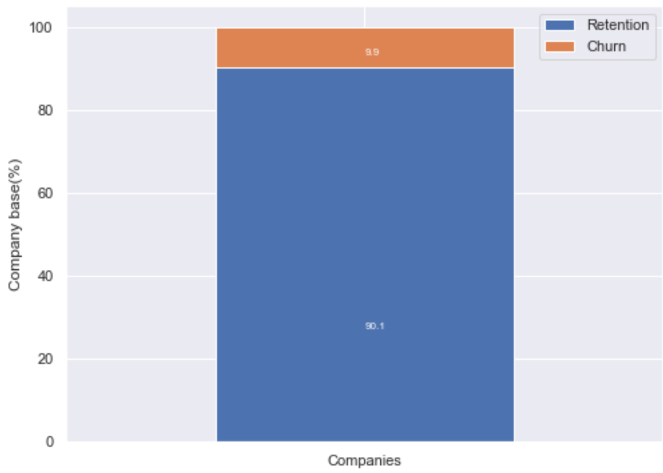
```
history_data.isnull().sum()/history_data.shape[0]
```

```
id              0.000000
price_date      0.000000
price_p1_var    0.007041
price_p2_var    0.007041
price_p3_var    0.007041
price_p1_fix    0.007041
price_p2_fix    0.007041
price_p3_fix    0.007041
dtype: float64
```

```
#Deep diving on the main parameters,first for the Churn
churn=train[['id','churn']]
churn.columns=['Companies','churn']
```

```
churn_total=churn.groupby(churn['churn']).count()
churn_percentage=churn_total/churn_total.sum()*100
```

```
ax=churn_percentage.transpose().plot(kind='bar',stacked=True,figsize=(8,6),rot=0)
for p in ax.patches:
    value=str(round(p.get_height(),1))
    if value=='0':
        countinue
    ax.annotate(value,((p.get_x()+p.get_width()/2)*0.5,p.get_y()+p.get_height()/2*0.6),
               color='white',size=(8))
plt.legend(['Retention','Churn'],loc="upper right")
plt.ylabel("Company base(%)");
```



About 10% of total customers have chruned

```
#Next see the acitivity distribution
activity=train[['id','activity_new','churn']]
activity=activity.groupby([activity['activity_new'],activity['churn']])['id'].count().unstack(level=1).sort_values(by=[0],ascending=False
```

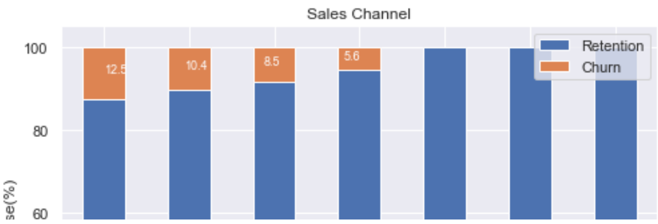
```
activity.plot(kind='bar',figsize=(12,10),width=2,stacked=True,title="SME Activity")
plt.ylabel("Number of Companies")
plt.xlabel('Activity')
plt.legend(['Retention','Churn'],loc="upper right")
plt.xticks([])
plt.show()
```

The xticks is not showing to facilitate the visualization and the distribution of the activity is despite the lack of 60% of the entries

```
activity_total=activity.fillna(0)[0]+activity.fillna(0)[1]
activity_percentage=activity.fillna(0)[1]/(activity_total)*100
pd.DataFrame({'Percentage churn':activity_percentage,
             'Total companies':activity_total}).sort_values(by='Percentage churn',ascending=False).head()
```

activity_new	Percentage churn	Total companies
xwkaesbkfsacseixxksofpddwfkbobki	100.0	1.0
wkwdccuiboaeaalcaawlwmldiwmpewma	100.0	1.0
ikiucmkuisupefxcxfxxulkpwssppfuo	100.0	1.0
opoiuuwdmxdssidluooopfswlkkkcsxf	100.0	1.0
pfcocskbxlmofswiflsbcefcpufbopuo	100.0	2.0

```
#Now is about Sales channel
channel=train[['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)


ax=channel_churn.plot(kind='bar',stacked=True,figsize=(8,6),rot=45)
for p in ax.patches:
    value=str(round(p.get_height(),1))
    if value=='0':
        countinue
    ax.annotate(value,((p.get_x()+p.get_width()/2)*0.94,p.get_y()+p.get_height()/2*0.99),
                color='white',size=(9))
plt.title('Sales Channel')
plt.legend(['Retention','Churn'],loc="upper right")
plt.ylabel("Company base(%)");
```

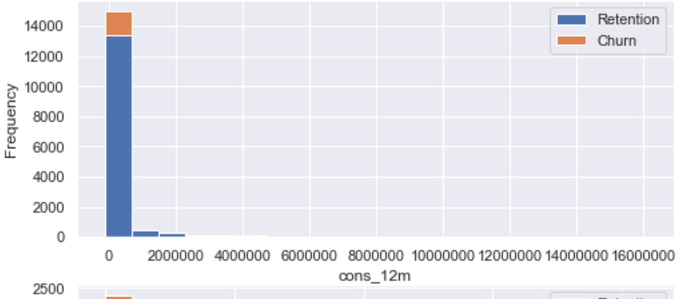


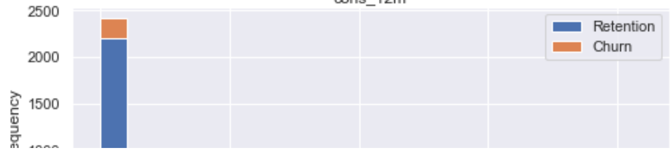

```
channel_total=channel.fillna(0)[0]+channel.fillna(0)[1]
channel_percentage=channel.fillna(0)[1]/(channel_total)*100
pd.DataFrame({"Churn percentage":channel_percentage,
             "Total companies":channel_total}).sort_values(by='Churn percentage',ascending=False).head()
```

channel_sales	Churn percentage	Total companies
foosdfpfkusacimwkcsosbicdxkicaua	12.498306	7377.0
usilxuppasemubllopkaafesmlibmsdf	10.387812	1444.0
ewpakwlliwisiwduibdlfmalxowmwpci	8.488613	966.0
lmkebamcaaclubfxadlmueccxoimlema	5.595755	2073.0
epumfxlbckeskwekxbiuasklxalciiuu	0.000000	4.0

channel_sales

```
#Next is the consumption
consumption=train[['id','cons_12m','cons_gas_12m','cons_last_month','imp_cons','has_gas','churn']]


fig,axs=plt.subplots(nrows=4,figsize=(8,15))
cons_12m=pd.DataFrame({'Retention':consumption[consumption['churn']==0]['cons_12m'],
                      'Churn':consumption[consumption['churn']==1]['cons_12m']})
cons_12m[['Retention','Churn']].plot(kind='hist',bins=20,ax=axs[0],stacked=True);
axs[0].set_xlabel('cons_12m')
axs[0].ticklabel_format(style='plain',axis='x')
cons_gas_12m=pd.DataFrame({'Retention':consumption[consumption['has_gas']=='t'][consumption[consumption['has_gas']=="t"]['churn']==0]['cc
                      'Churn':consumption[consumption['has_gas']=='t'][consumption[consumption['has_gas']=='t']['churn']==1]['cons_gas_12
cons_gas_12m[['Retention','Churn']].plot(kind='hist',bins=20,ax=axs[1],stacked=True);
axs[1].set_xlabel('cons_12m')
axs[1].ticklabel_format(style='plain',axis='x')
cons_last_month=pd.DataFrame({'Retention':consumption[consumption['churn']==0]['cons_last_month'],
                      'Churn':consumption[consumption['churn']==1]['cons_last_month']})
cons_last_month[['Retention','Churn']].plot(kind='hist',bins=20,ax=axs[2],stacked=True);
axs[2].set_xlabel('cons_12m')
axs[2].ticklabel_format(style='plain',axis='x')
imp_cons=pd.DataFrame({'Retention':consumption[consumption['churn']==0]['imp_cons'],
                      'Churn':consumption[consumption['churn']==1]['imp_cons']})
imp_cons[['Retention','Churn']].plot(kind='hist',bins=20,ax=axs[3],stacked=True);
axs[3].set_xlabel('imp_cons')
axs[3].ticklabel_format(style='plain',axis='x')
```





The distribution of the consumptions is highly right skewed and has a long tail, we need to check the outliers by use boxplot

```
fig,axs=plt.subplots(nrows=4,figsize=(10,15))
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])
for ax in axs:
    ax.ticklabel_format(style='plain',axis='x')
axs[0].set_xlim(-200000,2000000)
axs[1].set_xlim(-200000,2000000)
axs[2].set_xlim(-20000,100000)
plt.show();
```

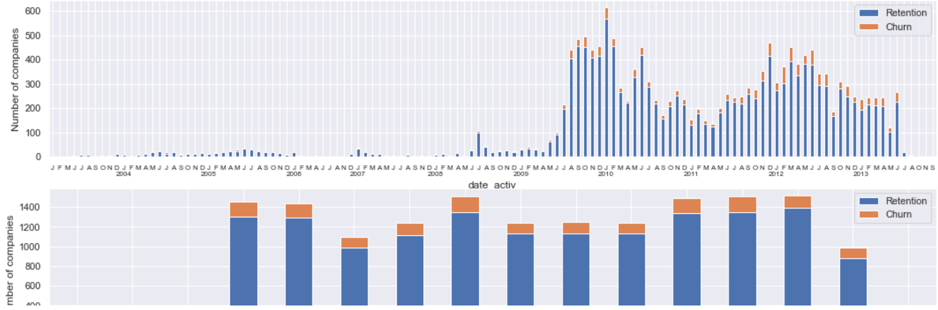
```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

It clearly that we can see the outliers and we will deal with them in the data cleaning

```
#Now is about Dates
dates=train[['id','date_activ','date_end','date_modif_prod','date_renewal','churn']].copy()
dates['date_activ']=pd.to_datetime(dates['date_activ'],format='%Y-%m-%d')
dates['date_end']=pd.to_datetime(dates['date_end'],format='%Y-%m-%d')
dates['date_modif_prod']=pd.to_datetime(dates['date_modif_prod'],format='%Y-%m-%d')
dates['date_renewal']=pd.to_datetime(dates['date_renewal'],format='%Y-%m-%d')
```

```
def line_format(label):
    """
    Convert time label to the format of pandas line plot
    """
    month=label.month_name()[:1]
    if label.month_name()=="January":
        month+=f'\n{label.year}'
    return month
```

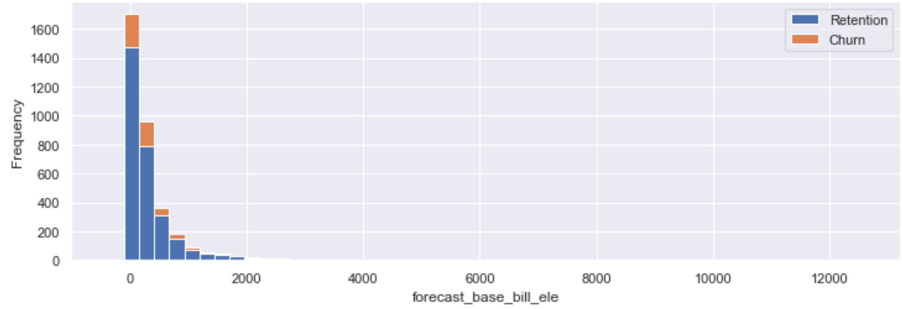
```
fig,axs=plt.subplots(nrows=4,figsize=(18,15))
date_activ=dates[['date_activ','churn','id']].set_index('date_activ').groupby([pd.Grouper(freq='M'),'churn']).count().unstack(level=1)
date_activ.plot(kind='bar',stacked=True,rot=0,ax=axs[0])
axs[0].set_xticklabels(map(lambda x:line_format(x),date_activ.index),fontsize=8)
axs[0].set_ylabel("Number of companies")
axs[0].legend(['Retention','Churn'],loc='upper right')
date_end=dates[['date_end','churn','id']].set_index('date_end').groupby([pd.Grouper(freq='M'),'churn']).count().unstack(level=1)
date_end.plot(kind='bar',stacked=True,rot=0,ax=axs[1])
axs[1].set_xticklabels(map(lambda x:line_format(x),date_end.index),fontsize=8)
axs[1].set_ylabel("Number of companies")
axs[1].legend(['Retention','Churn'],loc='upper right')
date_modif_prod=dates[['date_modif_prod','churn','id']].set_index('date_modif_prod').groupby([pd.Grouper(freq='M'),'churn']).count().unst
date_modif_prod.plot(kind='bar',stacked=True,rot=0,ax=axs[2])
axs[2].set_xticklabels(map(lambda x:line_format(x),date_modif_prod.index),fontsize=8)
axs[2].set_ylabel("Number of companies")
axs[2].legend(['Retention','Churn'],loc='upper right')
date_renewal=dates[['date_renewal','churn','id']].set_index('date_renewal').groupby([pd.Grouper(freq='M'),'churn']).count().unstack(level
date_renewal.plot(kind='bar',stacked=True,rot=0,ax=axs[3])
axs[3].set_xticklabels(map(lambda x:line_format(x),date_renewal.index),fontsize=8)
axs[3].set_ylabel("Number of companies")
axs[3].legend(['Retention','Churn'],loc='upper right');
```

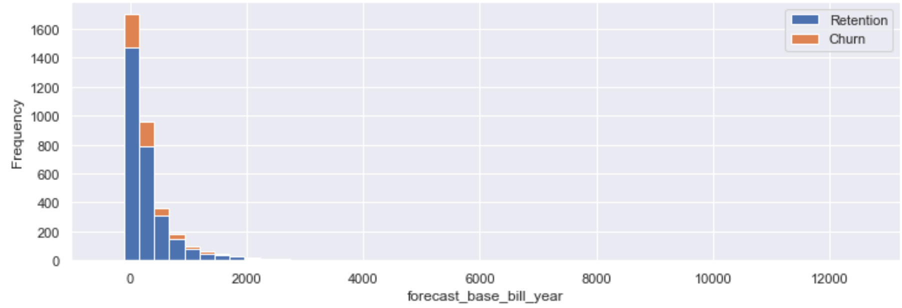


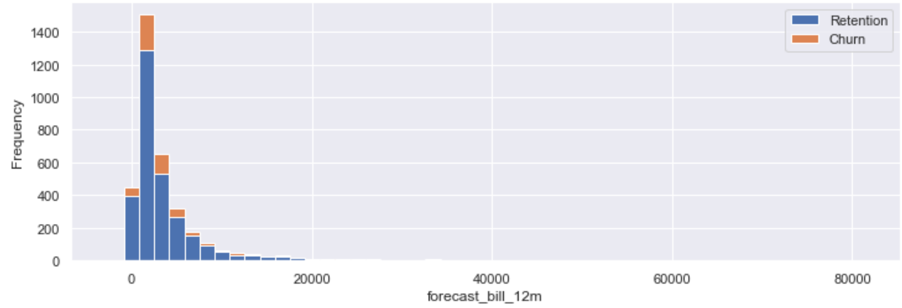
However, the date's distribution seems does not provide any insight.

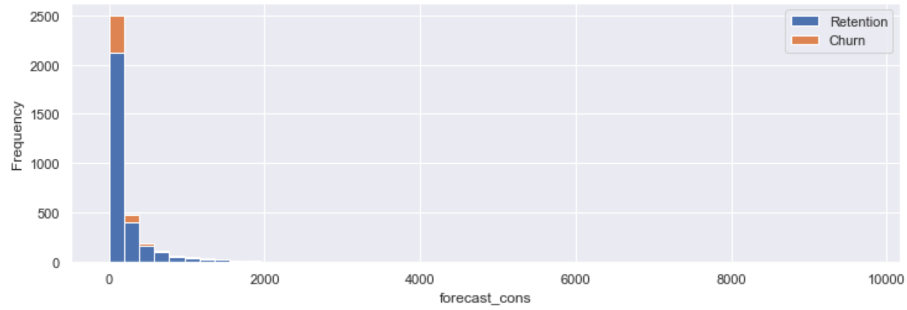
```
#Now is about the forecast
forecast=train[['id','forecast_base_bill_ele','forecast_base_bill_year','forecast_bill_12m','forecast_cons',
                'forecast_cons_12m','forecast_cons_year','forecast_discount_energy','forecast_meter_rent_12m','forecast_price_energy_p1',
```

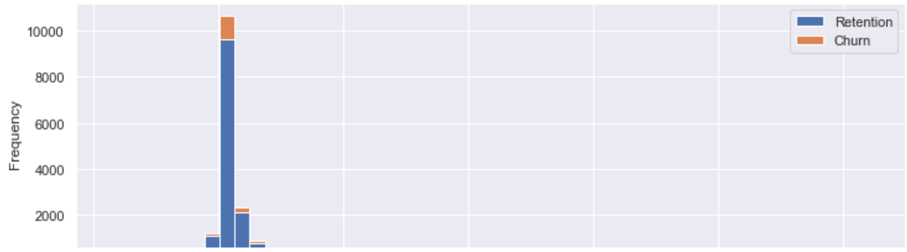
```
fig,axs=plt.subplots(nrows=11,figsize=(12,50))
forecast_base_bill_ele=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_base_bill_ele'],
                      'Churn':train[train['churn']==1]['forecast_base_bill_ele']})
forecast_base_bill_ele[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[0],stacked=True);
axs[0].set_xlabel('forecast_base_bill_ele')
axs[0].ticklabel_format(style='plain',axis='x')
forecast_base_bill_year=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_base_bill_year'],
                      'Churn':train[train['churn']==1]['forecast_base_bill_year']})
forecast_base_bill_year[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[1],stacked=True);
axs[1].set_xlabel('forecast_base_bill_year')
axs[1].ticklabel_format(style='plain',axis='x')
forecast_bill_12m=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_bill_12m'],
                      'Churn':train[train['churn']==1]['forecast_bill_12m']})
forecast_bill_12m[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[2],stacked=True);
axs[2].set_xlabel('forecast_bill_12m')
axs[2].ticklabel_format(style='plain',axis='x')
forecast_cons=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_cons'],
                      'Churn':train[train['churn']==1]['forecast_cons']})
forecast_cons[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[3],stacked=True);
axs[3].set_xlabel('forecast_cons')
axs[3].ticklabel_format(style='plain',axis='x')

forecast_cons_12m=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_cons_12m'],
                      'Churn':train[train['churn']==1]['forecast_cons_12m']})
forecast_cons_12m[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[4],stacked=True);
axs[4].set_xlabel('forecast_cons_12m')
axs[4].ticklabel_format(style='plain',axis='x')
forecast_cons_year=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_cons_year'],
                      'Churn':train[train['churn']==1]['forecast_cons_year']})
forecast_cons_year[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[5],stacked=True);
axs[5].set_xlabel('forecast_cons_year')
axs[5].ticklabel_format(style='plain',axis='x')
forecast_discount_energy=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_discount_energy'],
                      'Churn':train[train['churn']==1]['forecast_discount_energy']})
forecast_discount_energy[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[6],stacked=True);
axs[6].set_xlabel('cons_12m')
axs[6].ticklabel_format(style='plain',axis='x')
forecast_meter_rent_12m=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_meter_rent_12m'],
                      'Churn':train[train['churn']==1]['forecast_meter_rent_12m']})
forecast_meter_rent_12m[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[7],stacked=True);
axs[7].set_xlabel('forecast_meter_rent_12m')
axs[7].ticklabel_format(style='plain',axis='x')
forecast_price_energy_p1=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_price_energy_p1'],
                      'Churn':train[train['churn']==1]['forecast_price_energy_p1']})
forecast_price_energy_p1[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[8],stacked=True);
axs[8].set_xlabel('forecast_price_energy_p1')
axs[8].ticklabel_format(style='plain',axis='x')
forecast_price_energy_p2=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_price_energy_p2'],
                      'Churn':train[train['churn']==1]['forecast_price_energy_p2']})
forecast_price_energy_p2[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[9],stacked=True);
axs[9].set_xlabel('forecast_price_energy_p2')
axs[9].ticklabel_format(style='plain',axis='x')
forecast_price_pow_p1=pd.DataFrame({'Retention':train[train['churn']==0]['forecast_price_pow_p1'],
                      'Churn':train[train['churn']==1]['forecast_price_pow_p1']})
forecast_price_pow_p1[['Retention','Churn']].plot(kind='hist',bins=50,ax=axs[10],stacked=True);
axs[10].set_xlabel('forecast_price_pow_p1')
axs[10].ticklabel_format(style='plain',axis='x')
```


Retention
Churn
Frequency
forecast_base_bill_ele


Retention
Churn
Frequency
forecast_base_bill_year


Retention
Churn
Frequency
forecast_bill_12m

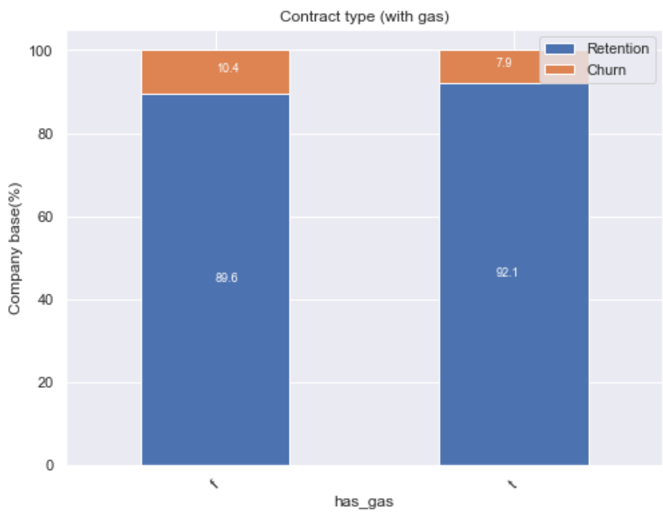

Retention
Churn
Frequency
forecast_cons


Retention
Churn
Frequency

It's similarily to the consumption plots, that lots of variables are highly skewed to the right.

```
#Now is for the contract type(electricity,gas)
contract_type=train[['id','has_gas','churn']]
contract=contract_type.groupby([contract_type['churn'],
                               contract_type['has_gas']])['id'].count().unstack(level=0)
```

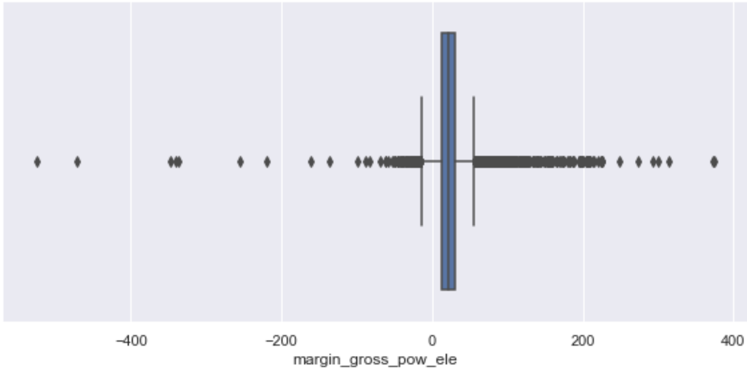
```
contract_percentage=(contract.div(contract.sum(axis=1),axis=0)*100).sort_values(by=[1],ascending=False)
ax=contract_percentage.plot(kind='bar',stacked=True,figsize=(8,6),rot=45)
for p in ax.patches:
    value=str(round(p.get_height(),1))
    if value=='0':
        countinue
    ax.annotate(value,((p.get_x()+p.get_width()/2)*0.94,p.get_y()+p.get_height()/2*0.99),
               color='white',size=(9))
plt.title('Contract type (with gas)')
plt.legend(['Retention','Churn'],loc="upper right")
plt.ylabel("Company base(%)");
```




```
#Now is about Margins
margin=train[['id','margin_gross_pow_ele','margin_net_pow_ele','net_margin']]
```

```
fig,axs=plt.subplots(nrows=3,figsize=(10,15))
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])
for ax in axs:
    ax.ticklabel_format(style='plain',axis='x')
plt.show()
```

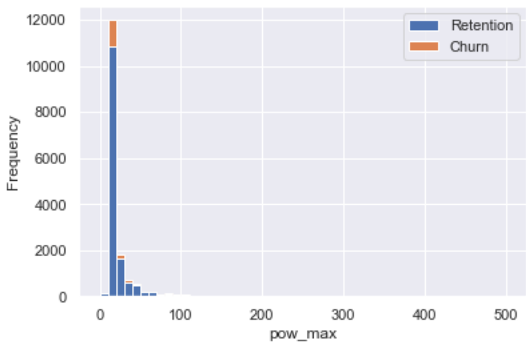
```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```



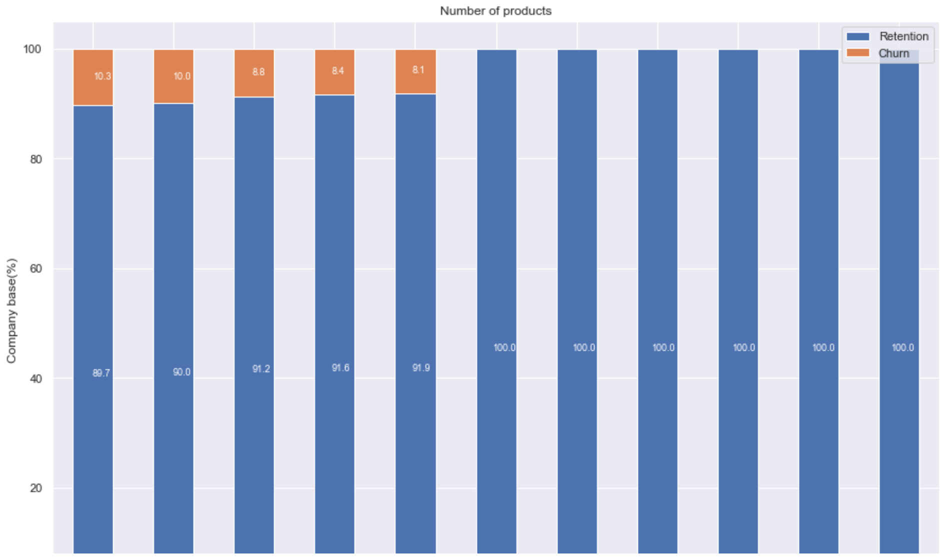
```
#Next is about the Subcribed power
power=train[['id','pow_max','churn']].fillna(0)
```

```
figure=plt.figure()
pow_max=pd.DataFrame({'Retention':power[power['churn']==0]['pow_max'],
                      'Churn':power[power['churn']==1]['pow_max']})
pow_max[['Retention','Churn']].plot(kind='hist',bins=50,stacked=True);
plt.xlabel('pow_max')
plt.ticklabel_format(style='plain',axis='x');
```

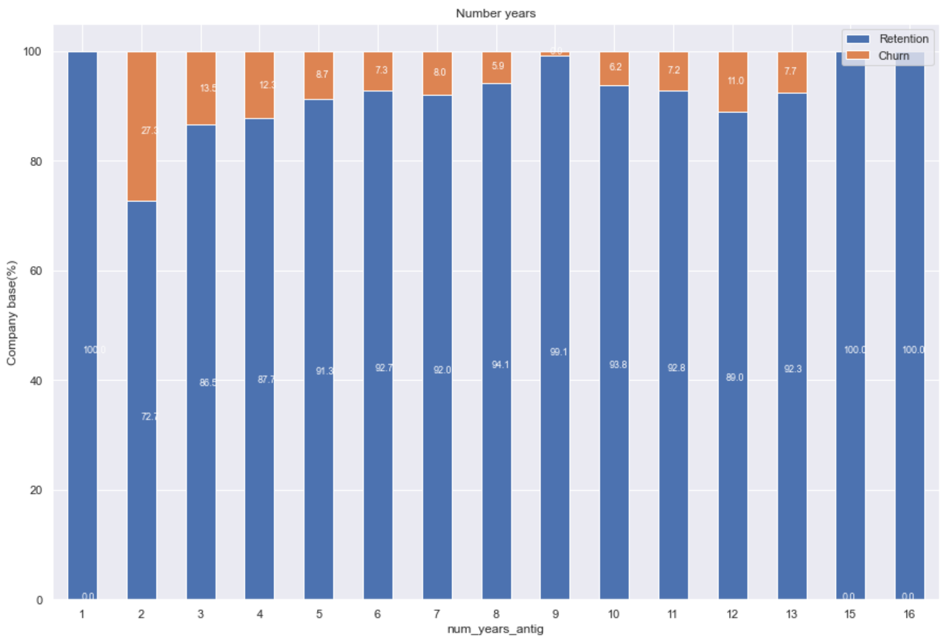
```
<Figure size 432x288 with 0 Axes>
```




```
#Last id for others variables
others=train[['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)


ax=products_percentage.plot(kind='bar',stacked=True,figsize=(15,10),rot=0)
for p in ax.patches:
    value=str(round(p.get_height(),1))
    if value=='0':
        countinue
    ax.annotate(value,((p.get_x()+p.get_width()/2)*0.99,p.get_y()+p.get_height()/2*0.9),
                color='white',size=(9))
plt.title('Number of products')
plt.legend(['Retention','Churn'],loc="upper right")
plt.ylabel("Company base(%)");
```

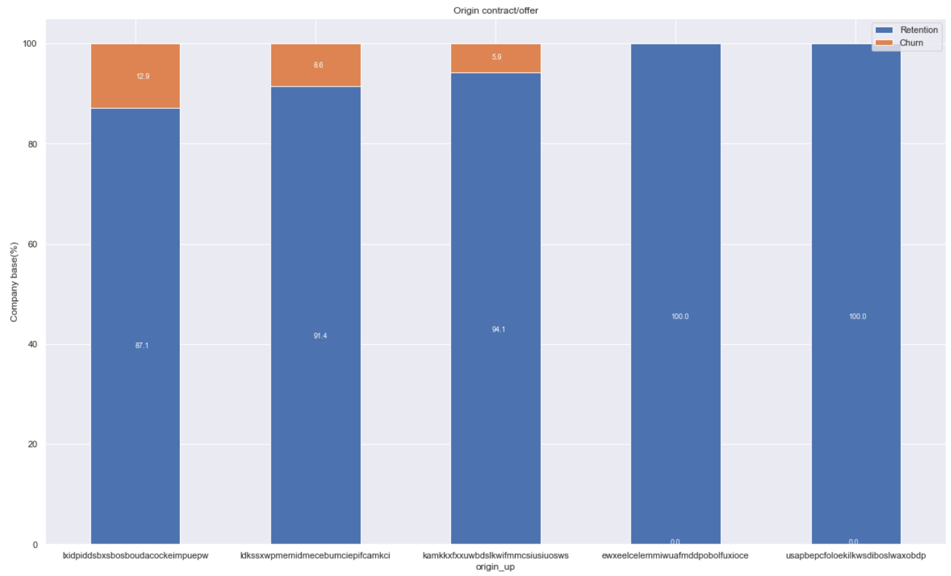



```
years_antig=others.groupby([others['num_years_antig'],others['churn']])['id'].count().unstack(level=1)
years_antig_percentage=(years_antig.div(years_antig.sum(axis=1),axis=0)*100)

ax=years_antig_percentage.plot(kind='bar',stacked=True,figsize=(15,10),rot=0)
for p in ax.patches:
    value=str(round(p.get_height(),1))
    if value=='0':
        countinue
    ax.annotate(value,((p.get_x()+p.get_width()/2)*0.99,p.get_y()+p.get_height()/2*0.9),
                color='white',size=(9))
plt.title('Number years')
plt.legend(['Retention','Churn'],loc="upper right")
plt.ylabel("Company base(%)");
```



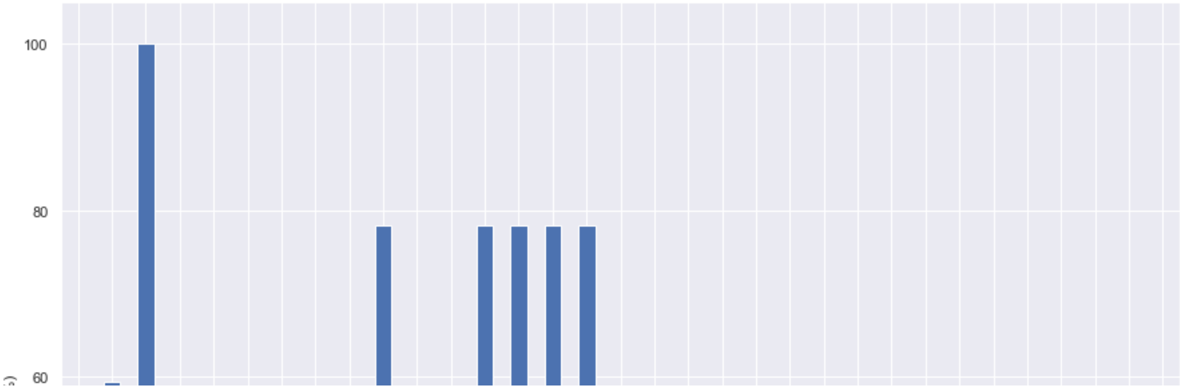

```
origin=others.groupby([others['origin_up'],others['churn']])['id'].count().unstack(level=1)
origin_percentage=(origin.div(origin.sum(axis=1),axis=0)*100).sort_values(by=[1],ascending=False)

ax=origin_percentage.plot(kind='bar',stacked=True,figsize=(20,12),rot=0)
for p in ax.patches:
    value=str(round(p.get_height(),1))
    if value=='0':
        countinue
    ax.annotate(value,((p.get_x()+p.get_width()/2)*0.99,p.get_y()+p.get_height()/2*0.9),
                color='white',size=(9))
```

```
plt.title('Origin contract/offer')
plt.legend(['Retention','Churn'],loc="upper right")
plt.ylabel("Company base(%)");
```



Data Cleaning

```
#plot the missing data
plt.figure(figsize=(15,12))
(train.isnull().sum()/len(train.index)*100).plot(kind='bar')
plt.xlabel('Variables')
plt.ylabel('Missing values(%)')
plt.show()
```



From the figure above, we can remove the variables that more than 60% values missing

```
train.drop(columns=['campaign_disc_ele','date_first_activ','forecast_base_bill_ele','forecast_base_bill_year','forecast_bill_12m','foreca
```

```
#Check The removed dataframe
pd.DataFrame({'Dataframe columns':train.columns})
```

	Dataframe columns
0	id
1	channel_sales
2	cons_12m
3	cons_gas_12m
4	cons_last_month
5	date_activ
6	date_end
7	date_modif_prod
8	date_renewal
9	forecast_cons_12m
10	forecast_cons_year
11	forecast_discount_energy
12	forecast_meter_rent_12m
13	forecast_price_energy_p1
14	forecast_price_energy_p2
15	forecast_price_pow_p1
16	has_gas
17	imp_cons
18	margin_gross_pow_ele
19	margin_net_pow_ele
20	nb_prod_act
21	net_margin
22	num_years_antig
23	origin_up
24	pow_max
25	churn

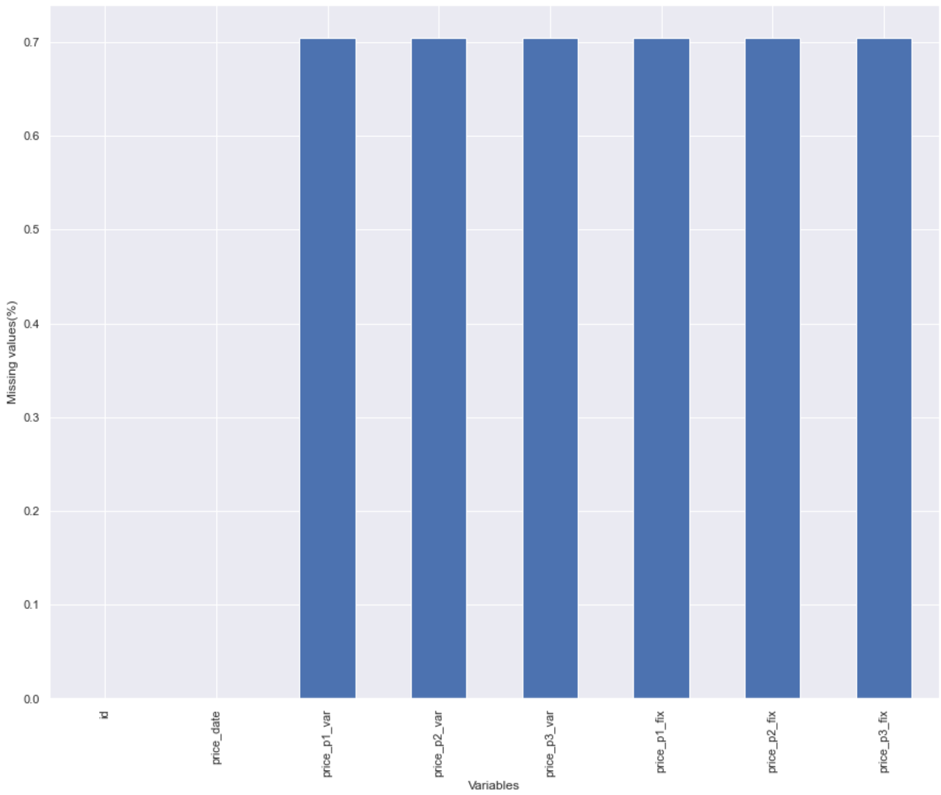
```
#Check the duplicates
train[train.duplicated()]
```

id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modi

0 rows × 26 columns

There seems no duplicated data of the train dataframe

```
#Check the history missing data
missing_data_percentage=history_data.isnull().sum()/len(history_data.index)*100


plt.figure(figsize=(15,12))
missing_data_percentage.plot(kind='bar')
plt.xlabel('Variables')
plt.ylabel('Missing values(%)')
plt.show()
```



There is not much data missing, we will subsitute them with the median in the next step

Formating data

```
#fill the missing date with the median date which use the value_counts()
train.loc[train['date_modif_prod'].isnull(),'date_modif_prod']=train['date_modif_prod'].value_counts().index[0]
train.loc[train['date_end'].isnull(),'date_end']=train['date_end'].value_counts().index[0]
train.loc[train['date_renewal'].isnull(),'date_renewal']=train['date_renewal'].value_counts().index[0]


#fill the price data with median
history_data.loc[history_data['price_p1_var'].isnull(),'price_p1_var']=history_data['price_p1_var'].median()
history_data.loc[history_data['price_p2_var'].isnull(),'price_p2_var']=history_data['price_p2_var'].median()
history_data.loc[history_data['price_p3_var'].isnull(),'price_p3_var']=history_data['price_p3_var'].median()
history_data.loc[history_data['price_p1_fix'].isnull(),'price_p1_fix']=history_data['price_p1_fix'].median()
history_data.loc[history_data['price_p2_fix'].isnull(),'price_p2_fix']=history_data['price_p2_fix'].median()
history_data.loc[history_data['price_p3_fix'].isnull(),'price_p3_fix']=history_data['price_p3_fix'].median()


#fill the negative data of history with median
history_data.loc[history_data['price_p1_fix']<0,'price_p1_fix']=history_data['price_p1_fix'].median()
history_data.loc[history_data['price_p2_fix']<0,'price_p2_fix']=history_data['price_p2_fix'].median()
history_data.loc[history_data['price_p3_fix']<0,'price_p3_fix']=history_data['price_p3_fix'].median()


#Transform date columns to datetime type
train['date_activ']=pd.to_datetime(train['date_activ'],format='%Y-%m-%d')
train['date_end']=pd.to_datetime(train['date_end'],format='%Y-%m-%d')
train['date_modif_prod']=pd.to_datetime(train['date_modif_prod'],format='%Y-%m-%d')
```

```
train['date_renewal']=pd.to_datetime(train['date_renewal'],format='%Y-%m-%d')
history_data['price_date']=pd.to_datetime(history_data['price_date'],format='%Y-%m-%d')
```

Saving data

```
#Make directly processed_data if it does not exist
train.to_csv('train_clean.csv', index = False)
history_data.to_csv('history_clean.csv', index = False)
```