BCG Task 3

Arnav Singh

Sub-task 1: Think through what key drivers of churn could be for our client

Sub-task 2: Build the features in order to get ready to model

Import packages

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import pandas as pd
import seaborn as sns
import datetime
import pickle
import warnings
warnings.filterwarnings("ignore")
```

```
sns.set(color_codes=True)
pd.set_option('display.max_columns',50)
```

Load data

```
date_cols=['date_activ','date_end','date_modif_prod','date_renewal']
train = pd.read_csv('train_clean.csv',parse_dates=date_cols)
train.head()
```

	id	channel_sales	cons_12m	cons_gas_12m
0	48ada52261e7cf58715202705a0451c9	lmkebamcaaclubfxadlmueccxoimlema	309275	0
1	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	4660	0
3	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0
4	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0

```
history = pd.read_csv('history_clean.csv',parse_dates=['price_date'])
history.head()
```

	id	price_date	price_p1_var	price_p2_var	price_p3_var	pr
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	

Feature engineering

First we need to do the Feature Selection We will create the average consumption of the year as one new feature

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

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

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

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


#Combine the mean year in a single dataframe
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_year_price_p1_fix']
mean_year['mean_year_price_p2']=mean_year['mean_year_price_p2_var']+mean_year['mean_year_price_p2_fix']
mean_year['mean_year_price_p3']=mean_year['mean_year_price_p3_var']+mean_year['mean_year_price_p3_fix']


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_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']


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_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']


history_new = pd.merge(mean_year,mean_6m, on='id',how='left')
history_new = pd.merge(mean_year,mean_3m, on='id',how='left')
history_new.head()
```

	id	mean_year_price_p1_var	mean_year_price_p2_var	mean_ye
0	0002203ffbb812588b632b9e628cc38d	0.124338	0.103794	
1	0004351ebdd665e6ee664792efc4fd13	0.146426	0.000000	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.181558	0.000000	
3	0010ee3855fdea87602a5b7aba8e42de	0.118757	0.098292	
4	00114d74e963e47177db89bc70108537	0.147926	0.000000	

Datetime

```
#Extract contract duration
#we will define the duration=date_end-date_activ
train['contract_duration']=((train['date_end']-train['date_activ'])/ np.timedelta64(1,'M')).astype(int)
train.head()
```

	id	channel_sales	cons_12m	cons_gas_12m
0	48ada52261e7cf58715202705a0451c9	lmkebamcaaclubfxadlmueccxoimlema	309275	0
1	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	4660	0
3	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0
4	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0

```
#set the reference time to be 2016-01-01
#write a finction to caculate the month difference between datetime features
```

```
def calculatemonth(referencetime,dataframe,column):
    time_diff=referencetime-dataframe[column]
    months=(time_diff/np.timedelta64(1,'M')).astype(int)
    return months
```

```
referencetime=pd.to_datetime('2016-01-01')
```

```
train['activ_diff']=calculatemonth(referencetime,train,'date_activ')
train['end_diff']=calculatemonth(referencetime,train,'date_end')
train['modif_diff']=calculatemonth(referencetime,train,'date_modif_prod')
train['renewal_diff']=calculatemonth(referencetime,train,'date_renewal')
train.head()
```

	id	channel_sales	cons_12m	cons_gas_12m
0	48ada52261e7cf58715202705a0451c9	lmkebamcaaclubfxadlmueccxoimlema	309275	0
1	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	4660	0
3	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0
4	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0

```
#Remove the date columns
train.drop(columns=['date_activ','date_end','date_modif_prod','date_renewal'],axis=1,inplace=True)
```

```
train.head()
```

	id	channel_sales	cons_12m	cons_gas_12m
0	48ada52261e7cf58715202705a0451c9	lmkebamcaaclubfxadlmueccxoimlema	309275	0
1	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946
2	d29c2c54acc38ff3c0614d0a653813dd	NaN	4660	0
3	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0
4	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0

Categorical Data

Binary encoding

```
#For the column has_gas,replace t for 1 and f for 0
train['has_gas']=train['has_gas'].replace(['t','f'],[1,0])
```

one-hot encoding

```
train['channel_sales']=train['channel_sales'].fillna('null_values_channel')
train['channel_sales']=train['channel_sales'].apply(lambda x:x[:4])
categories_channel=pd.get_dummies(train[['channel_sales']])
categories_channel.drop(columns=['channel_sales_null'],inplace=True)
categories_channel.head()
```

	channel_sales_epum	channel_sales_ewpa	channel_sales_fixd	channel_sales_foos	cl
0	0	0	0	0	
1	0	0	0	1	
2	0	0	0	0	
3	0	0	0	1	
4	0	0	0	0	

```
#for the column origin_up, first fill the null value
train['origin_up']=train['origin_up'].fillna('null_values_origin')
```

```
train['origin_up']=train['origin_up'].apply(lambda x:x[:4])
categories_origin= pd.get_dummies(train[['origin_up']])
categories_origin.drop(columns=['origin_up_null'],inplace=True)
categories_origin.head()
```

	origin_up_ewxe	origin_up_kamk	origin_up_ldks	origin_up_lxid	origin_up_usap
0	0	0	1	0	0
1	0	0	0	1	0
2	0	1	0	0	0
3	0	1	0	0	0
4	0	1	0	0	0

```
#Use the 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=train.drop(['channel_sales','origin_up'],axis=1)
train.head()
```

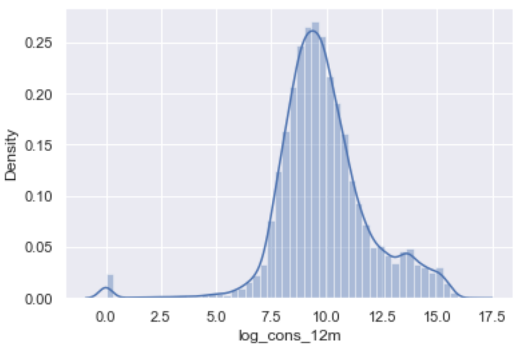
	id	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_:
0	48ada52261e7cf58715202705a0451c9	309275	0	10025	26520
1	24011ae4ebbe3035111d65fa7c15bc57	0	54946	0	0
2	d29c2c54acc38ff3c0614d0a653813dd	4660	0	0	189
3	764c75f661154dac3a6c254cd082ea7d	544	0	0	47
4	bba03439a292a1e166f80264c16191cb	1584	0	0	240

Numerical Data

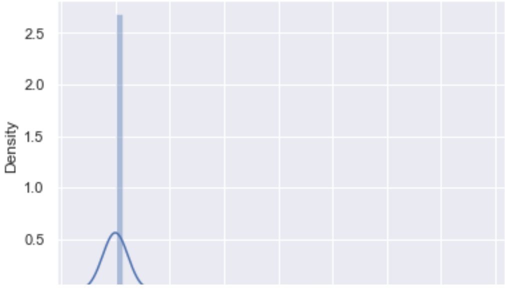
Distribution transformation

From the previous EDA we can see that some features are highly skewed, we need to transform the distribution to normal-like distribution

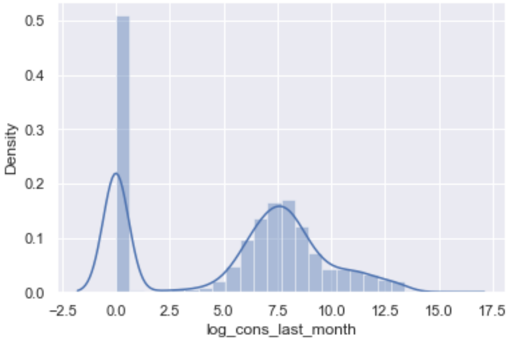
```
#First for the cons_12, remove the negative values and apply a log tranformation
train.loc[train.cons_12m<0,'cons_12m']=np.nan
train['cons_12m']=train['cons_12m'].dropna()
train['log_cons_12m']=train['cons_12m'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_cons_12m']);
```



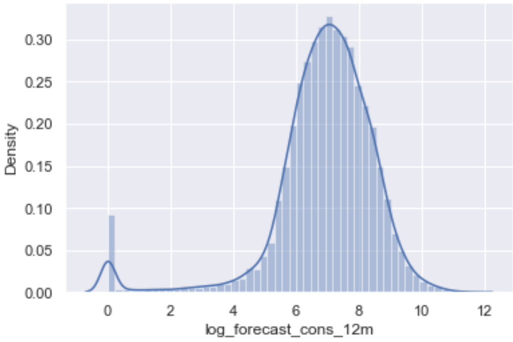

```
train.loc[train.cons_gas_12m<0,'cons_gas_12m']=np.nan
train['cons_gas_12m']=train['cons_gas_12m'].dropna()
train['log_cons_gas_12m']=train['cons_gas_12m'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_cons_gas_12m']);
```



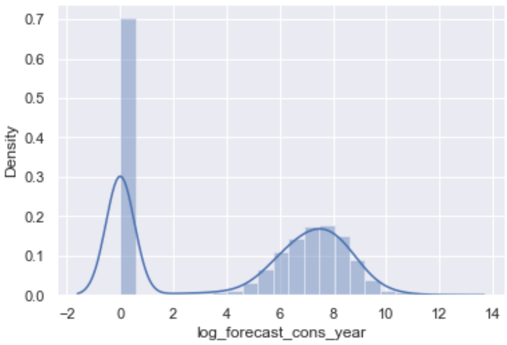

```
train.loc[train.cons_last_month<0,'cons_last_month']=np.nan
train['cons_last_month']=train['cons_last_month'].dropna()
train['log_cons_last_month']=train['cons_last_month'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_cons_last_month']);
```




```
train.loc[train.forecast_cons_12m<0,'forecast_cons_12m']=np.nan
train['forecast_cons_12m']=train['forecast_cons_12m'].dropna()
train['log_forecast_cons_12m']=train['forecast_cons_12m'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_forecast_cons_12m']);
```

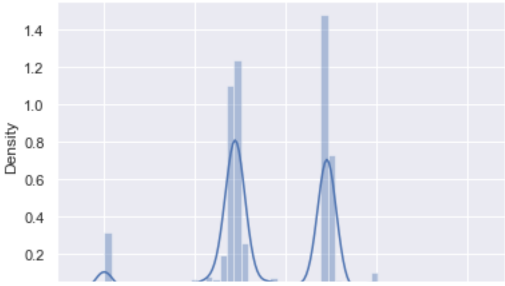



```
train.loc[train.forecast_cons_year<0,'forecast_cons_year']=np.nan
train['forecast_cons_year']=train['forecast_cons_year'].dropna()
train['log_forecast_cons_year']=train['forecast_cons_year'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_forecast_cons_year']);
```

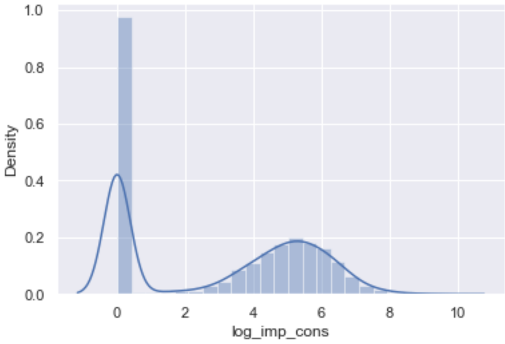



```
train.loc[train.forecast_meter_rent_12m<0,'forecast_meter_rent_12m']=np.nan
train['forecast_meter_rent_12m']=train['forecast_meter_rent_12m'].dropna()
train['log_forecast_meter_rent_12m']=train['forecast_meter_rent_12m'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_forecast_meter_rent_12m'])
```

```
<AxesSubplot:xlabel='log_forecast_meter_rent_12m', ylabel='Density'>
```




```
train.loc[train.imp_cons<0,'imp_cons']=np.nan
train['imp_cons']=train['imp_cons'].dropna()
train['log_imp_cons']=train['imp_cons'].apply(lambda x:np.log(1+x))
sns.distplot(train['log_imp_cons']);
```



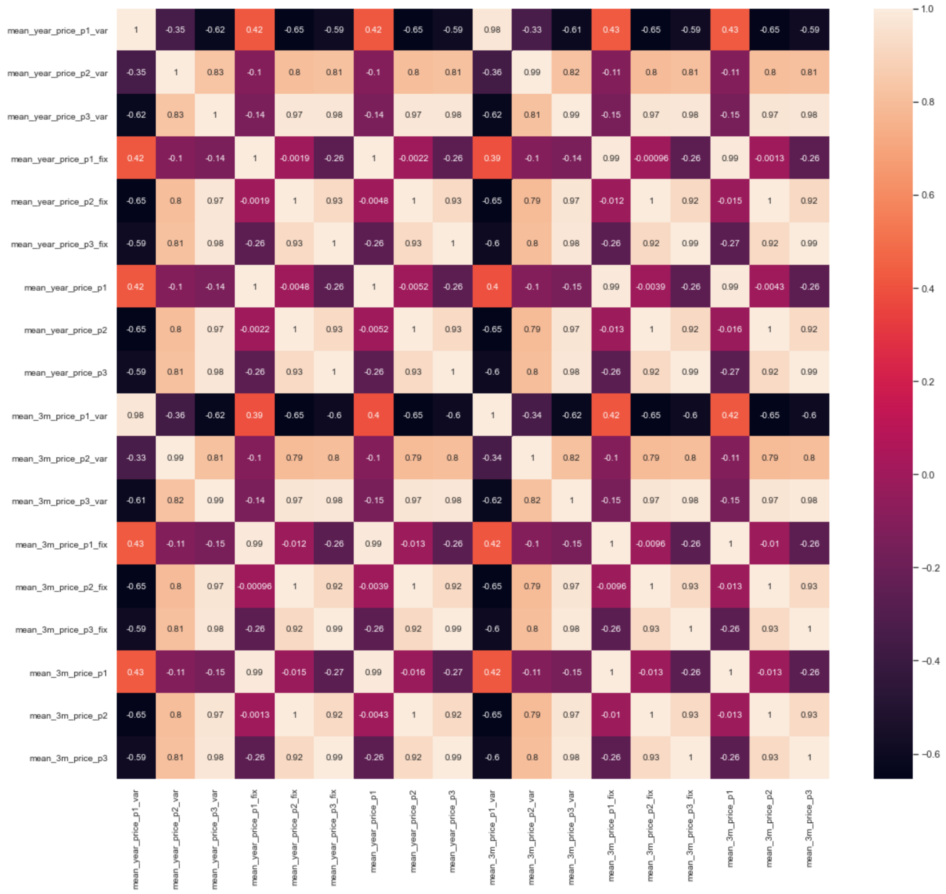

```
train=train.drop(['cons_12m','cons_gas_12m','cons_last_month','forecast_cons_12m','forecast_cons_year','forecast_meter_rent_12m','imp_cor
train.head()
```

	id	forecast_discount_energy	forecast_price_energy_p1	for
0	48ada52261e7cf58715202705a0451c9	0.0	0.095919	
1	24011ae4ebbe3035111d65fa7c15bc57	0.0	0.114481	
2	d29c2c54acc38ff3c0614d0a653813dd	0.0	0.145711	
3	764c75f661154dac3a6c254cd082ea7d	0.0	0.165794	
4	bba03439a292a1e166f80264c16191cb	0.0	0.146694	

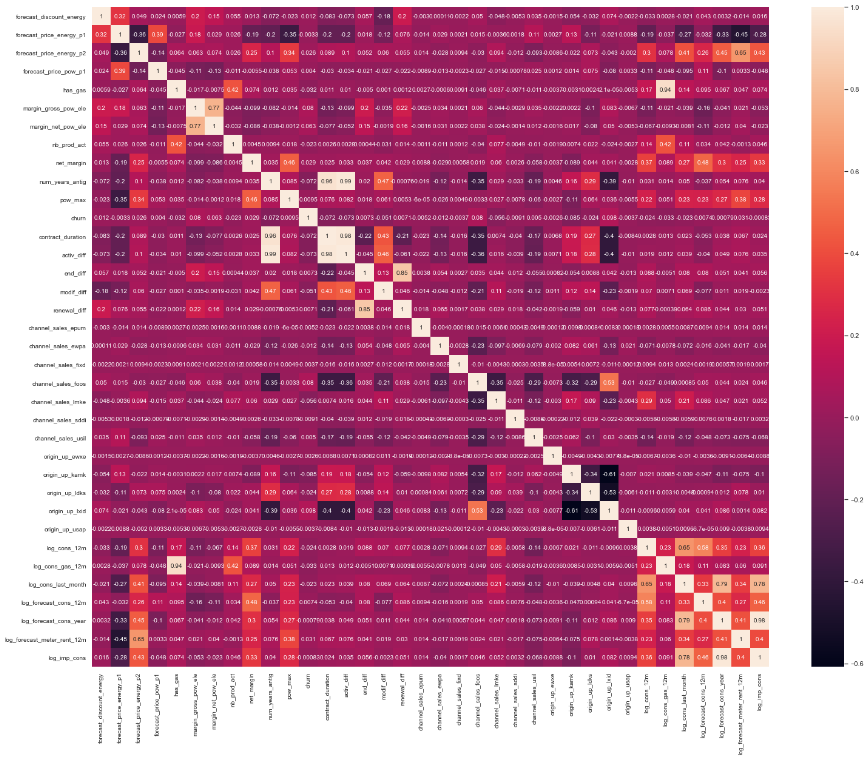
High correlation variables

```
#Calculate correlation of variables
corr_hist=history_new.corr()
```

```
plt.figure(figsize=(18,16))
sns.heatmap(corr_hist,xticklabels=corr_hist.columns.values,
           yticklabels=corr_hist.columns.values,annot=True,annot_kws={'size':10})
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



```
#Calculate correlation of variables
corr_train=train.corr()


plt.figure(figsize=(25,20))
sns.heatmap(corr_train,xticklabels=corr_train.columns.values,
            yticklabels=corr_train.columns.values,annot=True,annot_kws={'size':10})
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



As expected, num_years_antig has a high correlation with activ_diff, we can remove the num_years_antig since they are provides the same information.

```
train.drop(columns=['num_years_antig'],inplace=True)
```

Removing Outliers

As the previous EDA we can see that there have several outliers in the dataset,for simplisity, I will replace these outliers with the mean.

```
#I will use IQR method to detect outliers
def remove_outliers(df,col):
    df.loc[df[col]<0,col]=df[col].mean()
    q1=df[col].quantile(.25)
    q3=df[col].quantile(.75)
    iqr=q3-q1
    upper_bound=q3+(iqr*1.5)
    lower_bound=q1-(iqr*1.5)
    df=df[(lower_bound<df[col])|(df[col]<upper_bound)]
```

```
remove_outliers(history_new,'mean_year_price_p1_var')
remove_outliers(history_new,'mean_year_price_p2_var')
remove_outliers(history_new,'mean_year_price_p3_var')
remove_outliers(history_new,'mean_year_price_p1_fix')
remove_outliers(history_new,'mean_year_price_p2_fix')
remove_outliers(history_new,'mean_year_price_p3_fix')
remove_outliers(history_new,'mean_year_price_p1')
remove_outliers(history_new,'mean_year_price_p2')
remove_outliers(history_new,'mean_year_price_p3')
remove_outliers(train,'log_cons_12m')
remove_outliers(train,'log_cons_gas_12m')
remove_outliers(train,'log_cons_last_month')
remove_outliers(train,'log_forecast_cons_12m')
remove_outliers(train,'log_forecast_meter_rent_12m')
remove_outliers(train,'log_forecast_cons_year')
remove_outliers(train,'log_imp_cons')
remove_outliers(train,'forecast_discount_energy')
remove_outliers(train,'forecast_price_energy_p1')
remove_outliers(train,'forecast_price_energy_p2')
remove_outliers(train,'forecast_price_pow_p1')
remove_outliers(train,'margin_gross_pow_ele')
remove_outliers(train,'margin_net_pow_ele')
remove_outliers(train,'net_margin')
remove_outliers(train,'pow_max')
remove_outliers(train,'forecast_price_energy_p1')
```

Merge Data Together

```
df = pd.merge(train, history_new, on='id', how = 'left')
df.head()
```

	id	forecast_discount_energy	forecast_price_energy_p1	for
0	48ada52261e7cf58715202705a0451c9	0.0	0.095919	
1	24011ae4ebbe3035111d65fa7c15bc57	0.0	0.114481	
2	d29c2c54acc38ff3c0614d0a653813dd	0.0	0.145711	
3	764c75f661154dac3a6c254cd082ea7d	0.0	0.165794	
4	bba03439a292a1e166f80264c16191cb	0.0	0.146694	

5 rows × 54 columns

```
df =df.dropna()
```

```
df.isnull().sum().sort_values(ascending = False)
```

```
mean_3m_price_p3              0
activ_diff                    0
origin_up_ewxe                0
channel_sales_usil            0
channel_sales_sddi            0
channel_sales_lmke            0
channel_sales_foos            0
channel_sales_fixd            0
channel_sales_ewpa            0
channel_sales_epum            0
renewal_diff                  0
modif_diff                    0
end_diff                      0
contract_duration             0
mean_3m_price_p2              0
churn                         0
pow_max                       0
net_margin                    0
nb_prod_act                   0
margin_net_pow_ele            0
margin_gross_pow_ele          0
has_gas                       0
forecast_price_pow_p1         0
forecast_price_energy_p2      0
forecast_price_energy_p1      0
forecast_discount_energy      0
origin_up_kamk                0
origin_up_ldks                0
origin_up_lxid                0
origin_up_usap                0
mean_3m_price_p1              0
mean_3m_price_p3_fix          0
mean_3m_price_p2_fix          0
mean_3m_price_p1_fix          0
mean_3m_price_p3_var          0
mean_3m_price_p2_var          0
mean_3m_price_p1_var          0
mean_year_price_p3            0
mean_year_price_p2            0
mean_year_price_p1            0
mean_year_price_p3_fix        0
mean_year_price_p2_fix        0
mean_year_price_p1_fix        0
mean_year_price_p3_var        0
mean_year_price_p2_var        0
mean_year_price_p1_var        0
log_imp_cons                  0
log_forecast_meter_rent_12m   0
log_forecast_cons_year        0
log_forecast_cons_12m         0
log_cons_last_month           0
log_cons_gas_12m              0
log_cons_12m                  0
id                            0
dtype: int64
```

```
df=df.drop('id', axis = 1)
```

```
df.to_csv('feature_engineering.csv', index = False)
```

Modeling & Evaluation

Sub-Task 1:Build churn model(s) to try to predict the churn probability of any customer.

Sub-Task 2:Evaluate your model, using a holdout set, and with metrics of your choosing.

Sub-Task 3:Interpret the results and use them to formulate answers to the client's hypotheses and questions.

Import packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
sns.set(color_codes=True)

pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
```

Load Data

```
df=pd.read_csv("feature_engineering.csv")
df.head()
```

Saving...

	forecast_discount_energy	forecast_price_energy_p1	forecast_price_energy_p2	forecast_pr
[illegible]	[illegible]	0.095919	0.088347	
1	0.0	0.114481	0.098142	
2	0.0	0.145711	0.000000	
3	0.0	0.165794	0.087899	
4	0.0	0.146694	0.000000	

Splitting data

```
#First we need to specify features and target
y=df['churn']
X=df.drop('churn',axis=1)
```

```
#Check the binary target
y.value_counts()
```

```
0    14331
1     1528
Name: churn, dtype: int64
```

As we can see, the y(churn) is imbalanced

```
#Spliting dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

Base Models

We are going to quickly test the fit of 6 different models

- Logistic Regression: basic linear classifier (good to baseline)
- Random Forest: ensemble bagging classifier
- K-Nearest Neighbors: instance based classifier
- Support Vector Machines: maximum margin classifier

- Gaussian Naive Bayes: probabilistic classifier
- XGBoost: ensemble (extreme!) boosting classifier

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.utils import class_weight
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

def run_exps(X_train: pd.DataFrame , y_train: pd.DataFrame, X_test: pd.DataFrame, y_test: pd.DataFrame) -> pd.DataFrame:
    '''
    Lightweight script to test many models and find winners
    :param X_train: training split
    :param y_train: training target vector
    :param X_test: test split
    :param y_test: test target vector
    :return: DataFrame of predictions
    '''

    dfs=[]
    models =[('LogReg', LogisticRegression()),
        ('RF', RandomForestClassifier ()),
        ('KNN', KNeighborsClassifier ()),
        ('SVM', SVC()),
        ('GNB', GaussianNB()),
        ('XGB', XGBClassifier(eval_metric='mlogloss'))
            ]
    results = []
    names = []
    scoring = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted', 'roc_auc']
                                          nign']

        kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=90210)
        cv_results = model_selection.cross_validate(model, X_train, y_train, cv=kfold, scoring=scoring)
        clf = model.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print(name)
        print(classification_report(y_test, y_pred, target_names=target_names))
        results.append(cv_results)
        names.append(name)
        this_df = pd.DataFrame(cv_results)
        this_df['model'] = name
        dfs.append(this_df)
        final = pd.concat(dfs, ignore_index=True)
    return final


run_exps(X_train,y_train,X_test,y_test)
```

```
LogReg
              precision    recall  f1-score   support

   malignant       0.91      1.00      0.95      2871
      benign       0.33      0.00      0.01       301

    accuracy                           0.90      3172
   macro avg       0.62      0.50      0.48      3172
weighted avg       0.85      0.90      0.86      3172

RF
              precision    recall  f1-score   support

   malignant       0.91      1.00      0.95      2871
      benign       0.76      0.05      0.10       301

    accuracy                           0.91      3172
   macro avg       0.84      0.53      0.53      3172
weighted avg       0.90      0.91      0.87      3172

KNN
              precision    recall  f1-score   support

   malignant       0.91      0.99      0.95      2871
      benign       0.22      0.03      0.05       301

    accuracy                           0.90      3172
   macro avg       0.56      0.51      0.50      3172
weighted avg       0.84      0.90      0.86      3172

SVM
              precision    recall  f1-score   support

   malignant       0.91      1.00      0.95      2871
      benign       0.00      0.00      0.00       301

    accuracy                           0.91      3172
   macro avg       0.45      0.50      0.48      3172
weighted avg       0.82      0.91      0.86      3172
```

Saving...

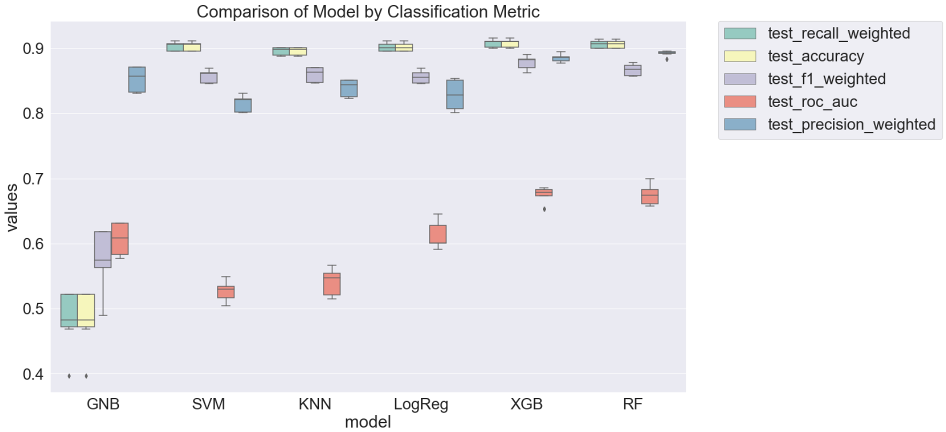
```
              precision    recall  f1-score   support

   malignant       0.94      0.50      0.65      2871
      benign       0.13      0.70      0.21       301

    accuracy                           0.52      3172
   macro avg       0.53      0.60      0.43      3172
weighted avg       0.86      0.52      0.61      3172

XGB
              precision    recall  f1-score   support

   malignant       0.92      0.99      0.95      2871
      benign       0.64      0.12      0.21       301

    accuracy                           0.91      3172
   macro avg       0.78      0.56      0.58      3172
weighted avg       0.89      0.91      0.88      3172
```

	fit_time	score_time	test_accuracy	test_precision_weighted	test_recall_weighted	test_
0	0.181278	0.015472	0.895193	0.801371	0.895193	
1	0.112731	0.009289	0.906225	0.853343	0.906225	
2	0.107510	0.008401	0.911313	0.831177	0.911313	
3	0.113473	0.009013	0.895546	0.824947	0.895546	
4	0.106269	0.008844	0.905400	0.844854	0.905400	
5	2.445273	0.100206	0.899921	0.893368	0.899921	
6	2.435634	0.099173	0.910165	0.893356	0.910165	
7	2.356782	0.124688	0.914466	0.899455	0.914466	
8	2.648319	0.105518	0.900670	0.902254	0.900670	
9	2.366336	0.097486	0.908159	0.891904	0.908159	
10	0.029028	0.461501	0.889283	0.825791	0.889283	
11	0.027826	0.458236	0.896769	0.846220	0.896769	
12	0.031085	0.419473	0.901064	0.850707	0.901064	
13	0.030494	0.422842	0.888057	0.823238	0.888057	
14	0.026901	0.385980	0.900276	0.840897	0.900276	

```
final=run_exps(X_train,y_train,X_test,y_test)
bootstraps = []
for model in list(set(final.model.values)):
    model_df = final.loc[final.model==model]
    bootstrap = model_df.sample(n=30, replace=True)
    bootstraps.append(bootstrap)

bootstrap_df = pd.concat(bootstraps, ignore_index=True)
results_long = pd.melt(bootstrap_df,id_vars=['model'],var_name='metrics', value_name='values')
time_metrics = ['fit_time','score_time'] # fit time metrics
## PERFORMANCE METRICS
results_long_nofit = results_long.loc[~results_long['metrics'].isin(time_metrics)] # get df without fit data
results_long_nofit = results_long_nofit.sort_values(by='values')
## TIME METRICS
results_long_fit = results_long.loc[results_long['metrics'].isin(time_metrics)] # df with fit data
results_long_fit = results_long_fit.sort_values(by='values')
```

```
LogReg
              precision    recall  f1-score   support

   malignant       0.91      1.00      0.95      2871
      benign       0.33      0.00      0.01       301

    accuracy                           0.90      3172
   macro avg       0.62      0.50      0.48      3172
weighted avg       0.85      0.90      0.86      3172

RF
              precision    recall  f1-score   support

   malignant       0.91      1.00      0.95      2871
      benign       0.86      0.06      0.11       301

    accuracy                           0.91      3172
   macro avg       0.88      0.53      0.53      3172
weighted avg       0.91      0.91      0.87      3172

Saving...                         ll  f1-score   support

   malignant       0.91      0.99      0.95      2871
      benign       0.22      0.03      0.05       301

    accuracy                           0.90      3172
   macro avg       0.56      0.51      0.50      3172
weighted avg       0.84      0.90      0.86      3172

SVM
              precision    recall  f1-score   support

   malignant       0.91      1.00      0.95      2871
      benign       0.00      0.00      0.00       301

    accuracy                           0.91      3172
   macro avg       0.45      0.50      0.48      3172
weighted avg       0.82      0.91      0.86      3172

GNB
              precision    recall  f1-score   support

   malignant       0.94      0.50      0.65      2871
      benign       0.13      0.70      0.21       301

    accuracy                           0.52      3172
   macro avg       0.53      0.60      0.43      3172
weighted avg       0.86      0.52      0.61      3172

XGB
              precision    recall  f1-score   support

   malignant       0.92      0.99      0.95      2871
      benign       0.64      0.12      0.21       301

    accuracy                           0.91      3172
   macro avg       0.78      0.56      0.58      3172
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20, 12))
sns.set(font_scale=2.5)
g = sns.boxplot(x="model", y="values", hue="metrics", data=results_long_nofit, palette="Set3")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Comparison of Model by Classification Metric')
plt.savefig('./benchmark_models_performance.png',dpi=300)
```



Saving...

It is clearly that GNBs fit our data poorly across alomast all the metrics, and the XGBoost and Random Forest fit the data very well

```
plt.figure(figsize=(20, 12))
sns.set(font_scale=2.5)
g = sns.boxplot(x="model", y="values", hue="metrics", data=results_long_fit, palette="Set3")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Comparison of Model by Fit and Score Time')
plt.savefig('./benchmark_models_time.png',dpi=300)
```

From the figure above we can see that the SVM is slow to train and score,for the Random Forest and XGBoost, the XGBoost has faster fit time

```
metrics = list(set(results_long_nofit.metrics.values))
bootstrap_df.groupby(['model'])[metrics].agg([np.std, np.mean])
```

	test_recall_weighted		test_precision_weighted		test_accuracy		test_f1_weighted	
	std	mean	std	mean	std	mean	std	mean
model								
GNB	0.034210	0.492161	0.017498	0.851936	0.034210	0.492161	0.036219	0.585762
KNN	0.005488	0.896126	0.011259	0.839678	0.005488	0.896126	0.009748	0.860121
LogReg	0.006366	0.901387	0.020117	0.829054	0.006366	0.901387	0.009245	0.855589
RF	0.005747	0.906204	0.003423	0.892445	0.005747	0.906204	0.008587	0.867215
SVM	0.006412	0.902569	0.011626	0.814896	0.006412	0.902569	0.009308	0.856484
XGB	0.006330	0.907486	0.006454	0.885163	0.006330	0.907486	0.010578	0.877279

```
time_metrics = list(set(results_long_fit.metrics.values))
bootstrap_df.groupby(['model'])[time_metrics].agg([np.std, np.mean])
```

	score_time		fit_time	
	std	mean	std	mean
GNB	[illegible]	[illegible]	0.000934	0.014204
KNN	0.022762	0.419944	0.002279	0.027204
LogReg	0.001222	0.009340	0.021929	0.119620
RF	0.048464	0.115296	0.186116	2.437337
SVM	0.019577	0.687027	0.087527	2.035073
XGB	0.000952	0.016685	0.130804	2.075427

Saving...

Based on the analysis of six models, I will focus on the XGBoost as continue refining model, not only because it has the best performing but also it has relativily fast train and score time

Model Finetuning

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb


#Create the random grid
params={
    'min_child_weight':[i for i in np.arange(1,15,1)],
    'gamma':[i for i in np.arange(0,6,0.5)],
    'subsample':[i for i in np.arange(0,1.1,0.1)],
    'colsample_bytree':[i for i in np.arange(0,1.1,0.1)],
    'max_depth':[i for i in np.arange(1,15,1)],
    'scale_pos_weight':[i for i in np.arange(0,0.15,0.01)],
    'learning_rate':[i for i in np.arange(0,0.15,0.01)],
    'n_estimators':[i for i in np.arange(0,2000,100)],
}


#Create model
xg=xgb.XGBClassifier(objective='binary:logistic',nthread=1,eval_metric='mlogloss')
```

```
#Random search of parameters,using 5
xg_random=RandomizedSearchCV(xg,param_distributions=params,
                            n_iter=1,scoring='roc_auc',
                            n_jobs=4,cv=5,verbose=3,random_state=1001)
xg_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   2 out of   5 | elapsed:   10.5s remaining:   15.8s
[Parallel(n_jobs=4)]: Done   5 out of   5 | elapsed:   13.6s finished
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           eval_metric='mlogloss', gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints...
                                        'n_estimators': [0, 100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100, 1200,
                                                         1300, 1400, 1500, 1600,
                                                         1700, 1800, 1900],
                                        'scale_pos_weight': [0.0, 0.01, 0.02,
                                                             0.03, 0.04, 0.05,
                                                             0.06, 0.07, 0.08,
                                                             0.09, 0.1, 0.11,
                                                             0.12, 0.13, 0.14],
                                        'subsample': [0.0, 0.1, 0.2,
                                                      0.30000000000000004, 0.4,
                                                      0.5, 0.6000000000000001,
                                                      0.7000000000000001, 0.8,
                                                      0.9, 1.0]},
                   random_state=1001, scoring='roc_auc', verbose=3)
```

Saving...

```
best_random={'subsample':0.8,
'scale_pos_weight':1,
'n_estimators':1100,
'max_depth':12,
'learning_rate':0.01,
'gamma':4.0,
'colsample_bytree':0.60}
```

```
#Create a model with the parameters found
model_random=xgb.XGBClassifier(objective='binary:logistic',
                               nthread=1,eval_metric='mlogloss',**best_random)
```

```
fprs,tprs,score=[],[],[]
```

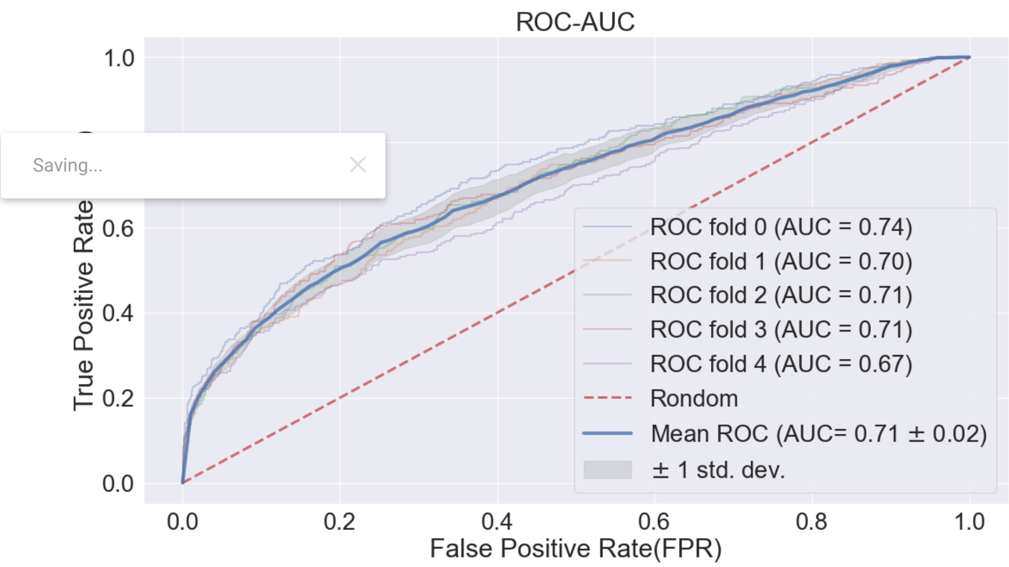
```
from sklearn.model_selection import StratifiedKFold
cv=StratifiedKFold(n_splits=5,random_state=13,shuffle=True)
```

```
from sklearn import metrics
def compute_roc_auc(model_,index):
    y_predict=model_.predict_proba(X.iloc[index])[:,1]
    fpr,tpr,threholds=metrics.roc_curve(y.iloc[index],y_predict)
    auc_score=metrics.auc(fpr,tpr)
    return fpr,tpr,auc_score
```

```
for (train,test), i in zip(cv.split(X,y),range(5)):
    model_random.fit(X.iloc[train],y.iloc[train])
    _,_,auc_score_train=compute_roc_auc(model_random,train)
    fpr,tpr,auc_score=compute_roc_auc(model_random,test)
    score.append((auc_score_train,auc_score))
    fprs.append(fpr)
    tprs.append(tpr)
```

```
def plot_roc_curve(fprs,tprs):
    tprs_interp=[]
    aucs=[]
    mean_fpr=np.linspace(0,1,100)
    f,ax=plt.subplots(figsize=(18,10))

    for i,(fpr,tpr) in enumerate(zip(fprs,tprs)):
        tprs_interp.append(np.interp(mean_fpr,fpr,tpr))
        tprs_interp[-1][0]=0.0
```

```
        roc_auc=metrics.auc(fpr,tpr)
        aucs.append(roc_auc)
        ax.plot(fpr,tpr,lw=2,alpha=0.3,
                label="ROC fold %d (AUC = %0.2f)" % (i, roc_auc))
    plt.plot([0,1],[0,1],linestyle='--',lw=3,color='r',label="Rondom",alpha=.8)
    mean_tpr=np.mean(tprs_interp,axis=0)
    mean_tpr[-1]=1.0
    mean_auc=metrics.auc(mean_fpr,mean_tpr)
    std_auc=np.std(aucs)
    ax.plot(mean_fpr,mean_tpr,color='b',
            label=r"Mean ROC (AUC= %0.2f $\pm$ %0.2f)" % (mean_auc,std_auc),
           lw=4,alpha=.8)
    std_tpr=np.std(tprs_interp,axis=0)
    tprs_upper=np.minimum(mean_tpr+std_tpr,1)
    tprs_lower=np.maximum(mean_tpr-std_tpr,0)

    ax.fill_between(mean_fpr,tprs_lower,tprs_upper,color='grey',
                   label=r'$\pm$ 1 std. dev.',alpha=.2)
    ax.set_xlim([-0.05,1.05])
    ax.set_ylim([-0.05,1.05])
    ax.set_xlabel('False Positive Rate(FPR)')
    ax.set_ylabel('True Positive Rate(TPR)')
    ax.set_title('ROC-AUC')
    ax.legend(loc='lower right')
    plt.show()
    return (f, ax)


plot_roc_curve(fprs,tprs)
plt.show()
```



Grid search with cross validation

```
from sklearn.model_selection import GridSearchCV


param_grid={'subsample':[0.7],
'scale_pos_weight':[1],
'n_estimators':[1100],
'min_child_weight':[1],
'max_depth':[12,13,14],
'learning_rate':[0.005,0.01],
'gamma':[4.0],
'colsample_bytree':[0.6]}
```

```
xg=xgb.XGBClassifier(objective='binary:logistic',eval_metric='mlogloss')


grid_search=GridSearchCV(estimator=xg,param_grid=param_grid,
                            cv=5,n_jobs= -1,verbose=2,scoring='roc_auc')
grid_search.fit(X_train,y_train)
```

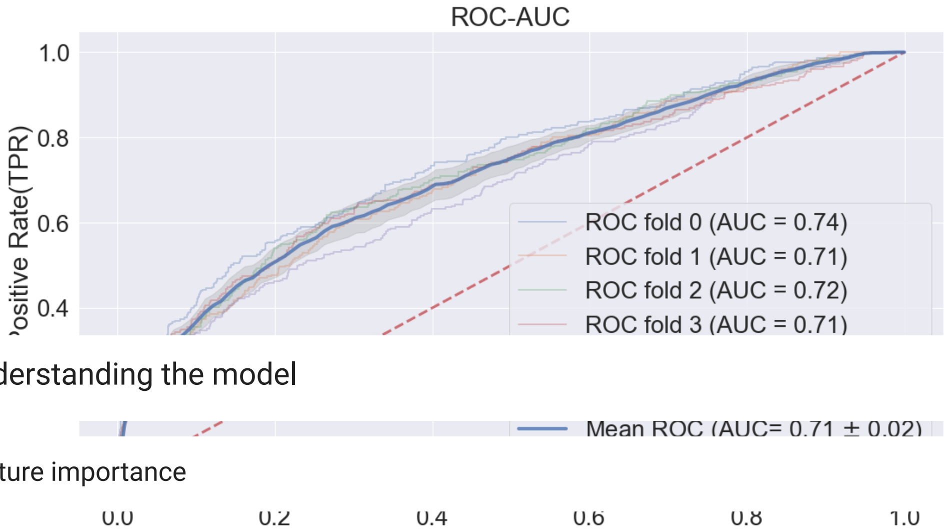
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 19.0min finished
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     eval_metric='mlogloss', gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,...
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, subsample=None,
                                     tree_method=None, validate_parameters=None,
                                     verbosity=None),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.6], 'gamma': [4.0],
                         'learning_rate': [0.005, 0.01],
                         'max_depth': [12, 13, 14], 'min_child_weight': [1],
                         'n_estimators': [1100], 'scale_pos_weight': [1],
                         'subsample': [0.7]},
             scoring='roc_auc', verbose=2)
```

```
best_grid=grid_search.best_params_
best_grid
```

Saving...

```
{'colsample_bytree': 0.6,
 'gamma': 4.0,
 'min_child_weight': 1,
 'n_estimators': 1100,
 'scale_pos_weight': 1,
 'subsample': 0.7}
```

```
model_grid=xgb.XGBClassifier(objective='binary:logistic',
                            nthread=1,eval_metric='mlogloss',**best_grid)

fprs,tprs,score=[],[],[]


for (train,test), i in zip(cv.split(X,y),range(5)):
    model_grid.fit(X.iloc[train],y.iloc[train])
    _,_,auc_score_train=compute_roc_auc(model_grid,train)
    fpr,tpr,auc_score=compute_roc_auc(model_grid,test)
    score.append((auc_score_train,auc_score))
    fprs.append(fpr)
    tprs.append(tpr)


plot_roc_curve(fprs,tprs)
plt.show()
```

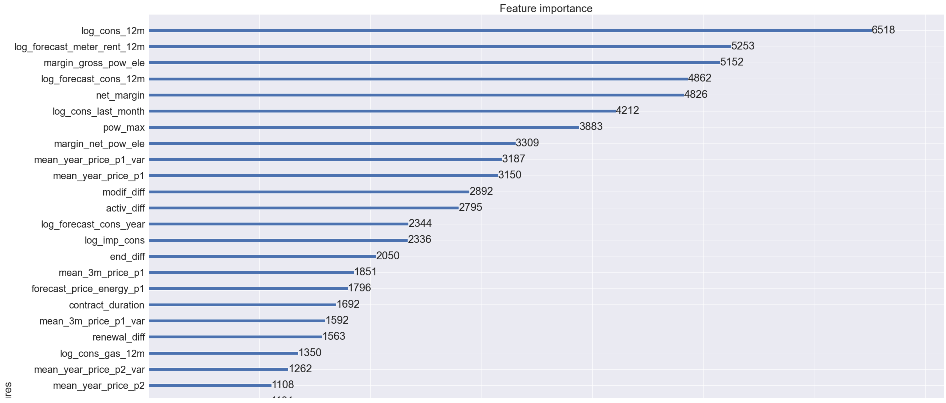


Understanding the model

Feature importance

```
fig,ax=plt.subplots()
fig.set_size_inches(40, 40)
xgb.plot_importance(model_grid,ax=ax);
```

Saving...



Partial dependence plot

mean_3m_price_p3_var 652

```
from sklearn.inspection import plot_partial_dependence
```

channel_sales_foos 403

```
model_grid_v2=xgb.XGBClassifier(objective='binary:logistic',
                                eval_metric='mlogloss',nthread=1,**best_grid)
model_grid_v2.fit(X_train.values,y_train.values)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.6, eval_metric='mlogloss',
              gamma=4.0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.005, max_delta_step=0,
              max_depth=14, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=1100, n_jobs=1, nthread=1,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=0.7, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

Saving...

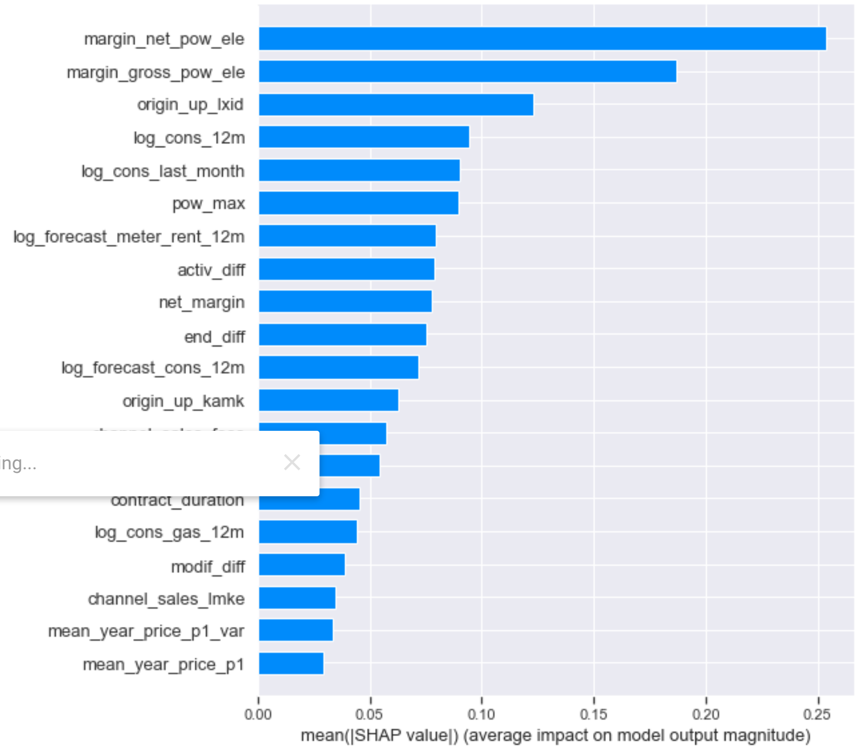
```
plot_partial_dependence(model_grid_v2,X_test.values,features=[16,49],feature_names=X_test.columns.tolist(),fig=fig)
```

```
<sklearn.inspection._plot.partial_dependence.PartialDependenceDisplay at 0x7f90a3c14ca0>
```

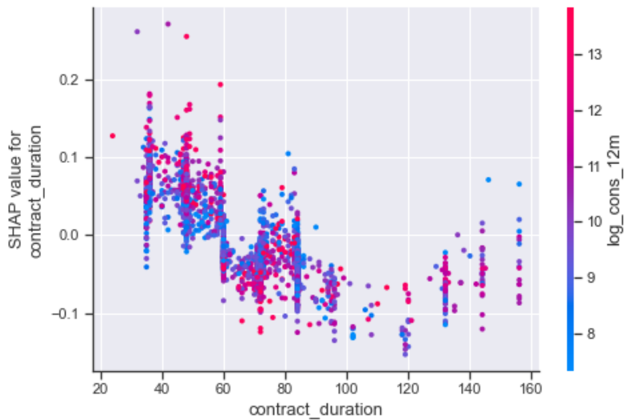
SHAP Feature importance

```
import shap
explainer=shap.TreeExplainer(model_grid)
shap_values=explainer.shap_values(X_test)
```

```
#Feature importance
shap.summary_plot(shap_values,X_test,plot_type='bar')
```

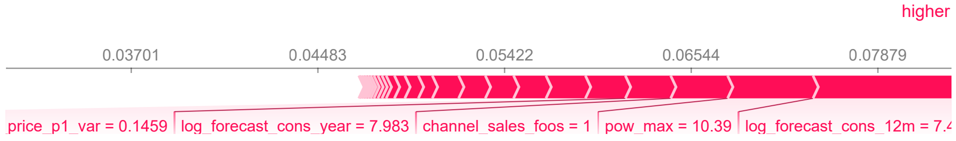



```
#Partial dependence plot
shap.dependence_plot('contract_duration',shap_values,X_test)
```



SHAP Single prediction

```
shap.initjs()
shap.force_plot(explainer.expected_value,shap_values[3171], X_test.iloc[3171,:],link='logit')
```


higher
0.03701
0.04483
0.05422
0.06544
0.07879
price_p1_var = 0.1459
log_forecast_cons_year = 7.983
channel_sales_foos = 1
pow_max = 10.39
log_forecast_cons_12m = 7.4

Saving...