Feature Engineering & Modelling

Uncovering signals within the data, predicting churn probability and evaluating model performance

---

1. Import packages
2. Load data
3. Feature engineering
4. Modelling
5. Evaluation
6. Conclusion

---

1. Import packages

In [1]:

```
import pandas as pd
import  numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold, GridSearchCV
from sklearn.ensemble import RandomForestClassifier

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
#from imblearn.pipeline import Pipeline

from sklearn.metrics import confusion_matrix, recall_score, precision_score, classification_report, make_scorer
from sklearn import metrics
```

In [2]:

```
from pprint import pprint
```

---

2. Load data

File failed to load: /extensions/MathZoom.js

```
clean_data_after_eda.csv')
df["date_activ"] = pd.to_datetime(df["date_activ"], format='%Y-%m-%d')
```

```
df["date_end"] = pd.to_datetime(df["date_end"], format='%Y-%m-%d')
df["date_modif_prod"] = pd.to_datetime(df["date_modif_prod"], format='%Y-%m-%d')
df["date_renewal"] = pd.to_datetime(df["date_renewal"], format='%Y-%m-%d')
```

In [4]:

```
df.head(3)
```

Out[4]:

	Unnamed: 0	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_prod	c
0	0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2016-06-15	2015-11-01	
1	1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2016-08-30	2009-08-21	
2	2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16	2016-04-16	2010-04-16	

3 rows × 45 columns

In [5]:

```
#drop the Unnamed column
df.drop(columns = 'Unnamed: 0', inplace=True)
```

---

3. Feature engineering

3.1 Difference between off-peak prices in December and preceding January

Below is the code created by your colleague to calculate the feature described above. Use this code to re-create this feature and then think about ways to build on this feature to create features with a higher predictive power.

In [6]:

```
price_df = pd.read_csv('./price_data.csv')
price_df["price_date"] = pd.to_datetime(price_df["price_date"], format='%Y-%m-%d')
price_df.head()
```

Out[6]:

	id	price_date	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_mid_peak_fix
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	0.0	44.266931	0.0	0.0

File failed to load: /extensions/MathZoom.js

	id	price_date	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_mid_peak_fix
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	0.0	44.266931	0.0	0.0
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	0.0	44.266931	0.0	0.0
3	038af19179925da21a25619c5a24b745	2015-04-01	0.149626	0.0	0.0	44.266931	0.0	0.0
4	038af19179925da21a25619c5a24b745	2015-05-01	0.149626	0.0	0.0	44.266931	0.0	0.0

In [7]:

```
# Group off-peak prices by companies and month
monthly_price_by_id = price_df.groupby(['id', 'price_date']).agg({'price_off_peak_var': 'mean', 'price_off_peak_fix': 'mean'}).reset_index()

# Get january and december prices
jan_prices = monthly_price_by_id.groupby('id').first().reset_index()
dec_prices = monthly_price_by_id.groupby('id').last().reset_index()

# Calculate the difference
diff = pd.merge(dec_prices.rename(columns={'price_off_peak_var': 'dec_1', 'price_off_peak_fix': 'dec_2'}), jan_prices.drop(columns='price_dat
diff['offpeak_diff_dec_jan_energy'] = diff['dec_1'] - diff['price_off_peak_var']
diff['offpeak_diff_dec_jan_power'] = diff['dec_2'] - diff['price_off_peak_fix']
diff = diff[['id', 'offpeak_diff_dec_jan_energy','offpeak_diff_dec_jan_power']]
diff.head()
```

Out[7]:

	id	offpeak_diff_dec_jan_energy	offpeak_diff_dec_jan_power
0	0002203ffbb812588b632b9e628cc38d	-0.006192	0.162916
1	0004351ebdd665e6ee664792efc4fd13	-0.004104	0.177779
2	0010bcc39e42b3c2131ed2ce55246e3c	0.050443	1.500000
3	0010ee3855fdea87602a5b7aba8e42de	-0.010018	0.162916
4	00114d74e963e47177db89bc70108537	-0.003994	-0.000001

In [8]:

```
columns = ['id', 'offpeak_diff_dec_jan_energy','offpeak_diff_dec_jan_power']
df = pd.merge(df,diff[columns], on='id')
```

Let us create some more features that can be created from price data

Like our collegue created a feature i.e., difference between the off peak prices in dec & jan of preceeding year, we can also create following features for both energy & power :

- [illegible] across time period
- max price difference across time period & month

File failed to load: /extensions/MathZoom.js

3.2 Avg price difference across time period

This feature will give some granuality to the yearly difference feature. This feature can play important role as it can reveal pattern on granual scale.

In [9]:

```
# Aggregating avg prices per period by companies
mean_prices = price_df.groupby(['id']).agg({
    'price_off_peak_var': 'mean',
    'price_peak_var': 'mean',
    'price_mid_peak_var': 'mean',
    'price_off_peak_fix': 'mean',
    'price_peak_fix': 'mean',
    'price_mid_peak_fix': 'mean'
}).reset_index()
```

In [10]:

```
# Calculate the avg difference between each periods
mean_prices['off_peak_peak_var_mean_diff'] = mean_prices['price_off_peak_var'] - mean_prices['price_peak_var']
mean_prices['peak_mid_peak_var_mean_diff'] = mean_prices['price_peak_var'] - mean_prices['price_mid_peak_var']
mean_prices['off_peak_mid_peak_var_mean_diff'] = mean_prices['price_off_peak_var'] - mean_prices['price_mid_peak_var']
mean_prices['off_peak_peak_fix_mean_diff'] = mean_prices['price_off_peak_fix'] - mean_prices['price_peak_fix']
mean_prices['peak_mid_peak_fix_mean_diff'] = mean_prices['price_peak_fix'] - mean_prices['price_mid_peak_fix']
mean_prices['off_peak_mid_peak_fix_mean_diff'] = mean_prices['price_off_peak_fix'] - mean_prices['price_mid_peak_fix']
```

In [11]:

```
columns = [
    'id',
    'off_peak_peak_var_mean_diff',
    'peak_mid_peak_var_mean_diff',
    'off_peak_mid_peak_var_mean_diff',
    'off_peak_peak_fix_mean_diff',
    'peak_mid_peak_fix_mean_diff',
    'off_peak_mid_peak_fix_mean_diff'
]
df = pd.merge(df, mean_prices[columns], on='id')
df.head()
```

Out[11]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_prod	date_renev
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2016-06-15	2015-11-01	2015-06-
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2016-08-30	2009-08-21	2015-08-
	254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-	2016-04-	2010-04-16	2015-04-

File failed to load: /extensions/MathZoom.js

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_prod	date_renev
						16	16		
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	2010-03-30	2016-03-30	2010-03-30	2015-03-
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526	2010-01-13	2016-03-07	2010-01-13	2015-03-

5 rows × 52 columns

3.3 Max price difference across time period & month

This feature can play important role as client would not like a sudden increase in short period of time & this can make them explore other options available in the market.

In [12]:

```
# Aggregate average prices per period by company & months
mean_prices_by_month = price_df.groupby(['id', 'price_date']).agg({
    'price_off_peak_var': 'mean',
    'price_peak_var': 'mean',
    'price_mid_peak_var': 'mean',
    'price_off_peak_fix': 'mean',
    'price_peak_fix': 'mean',
    'price_mid_peak_fix': 'mean'
}).reset_index()
```

In [13]:

```
# Calculate the mean difference between consecutive periods
mean_prices_by_month['off_peak_peak_var_mean_diff'] = mean_prices_by_month['price_off_peak_var'] - mean_prices_by_month['price_peak_var']
mean_prices_by_month['peak_mid_peak_var_mean_diff'] = mean_prices_by_month['price_peak_var'] - mean_prices_by_month['price_mid_peak_var']
mean_prices_by_month['off_peak_mid_peak_var_mean_diff'] = mean_prices_by_month['price_off_peak_var'] - mean_prices_by_month['price_mid_peak_v
mean_prices_by_month['off_peak_peak_fix_mean_diff'] = mean_prices_by_month['price_off_peak_fix'] - mean_prices_by_month['price_peak_fix']
mean_prices_by_month['peak_mid_peak_fix_mean_diff'] = mean_prices_by_month['price_peak_fix'] - mean_prices_by_month['price_mid_peak_fix']
mean_prices_by_month['off_peak_mid_peak_fix_mean_diff'] = mean_prices_by_month['price_off_peak_fix'] - mean_prices_by_month['price_mid_peak_f
```

In [14]:

```
# Calculate the maximum monthly difference across time periods
max_diff_across_periods_months = mean_prices_by_month.groupby(['id']).agg({
    'off_peak_peak_var_mean_diff': 'max',
    'peak_mid_peak_var_mean_diff': 'max',
    'off_peak_mid_peak_var_mean_diff': 'max',
    'off_peak_peak_fix_mean_diff': 'max',
    ix_mean_diff': 'max',
    'off_peak_mid_peak_fix_mean_diff': 'max'
```

File failed to load: /extensions/MathZoom.js

```
}).reset_index().rename(
    columns={
        'off_peak_peak_var_mean_diff': 'off_peak_peak_var_max_monthly_diff',
        'peak_mid_peak_var_mean_diff': 'peak_mid_peak_var_max_monthly_diff',
        'off_peak_mid_peak_var_mean_diff': 'off_peak_mid_peak_var_max_monthly_diff',
        'off_peak_peak_fix_mean_diff': 'off_peak_peak_fix_max_monthly_diff',
        'peak_mid_peak_fix_mean_diff': 'peak_mid_peak_fix_max_monthly_diff',
        'off_peak_mid_peak_fix_mean_diff': 'off_peak_mid_peak_fix_max_monthly_diff'
    }
)
```

In [15]:

```
columns = [
    'id',
    'off_peak_peak_var_max_monthly_diff',
    'peak_mid_peak_var_max_monthly_diff',
    'off_peak_mid_peak_var_max_monthly_diff',
    'off_peak_peak_fix_max_monthly_diff',
    'peak_mid_peak_fix_max_monthly_diff',
    'off_peak_mid_peak_fix_max_monthly_diff'
]

df = pd.merge(df, max_diff_across_periods_months[columns], on='id')
df.head()
```

Out[15]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_prod	date_renev
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2016-06-15	2015-11-01	2015-06-
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2016-08-30	2009-08-21	2015-08-
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16	2016-04-16	2010-04-16	2015-04-
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	2010-03-30	2016-03-30	2010-03-30	2015-03-
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526	2010-01-13	2016-03-07	2010-01-13	2015-03-

5 rows × 58 columns

File failed to load: /extensions/MathZoom.js

Lets move to other columns in clients data

3.4 Convert dates into months

As the predective models works better with numerical or categorical data & cant work with dates, so lets convert dates columns into number of months

We have 4 datetime features:

- date_activ = date of activation of the contract
- date_end = registered date of the end of the contract
- date_modif_prod = date of the last modification of the product
- date_renewal = date of the next contract renewal

From these features, we can create new features as follows:

- (refernce date = Jan 2016)
- active_period = # of months active
- expiry_period = # of months of the contract left
- modif_prod_period = # of months since last modification
- renewal_period = # of months since last renewal

How these features can be helpful to determin the churn:

- the clients who have higher active_period months will be more loyal as compared to the clients with lower no.of months
- the less number of months left in the contract can signal that client will be looking for better deals in market, if there are a lot of months left, so it could be possible that they can break the contract if not satified. if there are in middle of contarct period, some charges might be applied so the clients dont leave in between.
- if the modification has been done, that means they are custimizing the services as per clients needs that means client is less likely to churn.
- if the months since last renewal is less that means the clients is staying & if the number is more then there is decision time for client to look for better deals in market.

In [16]:

```
# Lets calculate these columns

df['Current date'] = '1-01-2016'
df['Current date'] = pd.to_datetime(df['Current date'])
df['active_p'] = ((df['Current date'] - df['date_activ'])/np.timedelta64(1, 'M')).astype(int)
df['expiry_p'] =  ((df['date_end'] - df['Current date'])/np.timedelta64(1, 'M')).astype(int)
df['modif_prod_p'] =  ((df['Current date'] - df['date_modif_prod'])/np.timedelta64(1, 'M')).astype(int)
df['renewal_p'] = ((df['Current date'] - df['date_renewal'])/np.timedelta64(1, 'M')).astype(int)
```

File failed to load: /extensions/MathZoom.js

In [17]:

```
df[['Current date','active_p','expiry_p','modif_prod_p','renewal_p']]
```

Out[17]:

	Current date	active_p	expiry_p	modif_prod_p	renewal_p
0	2016-01-01	30	5	2	6
1	2016-01-01	76	7	76	4
2	2016-01-01	68	3	68	8
3	2016-01-01	69	2	69	9
4	2016-01-01	71	2	71	9
...	...	...	...	...	...
14601	2016-01-01	43	4	7	19
14602	2016-01-01	40	7	40	4
14603	2016-01-01	46	1	46	10
14604	2016-01-01	40	7	40	4
14605	2016-01-01	72	11	72	0

14606 rows × 5 columns

In [18]:

```
# drop date columns as we dont require them for training our model
col_drop = ['date_activ', 'date_end', 'date_modif_prod', 'date_renewal','Current date']

df = df.drop(columns=col_drop)
df.head()
```

Out[18]:

	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_dis
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	0.00	0	
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	189.95	0	
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	47.96	0	
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	240.04	0	
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0	526	445.75	526	

File failed to load: /extensions/MathZoom.js

5 rows × 58 columns

In [19]:
```
#We do not require the 'id' column for the model so dropping that column

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

Out[19]:

	channel_sales	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_1
0	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	0.00	0	0.0	1
1	MISSING	4660	0	0	189.95	0	0.0	16
2	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	47.96	0	0.0	38
3	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	240.04	0	0.0	19
4	MISSING	4425	0	526	445.75	526	0.0	131

5 rows × 57 columns

Lets refer the Exploratory Data Analysis .ipynb file & perform further preprocessing of data .

3.4 Categorical Columns

In [20]:
```
pd_cols_object = df.select_dtypes(include=['object']).columns
pd_cols_object
```

Out[20]:
```
Index(['channel_sales', 'has_gas', 'origin_up'], dtype='object')
```

As per the requirement we are support to build a Random Forest Model. Random Forest Model can handle categorical variable very well.

1) 'has_gas' (Contact Type)

it have only "t" & "f" ,lets not cahnge anything here as it can be taken care by random forest model

In [21]:
```
df = pd.get_dummies(df, columns=['has_gas'],drop_first=True)
df.head()
```

File failed to load: /extensions/MathZoom.js

Out[21]:

	channel_sales	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_1
0	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	0.00	0	0.0	1
1	MISSING	4660	0	0	189.95	0	0.0	16
2	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	47.96	0	0.0	38
3	lmkebamcaaclubfxadlmueccxoimlema	1584	0	0	240.04	0	0.0	19
4	MISSING	4425	0	526	445.75	526	0.0	131

5 rows × 57 columns

2) 'channel_sales'

it has 8 sales channel out of which 3 sales channes have almost no data as compared to other 5. so lets drop the 3 sales channel that have almost no data

This will help to reduce the dimentaionality of feature space

In [22]:

```
df = pd.get_dummies(df, columns=['channel_sales'], prefix='ch')
df = df.drop(columns=['ch_sddiedcslfslkckwlfkdpoeeailfpeds', 'ch_epumfxlbckeskwekxbiuasklxalciiuu', 'ch_fixdbufsefwooaasfcxdxadsiekoceaa'])
df.head()
```

Out[22]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_12m	forecast_price_energy_off_peak
0	0	54946	0	0.00	0	0.0	1.78	0.114481
1	4660	0	0	189.95	0	0.0	16.27	0.145711
2	544	0	0	47.96	0	0.0	38.72	0.165794
3	1584	0	0	240.04	0	0.0	19.83	0.146694
4	4425	0	526	445.75	526	0.0	131.73	0.116900

5 rows × 61 columns

3) 'origin_up' (Electricity Campaign)

it has 6 campaigns out of which 3 have almost no data as compared to other 3. so lets drop the 3 campaigns that have almost no data

File failed to load: /extensions/MathZoom.js

he dimentaionality of feature space

In [23]:

```
df = pd.get_dummies(df, columns=['origin_up'], prefix='camp')
df = df.drop(columns=['camp_MISSING', 'camp_usapbepcfoloekilkwsdiboslwaxobdp', 'camp_ewxeelcelemmiwuafmddpobolfuxioce'])
df.head()
```

Out[23]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_12m	forecast_price_energy_off_peak
0	0	54946	0	0.00	0	0.0	1.78	0.114481
1	4660	0	0	189.95	0	0.0	16.27	0.145711
2	544	0	0	47.96	0	0.0	38.72	0.165794
3	1584	0	0	240.04	0	0.0	19.83	0.146694
4	4425	0	526	445.75	526	0.0	131.73	0.116900

5 rows × 63 columns

3.5 Numerical Columns

Addressing the skewness of numerical features is not a strict requirement when building a random forest model. Random forests are generally robust to outliers and skewed data, which is one of their strengths. However, addressing skewness in numerical features can sometimes improve the performance of the model.

Logarithmic transformation (log): This transformation is useful for reducing right-skewness and compressing large values. It is often used when dealing with data that has exponential growth or multiplicative relationships.

Lets list the skewed numerical columns from eda file

In [24]:

```
skewed = [
    'cons_12m',
    'cons_gas_12m',
    'cons_last_month',
    'imp_cons',
    'forecast_cons_12m',
    'forecast_cons_year',
    'forecast_discount_energy',
    'forecast_meter_rent_12m',
    'forecast_price_energy_peak',
    'margin_gross_pow_ele',
    'margin_net_pow_ele',
    'net_margin',
    'pow_max'
```

File failed to load: /extensions/MathZoom.js

```
df[skewed].describe()
```

Out[24]:

	cons_12m	cons_gas_12m	cons_last_month	imp_cons	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_12m	forecast_p
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	
mean	1.592203e+05	2.809238e+04	16090.269752	152.786896	1868.614880	1399.762906	0.966726	63.086871	
std	5.734653e+05	1.629731e+05	64364.196422	341.369366	2387.571531	3247.786255	5.108289	66.165783	
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	5.674750e+03	0.000000e+00	0.000000	0.000000	494.995000	0.000000	0.000000	16.180000	
50%	1.411550e+04	0.000000e+00	792.500000	37.395000	1112.875000	314.000000	0.000000	18.795000	
75%	4.076375e+04	0.000000e+00	3383.000000	193.980000	2401.790000	1745.750000	0.000000	131.030000	
max	6.207104e+06	4.154590e+06	771203.000000	15042.790000	82902.830000	175375.000000	30.000000	599.310000	

Note: We cannot apply log to a value of 0, so we will add a constant of 1 to all the values

In [25]:

```
# Apply log10 transformation
df["cons_12m"] = np.log10(df["cons_12m"] + 1)
df["cons_gas_12m"] = np.log10(df["cons_gas_12m"] + 1)
df["cons_last_month"] = np.log10(df["cons_last_month"] + 1)
df["imp_cons"] = np.log10(df["imp_cons"] + 1)
df["forecast_cons_12m"] = np.log10(df["forecast_cons_12m"] + 1)
df["forecast_cons_year"] = np.log10(df["forecast_cons_year"] + 1)
df["forecast_meter_rent_12m"] = np.log10(df["forecast_meter_rent_12m"] + 1)
df["imp_cons"] = np.log10(df["imp_cons"] + 1)
df["margin_gross_pow_ele"] = np.log10(df["margin_gross_pow_ele"] + 1)
df["margin_net_pow_ele"] = np.log10(df["margin_net_pow_ele"] + 1)
df["net_margin"] = np.log10(df["net_margin"] + 1)
df["pow_max"] = np.log10(df["pow_max"] + 1)
```

In [26]:

```
df[skewed].describe()
```

Out[26]:

	cons_12m	cons_gas_12m	cons_last_month	imp_cons	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_12m	forecast_p
count	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	14606.000000	
mean	4.223939	0.779244	2.264646	0.283253	2.962177	1.784610	0.966726	1.517203	
		1.717071	1.769305	0.250900	0.683592	1.584986	5.108289	0.571481	

File failed to load: /extensions/MathZoom.js

	cons_12m	cons_gas_12m	cons_last_month	imp_cons	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_12m	forecast_p
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	3.754023	0.000000	0.000000	0.000000	2.695477	0.000000	0.000000	1.235023	
50%	4.149727	0.000000	2.899547	0.412339	3.046836	2.498311	0.000000	1.296555	
75%	4.610285	0.000000	3.529430	0.517195	3.380716	3.242231	0.000000	2.120673	
max	6.792889	6.618528	5.887169	0.714108	4.918575	5.243970	30.000000	2.778376	

After the log transformation, we can observe the significant reduction in skewness

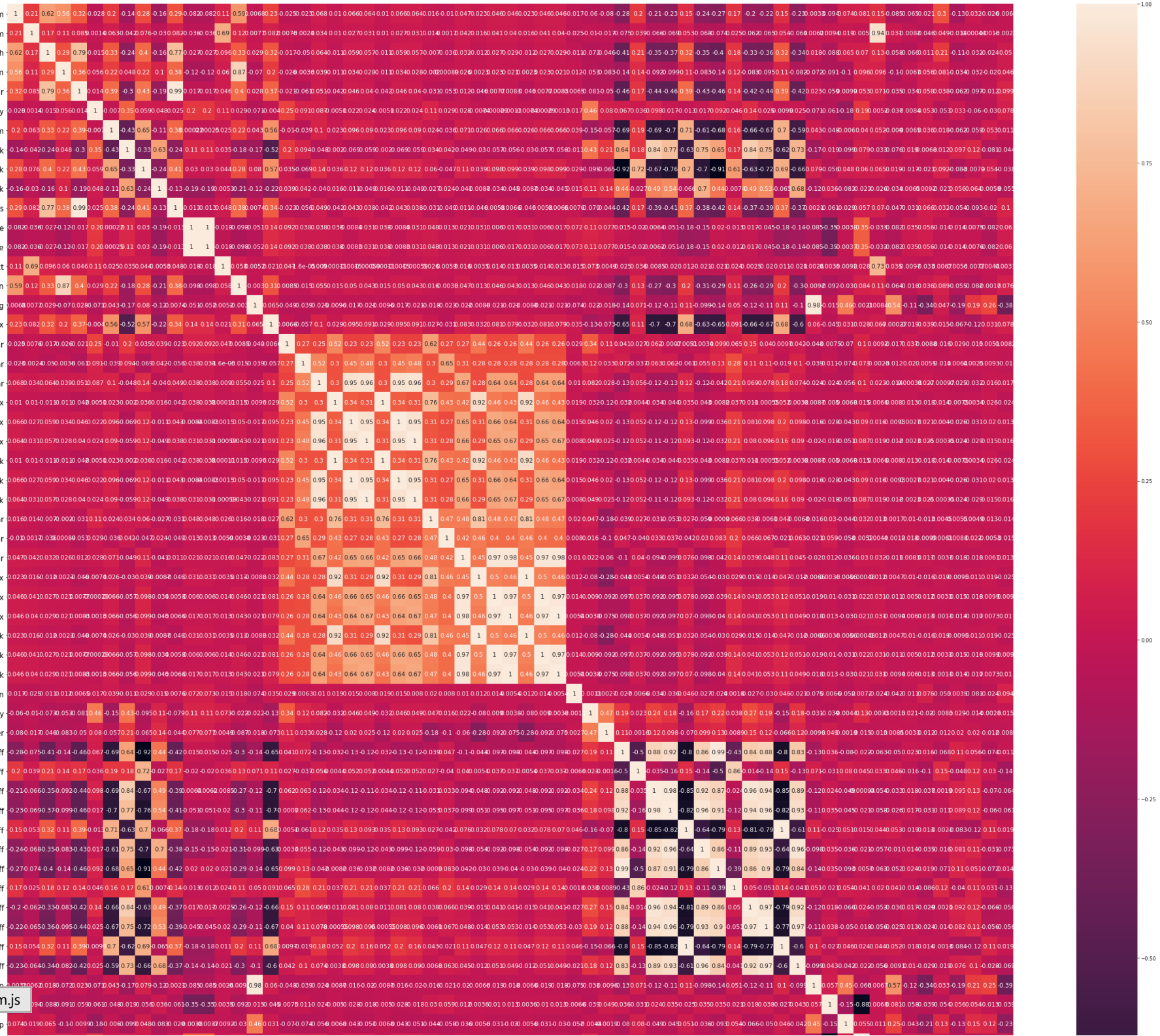
In [27]:

```
def plot_heatmap(correlation_matrix):
    # Plot correlation
    plt.figure(figsize=(45, 45))
    sns.heatmap(
    correlation_matrix,
    xticklabels=correlation_matrix.columns.values,
    yticklabels=correlation_matrix.columns.values,
    annot=True,
    annot_kws={'size': 12}
    )
    # Axis ticks size
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.show()
```

In [28]:

```
plot_heatmap(df.corr())
```

File failed to load: /extensions/MathZoom.js



File failed to load: /extensions/MathZoom.js

-- Create a dictionary to store correlated features

In [31]:

```
# Set the threshold for high correlation
correlation_threshold = 0.85

correlation_dict = {}

correlation_matrix =df.corr()

# Find pairs of highly correlated features and populate the dictionary
for i in range(len(correlation_matrix.columns)):
    correlated_features = set()
    for j in range(i + 1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) > correlation_threshold:
            correlated_features.add(correlation_matrix.columns[j])
    if correlated_features:
        correlation_dict[correlation_matrix.columns[i]] = correlated_features
```

In [33]:

```
# By using pprint, the dictionary will be displayed in a more organized and readable format,
# making it easier to see which features are highly correlated with each other.
pprint(correlation_dict, indent=4)
```

```
{   'cons_gas_12m': {'has_gas_t'},
    'expiry_p': {'renewal_p'},
    'forecast_cons_12m': {'net_margin'},
    'forecast_cons_year': {'imp_cons'},
    'forecast_discount_energy': {   'off_peak_peak_var_max_monthly_diff',
                                    'off_peak_peak_var_mean_diff'},
```

File failed to load: /extensions/MathZoom.js

```
'margin_gross_pow_ele': {'margin_net_pow_ele'},
'mean_6m_price_mid_peak_fix': {   'mean_6m_price_mid_peak',
                                  'mean_6m_price_peak'},
'mean_6m_price_mid_peak_var': {   'mean_6m_price_mid_peak',
                                  'mean_6m_price_mid_peak_fix',
                                  'mean_6m_price_peak',
                                  'mean_6m_price_peak_fix'},
'mean_6m_price_off_peak_fix': {'mean_6m_price_off_peak'},
'mean_6m_price_peak': {'mean_6m_price_mid_peak'},
'mean_6m_price_peak_fix': {   'mean_6m_price_mid_peak',
                              'mean_6m_price_mid_peak_fix',
                              'mean_6m_price_peak'},
'mean_year_price_mid_peak_fix': {   'mean_year_price_mid_peak',
                                    'mean_year_price_peak'},
'mean_year_price_mid_peak_var': {   'mean_year_price_mid_peak',
                                    'mean_year_price_mid_peak_fix',
                                    'mean_year_price_peak',
                                    'mean_year_price_peak_fix'},
'mean_year_price_off_peak': {   'mean_6m_price_off_peak',
                                'mean_6m_price_off_peak_fix'},
'mean_year_price_off_peak_fix': {   'mean_6m_price_off_peak',
                                    'mean_6m_price_off_peak_fix',
                                    'mean_year_price_off_peak'},
'mean_year_price_peak': {'mean_year_price_mid_peak'},
'mean_year_price_peak_fix': {   'mean_year_price_mid_peak',
                                'mean_year_price_mid_peak_fix',
                                'mean_year_price_peak'},
'num_years_antig': {'active_p'},
'off_peak_mid_peak_fix_mean_diff': {   'off_peak_mid_peak_fix_max_monthly_diff',
                                       'off_peak_mid_peak_var_max_monthly_diff',
                                       'off_peak_peak_fix_max_monthly_diff',
                                       'off_peak_peak_var_max_monthly_diff'},
'off_peak_mid_peak_var_max_monthly_diff': {   'off_peak_mid_peak_fix_max_monthly_diff',
                                              'off_peak_peak_fix_max_monthly_diff'},
'off_peak_mid_peak_var_mean_diff': {   'off_peak_mid_peak_fix_max_monthly_diff',
                                       'off_peak_mid_peak_fix_mean_diff',
                                       'off_peak_mid_peak_var_max_monthly_diff',
                                       'off_peak_peak_fix_max_monthly_diff',
                                       'off_peak_peak_fix_mean_diff',
                                       'off_peak_peak_var_max_monthly_diff'},
'off_peak_peak_fix_max_monthly_diff': {   'off_peak_mid_peak_fix_max_monthly_diff'},
'off_peak_peak_fix_mean_diff': {   'off_peak_mid_peak_fix_max_monthly_diff',
                                   'off_peak_mid_peak_fix_mean_diff',
                                   'off_peak_mid_peak_var_max_monthly_diff',
                                   'off_peak_peak_fix_max_monthly_diff',
                                   'off_peak_peak_var_max_monthly_diff'},
'off_peak_peak_var_max_monthly_diff': {   'off_peak_mid_peak_var_max_monthly_diff',
                                          'off_peak_peak_fix_max_monthly_diff'},
'off_peak_peak_var_mean_diff': {   'off_peak_mid_peak_fix_mean_diff',
                                   'off_peak_mid_peak_var_mean_diff',
                                   'off_peak_peak_fix_max_monthly_diff',
                                   'off_peak_peak_fix_mean_diff',
```

File failed to load: /extensions/MathZoom.js

```
                                        'off_peak_peak_var_max_monthly_diff'},
 'peak_mid_peak_fix_mean_diff': {'peak_mid_peak_fix_max_monthly_diff'},
 'peak_mid_peak_var_mean_diff': {'peak_mid_peak_var_max_monthly_diff'}}
```

Inferences:

- From the above dictionary, we can clearly identify couple of features to drop.
  ['cons_gas_12m','expiry_p','forecast_cons_12m','forecast_cons_year','margin_net_pow_ele','active_p']
- We can observe that there is high correlation btw the newly added price features so lets have a look in detail.Specifically -- the "mean_year" & "mean_6m" features are highly correlated -- the "mean_diff" & max_monthly_diff" featires are highly correlated

List of newly added price features:

Following price features were added as price sensitivity features in Task-2 (Exploratory Data Analysis)
["mean_year_price_off_peak","mean_year_price_peak","mean_year_price_mid_peak","mean_year_price_off_peak_var","mean_year_price_off_peak_fix","mean_year_price_p

Following price features were added in Task 3 (Feature Engineering)
['offpeak_diff_dec_jan_energy','offpeak_diff_dec_jan_power','off_peak_peak_var_mean_diff','peak_mid_peak_var_mean_diff','off_peak_mid_peak_var_mean_diff','off_peak

In [34]:
```
drop_features =['cons_gas_12m','expiry_p','forecast_cons_12m','forecast_cons_year','margin_net_pow_ele','active_p']
```

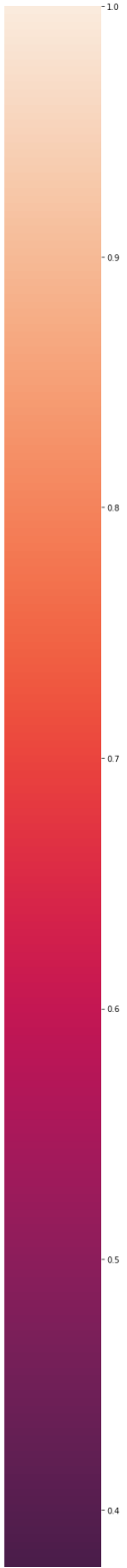
In [35]:
```
columns_t1 = ["mean_year_price_off_peak","mean_year_price_peak","mean_year_price_mid_peak","mean_year_price_off_peak_var","mean_year_price_of
columns_t2 = ['offpeak_diff_dec_jan_energy','offpeak_diff_dec_jan_power','off_peak_peak_var_mean_diff','peak_mid_peak_var_mean_diff','off_pea
```

In [36]:
```
df_t1 = df[columns_t1]
df_t2 = df[columns_t2]
```

In [38]:
```
plot_heatmap(df_t1.corr())
```

File failed to load: /extensions/MathZoom.js

	mean_year_price_off_peak	mean_year_price_peak	mean_year_price_mid_peak	mean_year_price_off_peak_var	mean_year_price_off_peak_fix	mean_year_price_peak_var	mean_year_price_peak_fix	mean_year_price_mid_peak_var	mean_year_price_mid_peak_fix	mean_6m_price_off_peak	mean_6m_price_peak	mean_6m_price_mid_peak	mean_6m_price_off_peak_var	mean_6m_price_off_peak_fix	[illegible]
mean_year_price_off_peak	1	0.34	0.31	0.52	1	0.3	0.34	0.3	0.31	0.92	0.46	0.43	0.76	0.92	0.43
mean_year_price_peak	0.34	1	0.95	0.23	0.34	0.45	1	0.95	0.95	0.31	0.66	0.64	0.31	0.31	0.27
mean_year_price_mid_peak	0.31	0.95	1	0.23	0.31	0.48	0.95	0.96	1	0.29	0.65	0.67	0.31	0.29	0.28
mean_year_price_off_peak_var	0.52	0.23	0.23	1	0.52	0.27	0.23	0.25	0.23	0.44	0.26	0.26	0.62	0.44	0.27
mean_year_price_off_peak_fix	1	0.34	0.31	0.52	1	0.3	0.34	0.3	0.31	0.92	0.46	0.43	0.76	0.92	0.43
mean_year_price_peak_var	0.3	0.45	0.48	0.27	0.3	1	0.45	0.52	0.48	0.28	0.28	0.28	0.3	0.28	0.65
mean_year_price_peak_fix	0.34	1	0.95	0.23	0.34	0.45	1	0.95	0.95	0.31	0.66	0.64	0.31	0.31	0.27
mean_year_price_mid_peak_var	0.3	0.95	0.96	0.25	0.3	0.52	0.95	1	0.96	0.28	0.64	0.64	0.3	0.28	0.29
mean_year_price_mid_peak_fix	0.31	0.95	1	0.23	0.31	0.48	0.95	0.96	1	0.29	0.65	0.67	0.31	0.29	0.28
mean_6m_price_off_peak	0.92	0.31	0.29	0.44	0.92	0.28	0.31	0.28	0.29	1	0.5	0.46	0.81	1	0.46
mean_6m_price_peak	0.46	0.66	0.65	0.26	0.46	0.28	0.66	0.64	0.65	0.5	1	0.97	0.48	0.5	0.4
mean_6m_price_mid_peak	0.43	0.64	0.67	0.26	0.43	0.28	0.64	0.64	0.67	0.46	0.97	1	0.47	0.46	0.4
mean_6m_price_off_peak_var	0.76	0.31	0.31	0.62	0.76	0.3	0.31	0.3	0.31	0.81	0.48	0.47	1	0.81	0.47
mean_6m_price_off_peak_fix	0.92	0.31	0.29	0.44	0.92	0.28	0.31	0.28	0.29	1	0.5	0.46	0.81	1	0.46
[illegible]	0.43	0.27	0.28	0.27	0.43	0.65	0.27	0.29	0.28	0.46	0.4	0.4	0.47	0.46	1



File failed to load: /extensions/MathZoom.js

In [39]:

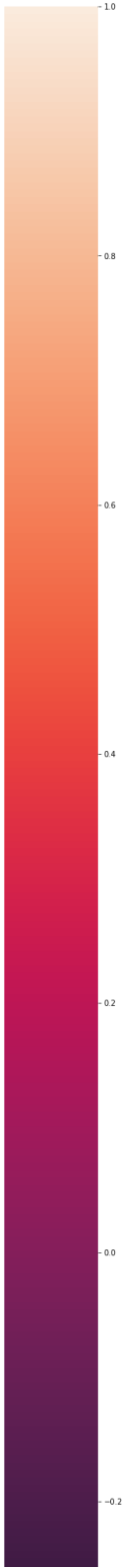
```
drop_features_t1 = ["mean_year_price_off_peak_fix",
"mean_year_price_peak_fix",
"mean_year_price_mid_peak",
"mean_year_price_mid_peak_var",
"mean_year_price_mid_peak_fix",
"mean_6m_price_off_peak",
"mean_6m_price_off_peak_fix",
"mean_6m_price_peak_fix",
"mean_6m_price_mid_peak",
"mean_6m_price_mid_peak_var",
"mean_6m_price_mid_peak_fix"]
```

In [40]:

```
plot_heatmap(df_t2.corr())
```

File failed to load: /extensions/MathZoom.js

	offpeak_diff_dec_jan_energy	offpeak_diff_dec_jan_power	off_peak_peak_var_mean_diff	peak_mid_peak_var_mean_diff	off_peak_mid_peak_var_mean_diff	off_peak_peak_fix_mean_diff	off_peak_mid_peak_fix_mean_diff	off_peak_peak_var_max_monthly_diff	peak_mid_peak_var_max_monthly_diff	off_peak_mid_peak_var_max_monthly_diff
offpeak_diff_dec_jan_energy	1	0.47	0.19	0.023	0.24	0.18	0.17	0.22	0.038	0.27
offpeak_diff_dec_jan_power	0.47	1	0.11	-0.0016	0.12	0.098	0.099	0.13	0.0089	0.15
off_peak_peak_var_mean_diff	0.19	0.11	1	-0.5	0.88	0.92	0.86	0.99	-0.43	0.84
peak_mid_peak_var_mean_diff	0.023	-0.0016	-0.5	1	-0.035	-0.16	-0.14	-0.5	0.86	-0.014
off_peak_mid_peak_var_mean_diff	0.24	0.12	0.88	-0.035	1	0.98	0.92	0.87	-0.024	0.96
off_peak_peak_fix_mean_diff	0.18	0.098	0.92	-0.16	0.98	1	0.96	0.91	-0.12	0.94
off_peak_mid_peak_fix_mean_diff	0.17	0.099	0.86	-0.14	0.92	0.96	1	0.86	-0.11	0.89
off_peak_peak_var_max_monthly_diff	0.22	0.13	0.99	-0.5	0.87	0.91	0.86	1	-0.39	0.86
peak_mid_peak_var_max_monthly_diff	0.038	0.0089	-0.43	0.86	-0.024	-0.12	-0.11	-0.39	1	0.05
off_peak_mid_peak_var_max_monthly_diff	0.27	0.15	0.84	-0.014	0.96	0.94	0.89	0.86	0.05	1



File failed to load: /extensions/MathZoom.js

In [41]:

```
keep_features_t2 = ['offpeak_diff_dec_jan_energy','offpeak_diff_dec_jan_power','peak_mid_peak_var_mean_diff']
drop_features_t2 = [item for item in columns_t2 if item not in keep_features_t2]
```

In [42]:

```
drop_features = drop_features+ drop_features_t1 + drop_features_t2
drop_features
```

Out[42]:

```
['cons_gas_12m',
 'expiry_p',
 'forecast_cons_12m',
 'forecast_cons_year',
 'margin_net_pow_ele',
 'active_p',
 'mean_year_price_off_peak_fix',
 'mean_year_price_peak_fix',
 'mean_year_price_mid_peak',
 'mean_year_price_mid_peak_var',
 'mean_year_price_mid_peak_fix',
 'mean_6m_price_off_peak',
 'mean_6m_price_off_peak_fix',
 'mean_6m_price_peak_fix',
 'mean_6m_price_mid_peak',
 'mean_6m_price_mid_peak_var',
 'mean_6m_price_mid_peak_fix',
 'off_peak_peak_var_mean_diff',
 'off_peak_mid_peak_var_mean_diff',
 'off_peak_peak_fix_mean_diff',
 'off_peak_mid_peak_fix_mean_diff',
 'off_peak_peak_var_max_monthly_diff',
```

File failed to load: /extensions/MathZoom.js

```
 'peak_mid_peak_var_max_monthly_diff',
 'off_peak_mid_peak_var_max_monthly_diff',
 'off_peak_peak_fix_max_monthly_diff',
 'off_peak_mid_peak_fix_max_monthly_diff']
```

In [43]:

```
df = df.drop(columns = drop_features, axis= 1)
df.columns
```

Out[43]:

```
Index(['cons_12m', 'cons_last_month', 'forecast_discount_energy',
       'forecast_meter_rent_12m', 'forecast_price_energy_off_peak',
       'forecast_price_energy_peak', 'forecast_price_pow_off_peak', 'imp_cons',
       'margin_gross_pow_ele', 'nb_prod_act', 'net_margin', 'num_years_antig',
       'pow_max', 'mean_year_price_off_peak_var', 'mean_year_price_peak_var',
       'mean_year_price_off_peak', 'mean_year_price_peak',
       'mean_6m_price_off_peak_var', 'mean_6m_price_peak_var',
       'mean_6m_price_peak', 'churn', 'offpeak_diff_dec_jan_energy',
       'offpeak_diff_dec_jan_power', 'peak_mid_peak_var_mean_diff',
       'peak_mid_peak_fix_mean_diff', 'peak_mid_peak_fix_max_monthly_diff',
       'modif_prod_p', 'renewal_p', 'has_gas_t', 'ch_MISSING',
       'ch_ewpakwlliwisiwduibdlfmalxowmwpci',
       'ch_foosdfpfkusacimwkcsosbicdxkicaua',
       'ch_lmkebamcaaclubfxadlmueccxoimlema',
       'ch_usilxuppasemubllopkaafesmlibmsdf',
       'camp_kamkkxfxxuwbdslkwifmmcsiusiuosws',
       'camp_ldkssxwpmemidmecebumciepifcamkci',
       'camp_lxidpiddsbxsbosboudacockeimpuepw'],
      dtype='object')
```

In [44]:

```
df.shape
```

Out[44]:

```
(14606, 37)
```

Feature Engineering is a recursive process by trail & error. Once we train our model & get the predictions, we will be abe to see the feature impoprtnce in case of Random Forest Model. Lets pause the feature engineering as of now & proceed.

Random Forest often outperforms other classification models due to the following reasons:

1) Reduced Overfitting: Random feature selection and data sampling during tree building reduce overfitting, making it less sensitive to noisy data.

2) Ensemble of Diverse Trees: It combines multiple decision trees, each trained on different subsets of data and features, leading to a more accurate and robust model.

File failed to load: /extensions/MathZoom.js

3) Robust to Outliers: Outliers have less impact as decisions are made based on majority votes, making it resilient to noisy data points.

4) Captures Non-linearity: Random Forest can model complex non-linear relationships in data, effectively handling intricate patterns.

5) Handles High-Dimensional Data: It performs well with datasets containing a large number of features without requiring extensive feature engineering.

6) Feature Importance: It naturally computes feature importance, aiding in feature selection and interpretation of results.

7) Less Prone to Bias: Unlike some models, Random Forest is less susceptible to bias as individual trees can balance each other's biases.

8) Limited Hyperparameter Tuning: Requires fewer hyperparameters to tune compared to other complex models, simplifying the optimization process.

9) Generalization: Random Forest performs well on both unseen data and training data, striking a balance between bias and variance.

10) Handling Missing Values: It can handle missing values without explicit imputation, streamlining the data preprocessing phase.

These factors collectively contribute to Random Forest's ability to provide accurate and reliable classifications across a wide range of datasets and scenarios.

4. Modelling

In [45]:

```
# Split data into features and target
X = df.drop('churn', axis=1)
y = df['churn']

# Split data into training(70%) and testing(30%) sets using stratified sampling (as the class imbalnced exists in churn data)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

In [46]:

```
# RandomUnderSampler to address the class imbalance problem
under_sampler = RandomUnderSampler(sampling_strategy= 0.75, random_state=42)  # Adjust the sampling_strategy as needed

# Apply to the training data
X_train_resampled, y_train_resampled = under_sampler.fit_resample(X_train, y_train)
```

What parameters to choose for training the Random Forest Model

- Setting "n_jobs" to -1 in the RandomForestClassifier hyperparameters indicates that you want to utilize all available CPU cores for parallel processing when building the individual decision trees in the forest. This can significantly speed up the training process, especially for larger datasets and models with a higher number of estimators (trees).

- Setting "random_state" to a specific value, such as random_state = 42, ensures that your results are reproducible, which is important for consistent testing, debugging, and comparing different experiments.

File failed to load: /extensions/MathZoom.js

- Setting "class_weight" to 'balanced' automatically adjusts the weights inversely proportional to class frequencies. This is important when dealing with imbalanced datasets to ensure that the model pays appropriate attention to minority classes.

What parameters to choose for hyperparameter tuning

- n_estimators: This hyperparameter controls the number of decision trees in the forest. Increasing the number of trees can lead to a more robust and accurate model, but it also increases computation time. It's important to find a balance between model performance and computational efficiency.

- max_depth: This hyperparameter restricts the maximum depth of each decision tree in the forest. A shallow tree (low max_depth) might be underfitting and not capture complex patterns, while a deep tree (high max_depth) can lead to overfitting. It's essential to tune this parameter to prevent overfitting and find the optimal level of tree depth.

- min_samples_split: This hyperparameter defines the minimum number of samples required to split an internal node. A higher value reduces the tree's tendency to overfit by preventing further splitting when there are only a few samples left in a node. Adjusting this parameter can help control the balance between model complexity and generalization.

How to address the class imbalance problem while training & cross validation

"StratifiedKFold" is a cross-validation technique used to address the potential issue of class imbalance in the training and validation splits during the evaluation of machine learning models. It's particularly useful when you have imbalanced datasets, where one class has significantly fewer samples than the other(s).

The traditional KFold cross-validation technique splits the dataset into k folds, where each fold is used as a validation set once and the rest as the training set. However, if your data is imbalanced, there's a risk that some folds may not contain representative samples of the minority class, leading to poor generalization.

StratifiedKFold addresses this by ensuring that each fold maintains the same class distribution as the original dataset. In other words, each fold will have a similar proportion of each class as the full dataset. This helps in improving the reliability of the cross-validation process, especially for imbalanced datasets.

Choice of Evaluation Metrics

In churn prediction, recall is crucial as it minimizes missed opportunities by focusing on identifying actual churn cases, which helps businesses retain customers and mitigate revenue loss due to false negatives.

Precision is vital in churn prediction as it ensures accurate identification of customers likely to leave, minimizing costly false positive predictions, optimizing resource allocation, and preserving customer trust by avoiding unnecessary retention efforts. It contributes to efficient decision-making and resource management while minimizing customer dissatisfaction.

To obtain the best results for two evaluation metrics, such as "Recall" and "Precision," you can use the refit parameter of the GridSearchCV to specify the primary

File failed to load: /extensions/MathZoom.js

ize.

In [47]:

```python
# Define a parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators' : [200, 300, 400, 500],
    'max_depth': [10, 12, 15, 20, 30, 35],
    'min_samples_split': [5, 8, 10]
}

# Create StratifiedKFold
stratified_cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Create a Random Forest model
random_forest_model = RandomForestClassifier(
    random_state=42,
    class_weight='balanced',
    n_jobs=-1
)

# Define custom scoring that includes both precision and recall
scoring = {
    'recall': make_scorer(recall_score),
    'precision': make_scorer(precision_score)
}

# Use GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(random_forest_model, param_grid, cv=stratified_cv, scoring=scoring, refit='recall')
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print("Best Hyperparameters:")
print(best_params)

best_model = grid_search.best_estimator_
print("Best Model:")
print(best_model)

# Fit the model on the training data with the best parameters
best_random_forest_model = best_model

# Fit the model on the resampled training data
best_random_forest_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test data
predictions = best_random_forest_model.predict(X_test)
```

File failed to load: /extensions/MathZoom.js

```
{'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 200}
```

```
Best Model:
RandomForestClassifier(class_weight='balanced', max_depth=10,
                       min_samples_split=10, n_estimators=200, n_jobs=-1,
                       random_state=42)
```

In [48]:

```python
# Generate the classification report
class_report = classification_report(y_test, predictions)

print("Classification Report:")
print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.69      0.80      3956
           1       0.18      0.61      0.27       426

    accuracy                           0.68      4382
   macro avg       0.56      0.65      0.53      4382
weighted avg       0.87      0.68      0.75      4382
```

In [49]:

```python
tn, fp, fn, tp = metrics.confusion_matrix(y_test, predictions).ravel()
```

In [50]:

```python
print(f"True positives: {tp}")
print(f"False positives: {fp}")
print(f"True negatives: {tn}")
print(f"False negatives: {fn}\n")
```

```
True positives: 258
False positives: 1216
True negatives: 2740
False negatives: 168
```

In [51]:

```python
y_test.value_counts()
```

Out[51]:

```
0    3956
1     426
Name: churn, dtype: int64
```

In [52]:

```python
# Get feature importance scores
feature_importance_scores = best_model.feature_importances_

# Get the features
feature_names = X.columns
```

File failed to load: /extensions/MathZoom.js

```python
# Set the display option to show all rows
pd.set_option('display.max_rows', None)

# Create a DataFrame to display feature importance
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance_scores})

# Sort the DataFrame by importance scores in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

print("Feature Importance:")
print(feature_importance_df)
```

```
Feature Importance:
                                 Feature  Importance
8                    margin_gross_pow_ele    0.107509
0                                cons_12m    0.070812
3                 forecast_meter_rent_12m    0.066873
10                              net_margin    0.057780
1                        cons_last_month    0.052209
25                          modif_prod_p    0.048132
15               mean_year_price_off_peak    0.043525
20            offpeak_diff_dec_jan_energy    0.042192
13           mean_year_price_off_peak_var    0.042119
12                               pow_max    0.038043
4          forecast_price_energy_off_peak    0.037418
7                               imp_cons    0.036178
26                             renewal_p    0.031114
22            peak_mid_peak_var_mean_diff    0.030559
11                        num_years_antig    0.030001
35  camp_lxidpiddsbxsbosboudacockeimpuepw    0.028058
17             mean_6m_price_off_peak_var    0.027879
16                  mean_year_price_peak    0.026913
5              forecast_price_energy_peak    0.023830
14               mean_year_price_peak_var    0.022320
33  camp_kamkkxfxxuwbdslkwifmmcsiusiuosws    0.016700
18                 mean_6m_price_peak_var    0.015881
21             offpeak_diff_dec_jan_power    0.015704
19                     mean_6m_price_peak    0.015461
23            peak_mid_peak_fix_mean_diff    0.012641
6             forecast_price_pow_off_peak    0.008802
30    ch_foosdfpfkusacimwkcsosbicdxkicaua    0.007993
9                              nb_prod_act    0.007736
31    ch_lmkebamcaaclubfxadlmueccxoimlema    0.007641
28                             ch_MISSING    0.006892
27                              has_gas_t    0.005826
34  camp_ldkssxwpmemidmecebumciepifcamkci    0.003936
29    ch_ewpakwlliwisiwduibdlfmalxowmwpci    0.003145
24     peak_mid_peak_fix_max_monthly_diff    0.002902
2                forecast_discount_energy    0.002675
32    ch_usilxuppasemubllopkaafesmlibmsdf    0.002600
```

File failed to load: /extensions/MathZoom.js

-- Net margin, Margin on Power, consumption over 12 months & last month and forecasted bill of meter rental for the next 12 months are the top 5 driver for churn in this model

-- Time seems to be an influential factor, especially the number of months they have done the last modification of product, number of months since last renewal & the loyality i.e.,antiquity of clients. These 3 time based features are in top 50% of influencing features.

-- The feature that our colleague recommended is in the top half in terms of how influential it is for energy but not for power.

--One of the Campaign feature is in top 50% of influencing feature but sales channel is clearly not the main driver for customer

Conclusion

Precision-Recall Balance: The model achieved a total of 249 true positives, indicating its ability to correctly identify actual churn cases. However, the number of false positives (1124) raises concerns about the precision of the positive predictions.

Negative Predictions: The model performed well in identifying true negatives (2832) which indicates its proficiency in recognizing cases where customers are likely to retain their services.

Missed Churn Cases: The model also had 177 false negatives, implying instances where actual churn cases were not accurately predicted. This suggests a need to improve the model's recall to capture more churn cases.

Business Impact: The high number of false positives might lead to unnecessary retention efforts, potentially straining resources and customer relationships. Addressing this is crucial to ensure efficient resource allocation.

Balancing Act: Balancing precision and recall is pivotal in this context. Adjusting the classification threshold can help strike a balance between minimizing false positives and capturing more true positives.

Refinement Opportunities: The model's performance can be enhanced through techniques such as adjusting class weights, feature engineering, and parameter tuning to improve precision and recall simultaneously.

Domain Expertise: Collaborating with domain experts to better understand exiting factors as well as other features influencing churn and refining features could lead to a more accurate model.

In [ ]:

```

```

File failed to load: /extensions/MathZoom.js