Feature engineering and Modelling

December 16, 2023

1 Churn Prediction Model with RandomForest classifier

[5]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

[6]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from datetime import datetime
import matplotlib.pyplot as plt

# Shows plots in jupyter notebook
%matplotlib inline

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

1.1 2. Load data

[11]:
```
df = pd.read_csv('./churn_data_modeling.csv')
df['date_activ'] = pd.to_datetime(df['date_activ'] , format='%m/%d/%Y')
df['date_end'] = pd.to_datetime(df['date_end'] ,  format='%m/%d/%Y')
df['date_modif_prod'] = pd.to_datetime(df['date_modif_prod'] ,  format='%m/%d/
↪%Y')
df['date_renewal'] = pd.to_datetime(df['date_renewal'] ,  format='%m/%d/%Y')
```

[12]:
```
df.head(3)
```

[12]:
```
   Unnamed: 0                                id  \
0           0  24011ae4ebbe3035111d65fa7c15bc57
1           1  d29c2c54acc38ff3c0614d0a653813dd
2           2  764c75f661154dac3a6c254cd082ea7d

                      channel_sales  cons_12m  cons_gas_12m  cons_last_month  \
0  foosdfpfkusacimwkcsosbicdxkicaua         0         54946                0
1                           MISSING      4660             0                0
```

```
2  foosdfpfkusacimwkcsosbicdxkicaua       544             0                 0

  date_activ   date_end date_modif_prod date_renewal  …  \
0 2013-06-15 2016-06-15      2015-11-01   2015-06-23  …
1 2009-08-21 2016-08-30      2009-08-21   2015-08-31  …
2 2010-04-16 2016-04-16      2010-04-16   2015-04-17  …

   mean_year_price_off_peak_var_y  mean_year_price_peak_var_y  \
0                        0.123500                    0.102447
1                        0.149968                    0.012212
2                        0.171116                    0.088581

   mean_year_price_mid_peak_var_y  mean_year_price_off_peak_fix_y  \
0                        0.072522                       40.635792
1                        0.000000                       44.266930
2                        0.000000                       44.393916

   mean_year_price_peak_fix_y  mean_year_price_mid_peak_fix_y  \
0                   24.381472                       16.254314
1                    0.000000                        0.000000
2                    0.000000                        0.000000

   mean_year_price_off_peak_y mean_year_price_peak_y  \
0                   40.759293              24.483919
1                   44.416898               0.012212
2                   44.565032               0.088581

   mean_year_price_med_peak_y  churn
0                   16.326836      1
1                    0.000000      0
2                    0.000000      0

[3 rows x 54 columns]
```

1.2 (BONUS) Further feature engineering

This section covers extra feature engineering that you may have thought of, as well as different ways you can transform your data to account for some of its statistical properties that we saw before, such as skewness.

1.2.1 Tenure

How long a company has been a client of PowerCo.

[13]:
```
df['tenure'] = ((df['date_end'] - df['date_activ'])/ np.timedelta64(1, 'Y')).
↪astype(int)
```

```
[14]: df.groupby(['tenure']).agg({'churn': 'mean'}).sort_values(by='churn',␣
      ↪ascending=False)
```

```
[14]:            churn
      tenure
      3       0.143713
      2       0.133080
      4       0.125756
      13      0.095238
      5       0.085425
      12      0.083333
      6       0.080713
      7       0.073394
      11      0.063584
      8       0.048000
      9       0.024096
      10      0.020000
```

It is evident that companies with a client tenure of 4 months or less exhibit a significantly higher likelihood of churning compared to those with a longer tenure. Notably, the transition from 4 to 5 months sees a notable 4% increase in the probability of churn, representing a substantial leap compared to other intervals of ordered tenure values. This suggests that surpassing the 4-month mark may serve as a crucial milestone in retaining customers for the long term.

This observed pattern is worth incorporating into modeling efforts, emphasizing the substantial influence of client tenure on the likelihood of churn.

When transforming dates into months:

- `months_activ`: Number of months active until the reference date (Jan 2016)
- `months_to_end`: Number of months left in the contract until the reference date (Jan 2016)
- `months_modif_prod`: Number of months since the last product modification until the reference date (Jan 2016)
- `months_renewal`: Number of months since the last renewal until the reference date (Jan 2016)

```
[15]: def convert_months(reference_date, df, column):
          """
          Input a column with timedeltas and return months
          """
          time_delta = reference_date - df[column]
          months = (time_delta / np.timedelta64(1, 'M')).astype(int)
          return months
```

```
[16]: # Create reference date
      reference_date = datetime(2016, 1, 1)

      # Create columns
      df['months_activ'] = convert_months(reference_date, df, 'date_activ')
```

```python
df['months_to_end'] = -convert_months(reference_date, df, 'date_end')
df['months_modif_prod'] = convert_months(reference_date, df, 'date_modif_prod')
df['months_renewal'] = convert_months(reference_date, df, 'date_renewal')
```

Representing dates as datetime objects proves impractical for a predictive model; thus, we utilized these datetime values to derive alternative features with potential predictive significance.

Taking an intuitive approach, one might speculate that clients with a lengthier tenure at PowerCo demonstrate greater brand loyalty, making them more inclined to stay. Conversely, newer clients may exhibit higher volatility. Hence, the introduction of the `months_activ` feature.

Considering the client's perspective, approaching the contract's end with PowerCo could lead to various considerations. Clients may seek better deals, explore contract renewal options, or, if recently joined, exercise the option to leave if dissatisfied. Additionally, clients in the middle of their contract may face charges for early termination, acting as a deterrent. Therefore, `months_to_end` emerges as an intriguing feature, potentially unveiling patterns and behaviors related to churn timing.

I posit that recent updates to a client's contract signify satisfaction or at least engagement with PowerCo's customer service. This engagement is indicative of a positive relationship, making `months_modif_prod` a valuable feature for gauging client involvement.

Lastly, the duration since a client last renewed a contract presents a compelling feature. It not only reflects engagement but also signifies a level of commitment. Contract renewals demonstrate a client's dedication, making `months_renewal` a meaningful feature to include in the predictive model.

[17]:
```python
# We no longer need the datetime columns that we used for feature engineering,␣
↪so we can drop them
remove = [
    'date_activ',
    'date_end',
    'date_modif_prod',
    'date_renewal'
]

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

[17]:
```
   Unnamed: 0                                id  \
0           0  24011ae4ebbe3035111d65fa7c15bc57
1           1  d29c2c54acc38ff3c0614d0a653813dd
2           2  764c75f661154dac3a6c254cd082ea7d
3           3  bba03439a292a1e166f80264c16191cb
4           4  149d57cf92fc41cf94415803a877cb4b

                     channel_sales  cons_12m  cons_gas_12m  cons_last_month  \
0  foosdfpfkusacimwkcsosbicdxkicaua         0         54946                0
1                          MISSING      4660             0                0
```

```
2  foosdfpfkusacimwkcsosbicdxkicaua       544              0                0
3  lmkebamcaaclubfxadlmueccxoimlema      1584              0                0
4                           MISSING      4425              0              526

   forecast_cons_12m  forecast_cons_year  forecast_discount_energy  \
0               0.00                   0                         0
1             189.95                   0                         0
2              47.96                   0                         0
3             240.04                   0                         0
4             445.75                 526                         0

   forecast_meter_rent_12m  …  mean_year_price_mid_peak_fix_y  \
0                     1.78  …                       16.254314
1                    16.27  …                        0.000000
2                    38.72  …                        0.000000
3                    19.83  …                        0.000000
4                   131.73  …                       16.282245

   mean_year_price_off_peak_y  mean_year_price_peak_y  \
0                   40.759293               24.483919
1                   44.416898                0.012212
2                   44.565032                0.088581
3                   44.571098                0.000000
4                   40.831065               24.527750

  mean_year_price_med_peak_y  churn  tenure  months_activ  months_to_end  \
0                  16.326836      1       3            30              5
1                   0.000000      0       7            76              7
2                   0.000000      0       6            68              3
3                   0.000000      0       6            69              2
4                  16.355497      0       6            71              2

   months_modif_prod  months_renewal
0                  2               6
1                 76               4
2                 68               8
3                 69               9
4                 71               9

[5 rows x 55 columns]
```

1.2.2 Boolean data (Convertion of Categorical data)

has_gas We simply want to transform this column from being categorical to being a binary flag

[18]:
```
df['has_gas'] = df['has_gas'].replace(['t', 'f'], [1, 0])
df.groupby(['has_gas']).agg({'churn': 'mean'})
```

[18]:
```
             churn
has_gas
0         0.100544
1         0.081856
```

1.2.3 Transforming categorical data

A predictive model cannot accept categorical or `string` values, hence as a data scientist you need to encode categorical features into numerical representations in the most compact and discriminative way possible.

The simplest method is to map each category to an integer (label encoding), however this is not always appropriate beecause it then introduces the concept of an order into a feature which may not inherently be present `0 < 1 < 2 < 3 ...`

Another way to encode categorical features is to use `dummy variables` AKA `one hot encoding`. This create a new feature for every unique value of a categorical column, and fills this column with either a 1 or a 0 to indicate that this company does or does not belong to this category.

channel_sales

[19]:
```python
# Transform into categorical type
df['channel_sales'] = df['channel_sales'].astype('category')

# Let's see how many categories are within this column
df['channel_sales'].value_counts()
```

[19]:
```
foosdfpfkusacimwkcsosbicdxkicaua    6754
MISSING                             3725
lmkebamcaaclubfxadlmueccxoimlema    1843
usilxuppasemubllopkaafesmlibmsdf    1375
ewpakwlliwisiwduibdlfmalxowmwpci     893
sddiedcslfslkckwlfkdpoeeailfpeds      11
epumfxlbckeskwekxbiuasklxalciiuu       3
fixdbufsefwooaasfcxdxadsiekoceaa       2
Name: channel_sales, dtype: int64
```

We have 8 categories, so we will create 8 dummy variables from this column. However, as you can see the last 3 categories in the output above, show that they only have 11, 3 and 2 occurrences respectively. Considering that our dataset has about 14000 rows, this means that these dummy variables will be almost entirely 0 and so will not add much predictive power to the model at all (since they're almost entirely a constant value and provide very little).

For this reason, we will drop these 3 dummy variables.

[20]:
```python
df = pd.get_dummies(df, columns=['channel_sales'], prefix='channel')
df = df.drop(columns=['channel_sddiedcslfslkckwlfkdpoeeailfpeds',␣
 ↪'channel_epumfxlbckeskwekxbiuasklxalciiuu',␣
 ↪'channel_fixdbufsefwooaasfcxdxadsiekoceaa'])
df.head()
```

[20]:

```
   Unnamed: 0                                id  cons_12m  cons_gas_12m  \
0           0  24011ae4ebbe3035111d65fa7c15bc57         0         54946
1           1  d29c2c54acc38ff3c0614d0a653813dd      4660             0
2           2  764c75f661154dac3a6c254cd082ea7d       544             0
3           3  bba03439a292a1e166f80264c16191cb      1584             0
4           4  149d57cf92fc41cf94415803a877cb4b      4425             0

   cons_last_month  forecast_cons_12m  forecast_cons_year  \
0                0               0.00                   0
1                0             189.95                   0
2                0              47.96                   0
3                0             240.04                   0
4              526             445.75                 526

   forecast_discount_energy  forecast_meter_rent_12m  \
0                         0                     1.78
1                         0                    16.27
2                         0                    38.72
3                         0                    19.83
4                         0                   131.73

   forecast_price_energy_off_peak  …  tenure  months_activ  months_to_end  \
0                        0.114481  …       3            30              5
1                        0.145711  …       7            76              7
2                        0.165794  …       6            68              3
3                        0.146694  …       6            69              2
4                        0.116900  …       6            71              2

   months_modif_prod  months_renewal  channel_MISSING  \
0                  2               6                0
1                 76               4                1
2                 68               8                0
3                 69               9                0
4                 71               9                1

   channel_ewpakwlliwisiwduibdlfmalxowmwpci  \
0                                         0
1                                         0
2                                         0
3                                         0
4                                         0

   channel_foosdfpfkusacimwkcsosbicdxkicaua  \
0                                         1
1                                         0
2                                         1
3                                         0
```

```
4                                         0

   channel_lmkebamcaaclubfxadlmueccxoimlema  \
0                                         0
1                                         0
2                                         0
3                                         1
4                                         0

  channel_usilxuppasemubllopkaafesmlibmsdf
0                                        0
1                                        0
2                                        0
3                                        0
4                                        0

[5 rows x 59 columns]
```

origin_up

[21]:
```
# Transform into categorical type
df['origin_up'] = df['origin_up'].astype('category')

# Let's see how many categories are within this column
df['origin_up'].value_counts()
```

[21]:
```
lxidpiddsbxsbosboudacockeimpuepw    7097
kamkkxfxxuwbdslkwifmmcsiusiuosws    4294
ldkssxwpmemidmecebumciepifcamkci    3148
MISSING                               64
usapbepcfoloekilkwsdiboslwaxobdp       2
ewxeelcelemmiwuafmddpobolfuxioce       1
Name: origin_up, dtype: int64
```

Similar to `channel_sales` the last 3 categories in the output above show very low frequency, so we will remove these from the features after creating dummy variables.

[22]:
```
df = pd.get_dummies(df, columns=['origin_up'], prefix='origin_up')
df = df.drop(columns=['origin_up_MISSING',␣
 ↪'origin_up_usapbepcfoloekilkwsdiboslwaxobdp',␣
 ↪'origin_up_ewxeelcelemmiwuafmddpobolfuxioce'])
df.head()
```

[22]:
```
   Unnamed: 0                                id  cons_12m  cons_gas_12m  \
0           0  24011ae4ebbe3035111d65fa7c15bc57         0         54946
1           1  d29c2c54acc38ff3c0614d0a653813dd      4660             0
2           2  764c75f661154dac3a6c254cd082ea7d       544             0
3           3  bba03439a292a1e166f80264c16191cb      1584             0
```

```
4          4  149d57cf92fc41cf94415803a877cb4b      4425              0

   cons_last_month  forecast_cons_12m  forecast_cons_year  \
0                0               0.00                   0
1                0             189.95                   0
2                0              47.96                   0
3                0             240.04                   0
4              526             445.75                 526

   forecast_discount_energy  forecast_meter_rent_12m  \
0                         0                     1.78
1                         0                    16.27
2                         0                    38.72
3                         0                    19.83
4                         0                   131.73

   forecast_price_energy_off_peak  …  months_modif_prod  months_renewal  \
0                        0.114481  …                  2               6
1                        0.145711  …                 76               4
2                        0.165794  …                 68               8
3                        0.146694  …                 69               9
4                        0.116900  …                 71               9

   channel_MISSING  channel_ewpakwlliwisiwduibdlfmalxowmwpci  \
0                0                                         0
1                1                                         0
2                0                                         0
3                0                                         0
4                1                                         0

   channel_foosdfpfkusacimwkcsosbicdxkicaua  \
0                                         1
1                                         0
2                                         1
3                                         0
4                                         0

   channel_lmkebamcaaclubfxadlmueccxoimlema  \
0                                         0
1                                         0
2                                         0
3                                         1
4                                         0

   channel_usilxuppasemubllopkaafesmlibmsdf  \
0                                         0
1                                         0
```

```
2                                          0
3                                          0
4                                          0

   origin_up_kamkkxfxxuwbdslkwifmmcsiusiuosws  \
0                                            0
1                                            1
2                                            1
3                                            1
4                                            1

   origin_up_ldkssxwpmemidmecebumciepifcamkci  \
0                                            0
1                                            0
2                                            0
3                                            0
4                                            0

   origin_up_lxidpiddsbxsbosboudacockeimpuepw
0                                            1
1                                            0
2                                            0
3                                            0
4                                            0

[5 rows x 61 columns]
```

1.2.4 Transforming numerical data

In the previous exercise we saw that some variables were highly skewed. The reason why we need to treat skewness is because some predictive models have inherent assumptions about the distribution of the features that are being supplied to it. Such models are called `parametric` models, and they typically assume that all variables are both independent and normally distributed.

Skewness isn't always a bad thing, but as a rule of thumb it is always good practice to treat highly skewed variables because of the reason stated above, but also as it can improve the speed at which predictive models are able to converge to its best solution.

There are many ways that you can treat skewed variables. You can apply transformations such as: - Square root - Cubic root - Logarithm

to a continuous numeric column and you will notice the distribution changes. For this use case we will use the 'Logarithm' transformation for the positively skewed features.

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

First I want to see the statistics of the skewed features, so that we can compare before and after transformation

[23]:
```python
skewed = [
    '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',
    'forecast_price_energy_peak',
    'forecast_price_pow_off_peak'
]

df[skewed].describe()
```

[23]:
```
           cons_12m  cons_gas_12m  cons_last_month  forecast_cons_12m  \
count  1.460600e+04  1.460600e+04     14606.000000       14606.000000
mean   1.592203e+05  2.809238e+04     16090.269752        1868.614880
std    5.734653e+05  1.629731e+05     64364.196422        2387.571531
min    0.000000e+00  0.000000e+00         0.000000           0.000000
25%    5.674750e+03  0.000000e+00         0.000000         494.995000
50%    1.411550e+04  0.000000e+00       792.500000        1112.875000
75%    4.076375e+04  0.000000e+00      3383.000000        2401.790000
max    6.207104e+06  4.154590e+06    771203.000000       82902.830000

       forecast_cons_year  forecast_discount_energy  forecast_meter_rent_12m  \
count        14606.000000              14606.000000             14606.000000
mean          1399.762906                  0.966726                63.086871
std           3247.786255                  5.108289                66.165783
min              0.000000                  0.000000                 0.000000
25%              0.000000                  0.000000                16.180000
50%            314.000000                  0.000000                18.795000
75%           1745.750000                  0.000000               131.030000
max         175375.000000                 30.000000               599.310000

       forecast_price_energy_off_peak  forecast_price_energy_peak  \
count                    14606.000000                14606.000000
mean                         0.137283                    0.050491
std                          0.024623                    0.049037
min                          0.000000                    0.000000
25%                          0.116340                    0.000000
50%                          0.143166                    0.084138
75%                          0.146348                    0.098837
max                          0.273963                    0.195975

       forecast_price_pow_off_peak
count                 14606.000000
```

```
mean                      43.130056
std                        4.485988
min                        0.000000
25%                       40.606701
50%                       44.311378
75%                       44.311378
max                       59.266378
```

We can see that the standard deviation for most of these features is quite high.

[24]:
```python
# 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["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)
```

[25]:
```python
df[skewed].describe()
```

[25]:
```
           cons_12m  cons_gas_12m  cons_last_month  forecast_cons_12m  \
count  14606.000000  14606.000000     14606.000000       14606.000000
mean       4.223939      0.779244         2.264646           2.962177
std        0.884515      1.717071         1.769305           0.683592
min        0.000000      0.000000         0.000000           0.000000
25%        3.754023      0.000000         0.000000           2.695477
50%        4.149727      0.000000         2.899547           3.046836
75%        4.610285      0.000000         3.529430           3.380716
max        6.792889      6.618528         5.887169           4.918575

       forecast_cons_year  forecast_discount_energy  forecast_meter_rent_12m  \
count        14606.000000              14606.000000             14606.000000
mean             1.784610                  0.966726                 1.517203
std              1.584986                  5.108289                 0.571481
min              0.000000                  0.000000                 0.000000
25%              0.000000                  0.000000                 1.235023
50%              2.498311                  0.000000                 1.296555
75%              3.242231                  0.000000                 2.120673
max              5.243970                 30.000000                 2.778376

       forecast_price_energy_off_peak  forecast_price_energy_peak  \
count                    14606.000000                14606.000000
mean                         0.137283                    0.050491
std                          0.024623                    0.049037
min                          0.000000                    0.000000
25%                          0.116340                    0.000000
50%                          0.143166                    0.084138
```

```
75%                          0.146348                      0.098837
max                          0.273963                      0.195975

       forecast_price_pow_off_peak
count                 14606.000000
mean                     43.130056
std                       4.485988
min                       0.000000
25%                      40.606701
50%                      44.311378
75%                      44.311378
max                      59.266378
```

Now we can see that for the majority of the features, their standard deviation is much lower after transformation. This is a good thing, it shows that these features are more stable and predictable now.

Let's quickly check the distributions of some of these features too.

1.2.5 Transforming categorical data

A predictive model cannot accept categorical or `string` values, hence as a data scientist you need to encode categorical features into numerical representations in the most compact and discriminative way possible.

The simplest method is to map each category to an integer (label encoding), however this is not always appropriate beecause it then introduces the concept of an order into a feature which may not inherently be present `0 < 1 < 2 < 3 ...`

Another way to encode categorical features is to use `dummy variables` AKA `one hot encoding`. This create a new feature for every unique value of a categorical column, and fills this column with either a 1 or a 0 to indicate that this company does or does not belong to this category.

channel_sales

[21]:
```python
# Transform into categorical type
df['channel_sales'] = df['channel_sales'].astype('category')

# Let's see how many categories are within this column
df['channel_sales'].value_counts()
```

[21]:
```
foosdfpfkusacimwkcsosbicdxkicaua    6754
MISSING                             3725
lmkebamcaaclubfxadlmueccxoimlema    1843
usilxuppasemubllopkaafesmlibmsdf    1375
ewpakwlliwisiwduibdlfmalxowmwpci     893
sddiedcslfslkckwlfkdpoeeailfpeds      11
epumfxlbckeskwekxbiuasklxalciiuu       3
fixdbufsefwooaasfcxdxadsiekoceaa       2
Name: channel_sales, dtype: int64
```

We have 8 categories, so we will create 8 dummy variables from this column. However, as you can see the last 3 categories in the output above, show that they only have 11, 3 and 2 occurrences respectively. Considering that our dataset has about 14000 rows, this means that these dummy variables will be almost entirely 0 and so will not add much predictive power to the model at all (since they're almost entirely a constant value and provide very little).

For this reason, we will drop these 3 dummy variables.

[22]:
```
df = pd.get_dummies(df, columns=['channel_sales'], prefix='channel')
df = df.drop(columns=['channel_sddiedcslfslkckwlfkdpoeeailfpeds',␣
↪'channel_epumfxlbckeskwekxbiuasklxalciiuu',␣
↪'channel_fixdbufsefwooaasfcxdxadsiekoceaa'])
df.head()
```

[22]:
```
                                 id  cons_12m  cons_gas_12m  cons_last_month  \
0  24011ae4ebbe3035111d65fa7c15bc57         0         54946                0
1  d29c2c54acc38ff3c0614d0a653813dd      4660             0                0
2  764c75f661154dac3a6c254cd082ea7d       544             0                0
3  bba03439a292a1e166f80264c16191cb      1584             0                0
4  149d57cf92fc41cf94415803a877cb4b      4425             0              526

   forecast_cons_12m  forecast_cons_year  forecast_discount_energy  \
0               0.00                   0                       0.0
1             189.95                   0                       0.0
2              47.96                   0                       0.0
3             240.04                   0                       0.0
4             445.75                 526                       0.0

   forecast_meter_rent_12m  forecast_price_energy_off_peak  \
0                     1.78                        0.114481
1                    16.27                        0.145711
2                    38.72                        0.165794
3                    19.83                        0.146694
4                   131.73                        0.116900

   forecast_price_energy_peak  …  tenure  months_activ  months_to_end  \
0                    0.098142  …       3            30              5
1                    0.000000  …       7            76              7
2                    0.087899  …       6            68              3
3                    0.000000  …       6            69              2
4                    0.100015  …       6            71              2

   months_modif_prod  months_renewal  channel_MISSING  \
0                  2               6                0
1                 76               4                1
2                 68               8                0
3                 69               9                0
4                 71               9                1
```

```
   channel_ewpakwlliwisiwduibdlfmalxowmwpci  \
0                                         0
1                                         0
2                                         0
3                                         0
4                                         0

   channel_foosdfpfkusacimwkcsosbicdxkicaua  \
0                                         1
1                                         0
2                                         1
3                                         0
4                                         0

  channel_lmkebamcaaclubfxadlmueccxoimlema  \
0                                        0
1                                        0
2                                        0
3                                        1
4                                        0

   channel_usilxuppasemubllopkaafesmlibmsdf
0                                         0
1                                         0
2                                         0
3                                         0
4                                         0

[5 rows x 63 columns]
```

origin_up

[23]:
```
# Transform into categorical type
df['origin_up'] = df['origin_up'].astype('category')

# Let's see how many categories are within this column
df['origin_up'].value_counts()
```

[23]:
```
lxidpiddsbxsbosboudacockeimpuepw    7097
kamkkxfxxuwbdslkwifmmcsiusiuosws    4294
ldkssxwpmemidmecebumciepifcamkci    3148
MISSING                               64
usapbepcfoloekilkwsdiboslwaxobdp       2
ewxeelcelemmiwuafmddpobolfuxioce       1
Name: origin_up, dtype: int64
```

Similar to `channel_sales` the last 3 categories in the output above show very low frequency, so we will remove these from the features after creating dummy variables.

[24]:
```
df = pd.get_dummies(df, columns=['origin_up'], prefix='origin_up')
df = df.drop(columns=['origin_up_MISSING',␣
↪'origin_up_usapbepcfoloekilkwsdiboslwaxobdp',␣
↪'origin_up_ewxeelcelemmiwuafmddpobolfuxioce'])
df.head()
```

[24]:
```
                                 id  cons_12m  cons_gas_12m  cons_last_month  \
0  24011ae4ebbe3035111d65fa7c15bc57         0         54946                0
1  d29c2c54acc38ff3c0614d0a653813dd      4660             0                0
2  764c75f661154dac3a6c254cd082ea7d       544             0                0
3  bba03439a292a1e166f80264c16191cb      1584             0                0
4  149d57cf92fc41cf94415803a877cb4b      4425             0              526

   forecast_cons_12m  forecast_cons_year  forecast_discount_energy  \
0               0.00                   0                       0.0
1             189.95                   0                       0.0
2              47.96                   0                       0.0
3             240.04                   0                       0.0
4             445.75                 526                       0.0

   forecast_meter_rent_12m  forecast_price_energy_off_peak  \
0                     1.78                        0.114481
1                    16.27                        0.145711
2                    38.72                        0.165794
3                    19.83                        0.146694
4                   131.73                        0.116900

   forecast_price_energy_peak  …  months_modif_prod  months_renewal  \
0                    0.098142  …                  2               6
1                    0.000000  …                 76               4
2                    0.087899  …                 68               8
3                    0.000000  …                 69               9
4                    0.100015  …                 71               9

   channel_MISSING  channel_ewpakwlliwisiwduibdlfmalxowmwpci  \
0                0                                         0
1                1                                         0
2                0                                         0
3                0                                         0
4                1                                         0

   channel_foosdfpfkusacimwkcsosbicdxkicaua  \
0                                         1
1                                         0
2                                         1
3                                         0
4                                         0
```

```
   channel_lmkebamcaaclubfxadlmueccxoimlema  \
0                                        0
1                                        0
2                                        0
3                                        1
4                                        0

   channel_usilxuppasemubllopkaafesmlibmsdf  \
0                                        0
1                                        0
2                                        0
3                                        0
4                                        0

   origin_up_kamkkxfxxuwbdslkwifmmcsiusiuosws  \
0                                          0
1                                          1
2                                          1
3                                          1
4                                          1

   origin_up_ldkssxwpmemidmecebumciepifcamkci  \
0                                          0
1                                          0
2                                          0
3                                          0
4                                          0

   origin_up_lxidpiddsbxsbosboudacockeimpuepw
0                                          1
1                                          0
2                                          0
3                                          0
4                                          0

[5 rows x 65 columns]
```

1.2.6 Transforming numerical data

In the previous exercise we saw that some variables were highly skewed. The reason why we need to treat skewness is because some predictive models have inherent assumptions about the distribution of the features that are being supplied to it. Such models are called `parametric` models, and they typically assume that all variables are both independent and normally distributed.

Skewness isn't always a bad thing, but as a rule of thumb it is always good practice to treat highly skewed variables because of the reason stated above, but also as it can improve the speed at which predictive models are able to converge to its best solution.

There are many ways that you can treat skewed variables. You can apply transformations such as: - Square root - Cubic root - Logarithm

to a continuous numeric column and you will notice the distribution changes. For this use case we will use the 'Logarithm' transformation for the positively skewed features.

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

First I want to see the statistics of the skewed features, so that we can compare before and after transformation

[25]:
```python
skewed = [
    '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',
    'forecast_price_energy_peak',
    'forecast_price_pow_off_peak'
]

df[skewed].describe()
```

[25]:
```
           cons_12m  cons_gas_12m  cons_last_month  forecast_cons_12m  \
count  1.460600e+04  1.460600e+04     14606.000000       14606.000000
mean   1.592203e+05  2.809238e+04     16090.269752        1868.614880
std    5.734653e+05  1.629731e+05     64364.196422        2387.571531
min    0.000000e+00  0.000000e+00         0.000000           0.000000
25%    5.674750e+03  0.000000e+00         0.000000         494.995000
50%    1.411550e+04  0.000000e+00       792.500000        1112.875000
75%    4.076375e+04  0.000000e+00      3383.000000        2401.790000
max    6.207104e+06  4.154590e+06    771203.000000       82902.830000

       forecast_cons_year  forecast_discount_energy  forecast_meter_rent_12m  \
count        14606.000000              14606.000000             14606.000000
mean          1399.762906                  0.966726                63.086871
std           3247.786255                  5.108289                66.165783
min              0.000000                  0.000000                 0.000000
25%              0.000000                  0.000000                16.180000
50%            314.000000                  0.000000                18.795000
75%           1745.750000                  0.000000               131.030000
max         175375.000000                 30.000000               599.310000

       forecast_price_energy_off_peak  forecast_price_energy_peak  \
count                    14606.000000                14606.000000
mean                         0.137283                    0.050491
```

```
std                         0.024623                     0.049037
min                         0.000000                     0.000000
25%                         0.116340                     0.000000
50%                         0.143166                     0.084138
75%                         0.146348                     0.098837
max                         0.273963                     0.195975

       forecast_price_pow_off_peak
count                 14606.000000
mean                     43.130056
std                       4.485988
min                       0.000000
25%                      40.606701
50%                      44.311378
75%                      44.311378
max                      59.266378
```

We can see that the standard deviation for most of these features is quite high.

[26]:
```python
# 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["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)
```

[27]:
```python
df[skewed].describe()
```

[27]:
```
           cons_12m  cons_gas_12m  cons_last_month  forecast_cons_12m  \
count  14606.000000  14606.000000     14606.000000       14606.000000
mean       4.223939      0.779244         2.264646           2.962177
std        0.884515      1.717071         1.769305           0.683592
min        0.000000      0.000000         0.000000           0.000000
25%        3.754023      0.000000         0.000000           2.695477
50%        4.149727      0.000000         2.899547           3.046836
75%        4.610285      0.000000         3.529430           3.380716
max        6.792889      6.618528         5.887169           4.918575

       forecast_cons_year  forecast_discount_energy  forecast_meter_rent_12m  \
count        14606.000000              14606.000000             14606.000000
mean             1.784610                  0.966726                 1.517203
std              1.584986                  5.108289                 0.571481
min              0.000000                  0.000000                 0.000000
25%              0.000000                  0.000000                 1.235023
50%              2.498311                  0.000000                 1.296555
75%              3.242231                  0.000000                 2.120673
```

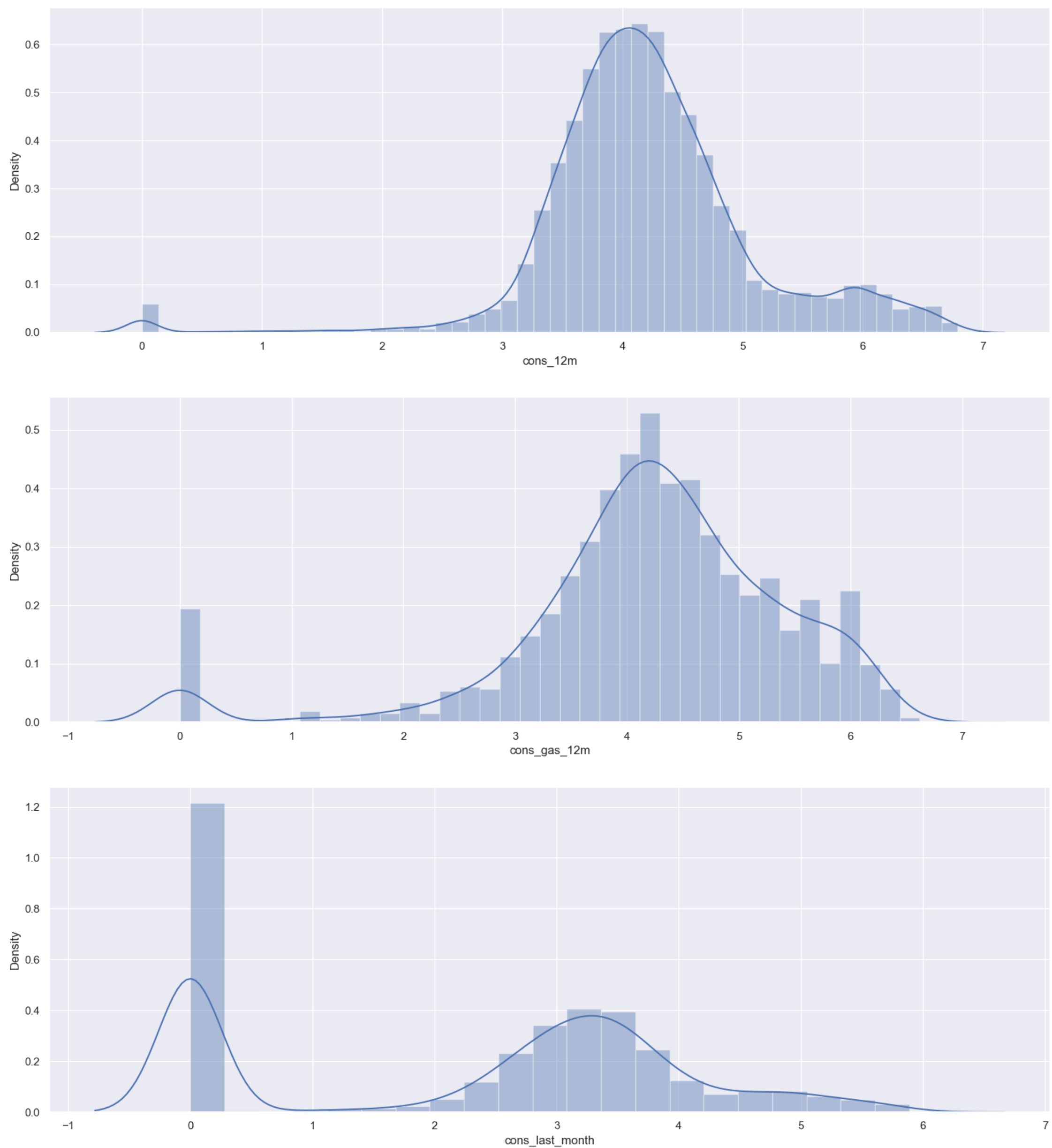
```
max              5.243970                    30.000000                  2.778376

       forecast_price_energy_off_peak  forecast_price_energy_peak  \
count                    14606.000000                14606.000000
mean                         0.137283                    0.050491
std                          0.024623                    0.049037
min                          0.000000                    0.000000
25%                          0.116340                    0.000000
50%                          0.143166                    0.084138
75%                          0.146348                    0.098837
max                          0.273963                    0.195975

       forecast_price_pow_off_peak
count                 14606.000000
mean                     43.130056
std                       4.485988
min                       0.000000
25%                      40.606701
50%                      44.311378
75%                      44.311378
max                      59.266378
```

Now we can see that for the majority of the features, their standard deviation is much lower after transformation. This is a good thing, it shows that these features are more stable and predictable now.

Let's quickly check the distributions of some of these features too.

[26]:

```
fig, axs = plt.subplots(nrows=3, figsize=(18, 20))
# Plot histograms
sns.distplot((df["cons_12m"].dropna()), ax=axs[0])
sns.distplot((df[df["has_gas"]==1]["cons_gas_12m"].dropna()), ax=axs[1])
sns.distplot((df["cons_last_month"].dropna()), ax=axs[2])
plt.show()
```

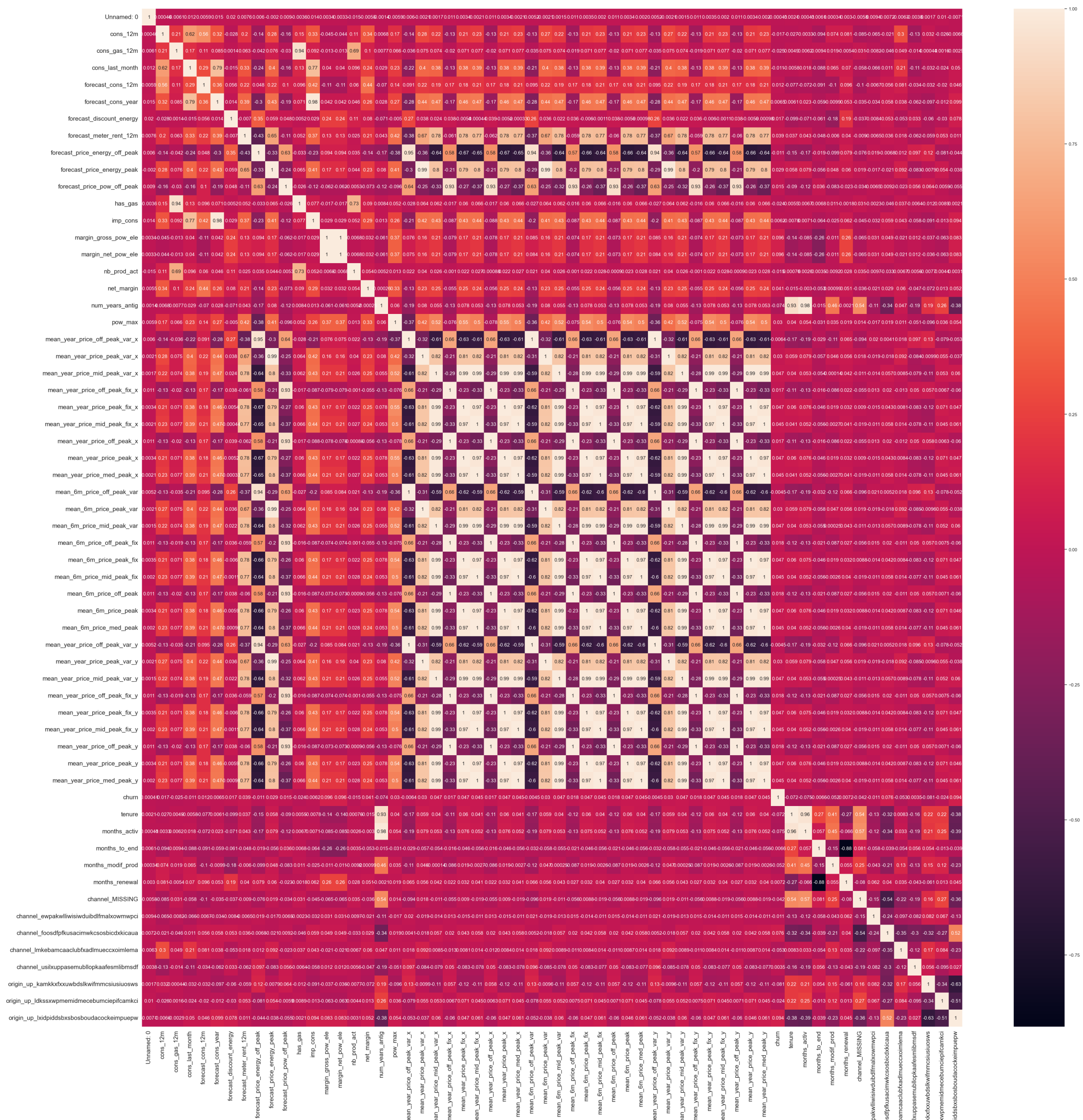


[27]:
```
#Correlation
```

[28]:
```
correlation = df.corr()
```

[29]:
```
# Plot correlation
plt.figure(figsize=(45, 45))
sns.heatmap(
    correlation,
    xticklabels=correlation.columns.values,
    yticklabels=correlation.columns.values,
```

```
    annot=True,
    annot_kws={'size': 12}
)
# Axis ticks size
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



1.3 5. Modelling

We now have a dataset containing features that we have engineered and we are ready to start training a predictive model. Remember, we only need to focus on training a `Random Forest` classifier.

[30]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

1.3.1 Data sampling

The first thing we want to do is split our dataset into training and test samples. The reason why we do this, is so that we can simulate a real life situation by generating predictions for our test sample, without showing the predictive model these data points. This gives us the ability to see how well our model is able to generalise to new data, which is critical.

A typical % to dedicate to testing is between 20-30, for this example we will use a 75-25% split between train and test respectively.

[31]:
```
# Make a copy of our data
train_df = df.copy()

# Separate target variable from independent variables
y = df['churn']
X = df.drop(columns=['id', 'churn'])
print(X.shape)
print(y.shape)
```

```
(14606, 59)
(14606,)
```

[32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(10954, 59)
(10954,)
(3652, 59)
(3652,)
```

1.3.2 Model training

Once again, we are using a `Random Forest` classifier in this example. A Random Forest sits within the category of `ensemble` algorithms because internally the `Forest` refers to a collection of `Decision Trees` which are tree-based learning algorithms. As the data scientist, you can control how large the forest is (that is, how many decision trees you want to include).

The reason why an `ensemble` algorithm is powerful is because of the laws of averaging, weak learners and the central limit theorem. If we take a single decision tree and give it a sample of data and some parameters, it will learn patterns from the data. It may be overfit or it may be underfit, but that is now our only hope, that single algorithm.

With `ensemble` methods, instead of banking on 1 single trained model, we can train 1000's of decision trees, all using different splits of the data and learning different patterns. It would be like asking 1000 people to all learn how to code. You would end up with 1000 people with different answers, methods and styles! The weak learner notion applies here too, it has been found that if you train your learners not to overfit, but to learn weak patterns within the data and you have a lot of these weak learners, together they come together to form a highly predictive pool of knowledge! This is a real life application of many brains are better than 1.

Now instead of relying on 1 single decision tree for prediction, the random forest puts it to the overall views of the entire collection of decision trees. Some ensemble algorithms using a voting approach to decide which prediction is best, others using averaging.

As we increase the number of learners, the idea is that the random forest's performance should converge to its best possible solution.

Some additional advantages of the random forest classifier include:

- The random forest uses a rule-based approach instead of a distance calculation and so features do not need to be scaled
- It is able to handle non-linear parameters better than linear based models

On the flip side, some disadvantages of the random forest classifier include:

- The computational power needed to train a random forest on a large dataset is high, since we need to build a whole ensemble of estimators.
- Training time can be longer due to the increased complexity and size of thee ensemble

[33]:
```
model = RandomForestClassifier(
    n_estimators=1000
)
model.fit(X_train, y_train)
```

[33]:
```
RandomForestClassifier(n_estimators=1000)
```

The `scikit-learn` documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, has a lot of information about the algorithm and the parameters that you can use when training a model.

For this example, I am using `n_estimators` = 1000. This means that my random forest will consist of 1000 decision trees. There are many more parameters that you can fine-tune within the random forest and finding the optimal combinations of parameters can be a manual task of exploration, trial and error, which will not be covered during this notebook.

1.3.3 Evaluation

Now let's evaluate how well this trained model is able to predict the values of the test dataset.

We are going to use 3 metrics to evaluate performance:

- Accuracy = the ratio of correctly predicted observations to the total observations
- Precision = the ability of the classifier to not label a negative sample as positive
- Recall = the ability of the classifier to find all the positive samples

The reason why we are using these three metrics is because a simple accuracy is not always a good measure to use. To give an example, let's say you're predicting heart failures with patients in a hospital and there were 100 patients out of 1000 that did have a heart failure.

If you predicted 80 out of 100 (80%) of the patients that did have a heart failure correctly, you might think that you've done well! However, this also means that you predicted 20 wrong and what may the implications of predicting these remaining 20 patients wrong? Maybe they miss out on getting vital treatment to save their lives.

As well as this, what about the impact of predicting negative cases as positive (people not having heart failure being predicted that they did), maybe a high number of false positives means that resources get used up on thee wrong people and a lot of time is wasted when they could have been helping the real heart failure sufferers.

This is just an example, but it illustrates why other performance metrics are necessary such `Precision` and `Recall`, which are good measures to use in a classification scenario.

[34]:
```
predictions = model.predict(X_test)
tn, fp, fn, tp = metrics.confusion_matrix(y_test, predictions).ravel()
```

[35]:
```
y_test.value_counts()
```

[35]:
```
0    3286
1     366
Name: churn, dtype: int64
```

[36]:
```
print(f"True positives: {tp}")
print(f"False positives: {fp}")
print(f"True negatives: {tn}")
print(f"False negatives: {fn}\n")

print(f"Accuracy: {metrics.accuracy_score(y_test, predictions)}")
print(f"Precision: {metrics.precision_score(y_test, predictions)}")
print(f"Recall: {metrics.recall_score(y_test, predictions)}")
```

```
True positives: 18
False positives: 1
True negatives: 3285
False negatives: 348

Accuracy: 0.9044359255202629
Precision: 0.9473684210526315
Recall: 0.04918032786885246
```

Looking at these results there are a few things to point out:

Note: If you are running this notebook yourself, you may get slightly different answers!

- Within the test set about 10% of the rows are churners (churn = 1).
- Looking at the true negatives, we have 3282 out of 3286. This means that out of all the negative cases (churn = 0), we predicted 3282 as negative (hence the name True negative). This is great!
- Looking at the false negatives, this is where we have predicted a client to not churn (churn = 0) when in fact they did churn (churn = 1). This number is quite high at 348, we want to get the false negatives to as close to 0 as we can, so this would need to be addressed when improving the model.
- Looking at false positives, this is where we have predicted a client to churn when they actually didnt churn. For this value we can see there are 4 cases, which is great!
- With the true positives, we can see that in total we have 366 clients that churned in the test dataset. However, we are only able to correctly identify 18 of those 366, which is very poor.
- Looking at the accuracy score, this is very misleading! Hence the use of precision and recall is important. The accuracy score is high, but it does not tell us the whole story.
- Looking at the precision score, this shows us a score of 0.82 which is not bad, but could be improved.
- However, the recall shows us that the classifier has a very poor ability to identify positive samples. This would be the main concern for improving this model!

So overall, we're able to very accurately identify clients that do not churn, but we are not able to predict cases where clients do churn! What we are seeing is that a high % of clients are being identified as not churning when they should be identified as churning. This in turn tells me that the current set of features are not discriminative enough to clearly distinguish between churners and non-churners.

A data scientist at this point would go back to feature engineering to try and create more predictive features. They may also experiment with optimising the parameters within the model to improve performance. For now, lets dive into understanding the model a little more.

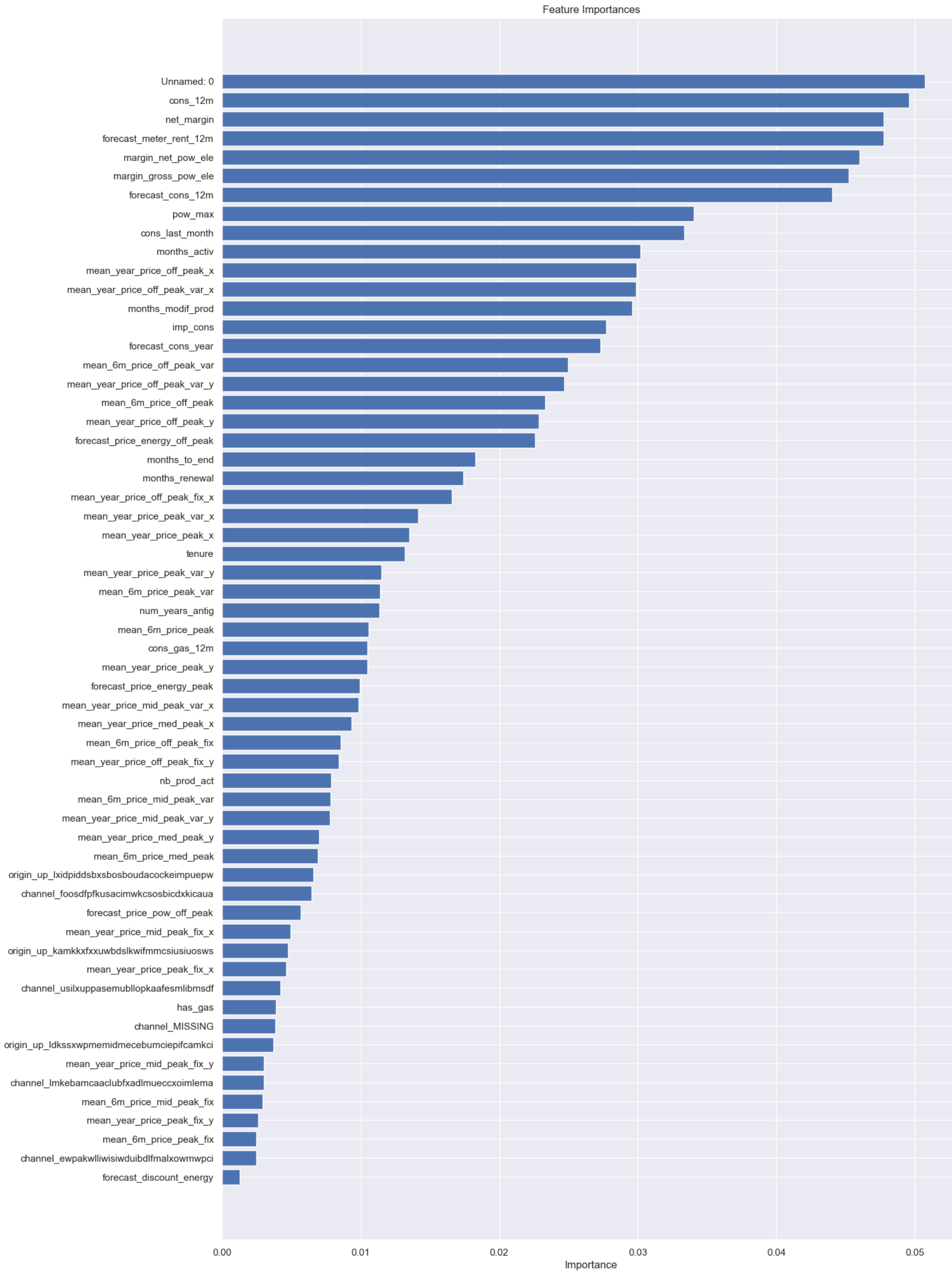
1.3.4 Model understanding

A simple way of understanding the results of a model is to look at feature importances. Feature importances indicate the importance of a feature within the predictive model, there are several ways to calculate feature importance, but with the Random Forest classifier, we're able to extract feature importances using the built-in method on the trained model. In the Random Forest case, the feature importance represents the number of times each feature is used for splitting across all trees.

[37]:
```
feature_importances = pd.DataFrame({
    'features': X_train.columns,
    'importance': model.feature_importances_
}).sort_values(by='importance', ascending=True).reset_index()
```

[38]:
```
plt.figure(figsize=(15, 25))
plt.title('Feature Importances')
plt.barh(range(len(feature_importances)), feature_importances['importance'],␣
 ↪color='b', align='center')
plt.yticks(range(len(feature_importances)), feature_importances['features'])
plt.xlabel('Importance')
```

```
plt.show()
```



From this chart, we can observe the following points:

- Net margin and consumption over 12 months is a top driver for churn in this model
- Margin on power subscription also is an influential driver
- Time seems to be an influential factor, especially the number of months they have been active, their tenure and the number of months since they updated their contract
- The feature that our colleague recommended is in the top half in terms of how influential it is and some of the features built off the back of this actually outperform it
- Our price sensitivity features are scattered around but are not the main driver for a customer churning

The last observation is important because this relates back to our original hypothesis:

`> Is churn driven by the customers' price sensitivity?`

Based on the output of the feature importances, it is not a main driver but it is a weak contributor. However, to arrive at a conclusive result, more experimentation is needed.

[39]:
```
proba_predictions = model.predict_proba(X_test)
probabilities = proba_predictions[:, 1]
```

[40]:
```
X_test = X_test.reset_index()
X_test.drop(columns='index', inplace=True)
```

[42]:
```
X_test['churn'] = predictions.tolist()
X_test['churn_probability'] = probabilities.tolist()
X_test.to_csv('output_sample_data_with_predictions.csv')
```