I am working on customer churn project where we need to predict if the customer will leave or stay with the company in future based on the features provided. This is a clssification problem as the outcome is binary.

We learned how to read any unknown data with different visualization skills. We learned how to make prediction model whether it may be a classification problem or a regression problem. As a student in Big data analytics I believe the knowledge I gained is a vary good starting point and base to learn everyting in more detail in other subjects.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns
```

In [2]:

```
df = pd.read_csv('ppg_churn.csv')
```

In [3]:

```
df.shape
```

Out[3]:

```
(5000, 20)
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   state   5000 non-null   object 
 1   X02     5000 non-null   int64  
 2   X03     5000 non-null   object 
 3   X04     5000 non-null   object 
 4   X05     5000 non-null   object 
 5   X06     5000 non-null   int64  
 6   X07     5000 non-null   float64
 7   X08     5000 non-null   int64  
 8   X09     5000 non-null   float64
 9   X10     5000 non-null   float64
 10  X11     5000 non-null   int64  
 11  X12     5000 non-null   float64
 12  X13     5000 non-null   float64
 13  X14     5000 non-null   int64  
 14  X15     5000 non-null   float64
 15  X16     5000 non-null   float64
 16  X17     5000 non-null   int64  
 17  X18     5000 non-null   float64
 18  X19     5000 non-null   int64  
 19  churn   5000 non-null   object 
dtypes: float64(8), int64(7), object(5)
memory usage: 781.4+ KB
```

In [5]:

```
df.describe()
```

Out[5]:

| | X02 | X06 | X07 | X08 | X09 | X10 | X11 | X12 | X1 |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 5000.00000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.00000 |
| **mean** | 100.25860 | 7.755200 | 180.288900 | 100.029400 | 30.649668 | 200.636560 | 100.191000 | 17.054322 | 200.39162 |
| **std** | 39.69456 | 13.546393 | 53.894699 | 19.831197 | 9.162069 | 50.551309 | 19.826496 | 4.296843 | 50.52778 |
| **min** | 1.00000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| **25%** | 73.00000 | 0.000000 | 143.700000 | 87.000000 | 24.430000 | 166.375000 | 87.000000 | 14.140000 | 166.90000 |
| **50%** | 100.00000 | 0.000000 | 180.100000 | 100.000000 | 30.620000 | 201.000000 | 100.000000 | 17.090000 | 200.40000 |
| **75%** | 127.00000 | 17.000000 | 216.200000 | 113.000000 | 36.750000 | 234.100000 | 114.000000 | 19.900000 | 234.70000 |
| **max** | 243.00000 | 52.000000 | 351.500000 | 165.000000 | 59.760000 | 363.700000 | 170.000000 | 30.910000 | 395.00000 |

In [6]:

```
df.isna().sum()
```

Out[6]:

```
state    0
X02      0
X03      0
X04      0
X05      0
X06      0
X07      0
X08      0
X09      0
X10      0
X11      0
X12      0
X13      0
X14      0
X15      0
X16      0
X17      0
X18      0
X19      0
churn    0
dtype: int64
```

In [7]:

```
df.isna().sum()
```

Out[7]:

```
state    0
X02      0
X03      0
X04      0
X05      0
X06      0
```

```
X07      0
X08      0
X09      0
X10      0
X11      0
X12      0
X13      0
X14      0
X15      0
X16      0
X17      0
X18      0
X19      0
churn    0
dtype: int64
```

In [8]:

```
df.X03.value_counts()
```

Out[8]:

```
AA    2495
BB    1259
CC    1246
Name: X03, dtype: int64
```

In [9]:

```
df.X04.value_counts()
```

Out[9]:

```
Z1    4527
Z2     473
Name: X04, dtype: int64
```

In [10]:

```
df.X05.value_counts()
```

Out[10]:

```
V2    3677
V1    1323
Name: X05, dtype: int64
```

In [11]:

```
df.churn.value_counts()
```

Out[11]:

```
no     4293
yes     707
Name: churn, dtype: int64
```

The state variable is an object data type. This it is an categorical variable even though it has 51 unique values

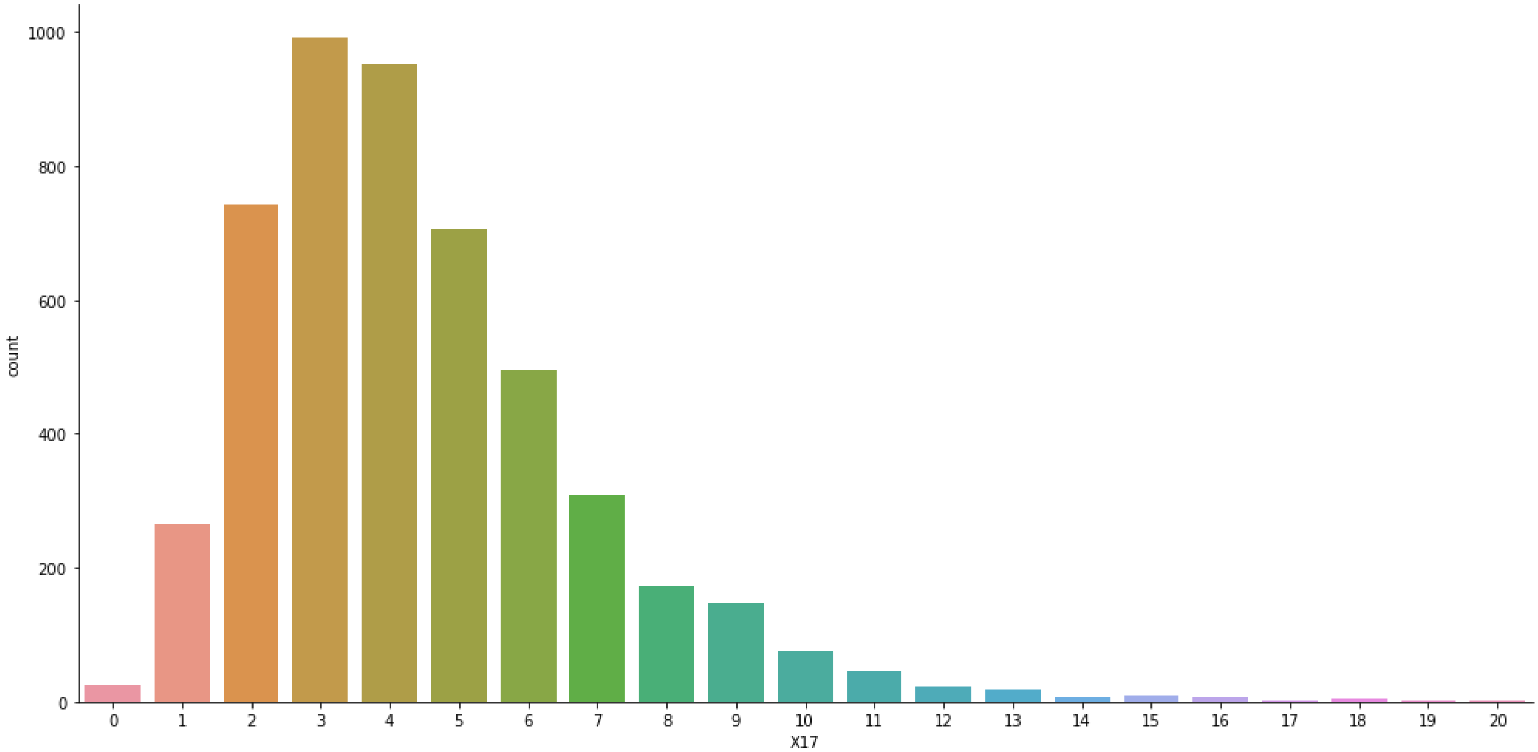
The `X03` , `X04` , `X05` have less than 4 unique values and the `.info()` method revealed these variables are also object data type. So they can also be assumed as categorical variables

The variables `X17` and `X19` have less than 25 unique values and both are integer data types.

In [12]:

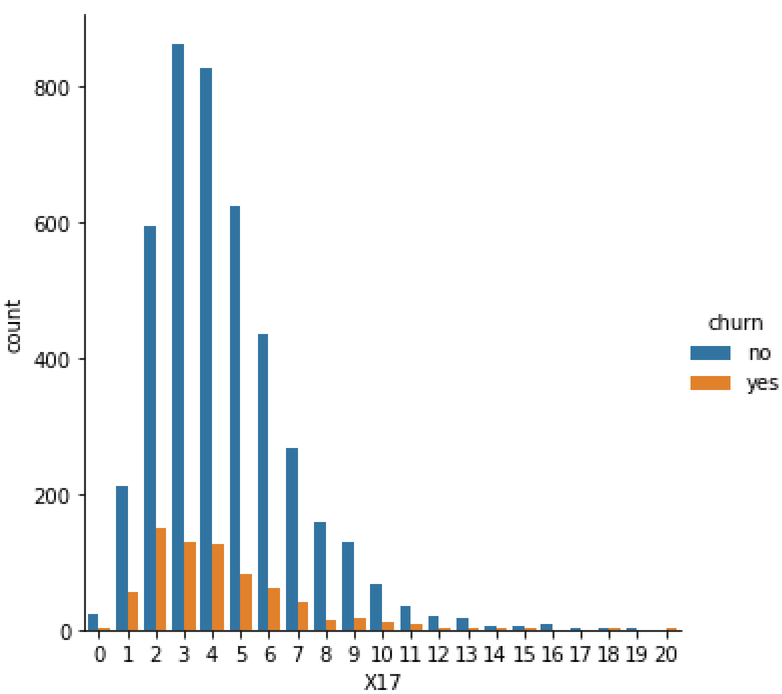
```
sns.catplot(data=df, x='X17', kind='count', height=7, aspect=2.0)
```

```
plt.show()
```




In [13]:

```
sns.catplot(data=df, x='X17', hue='churn', kind='count')

plt.show()
```

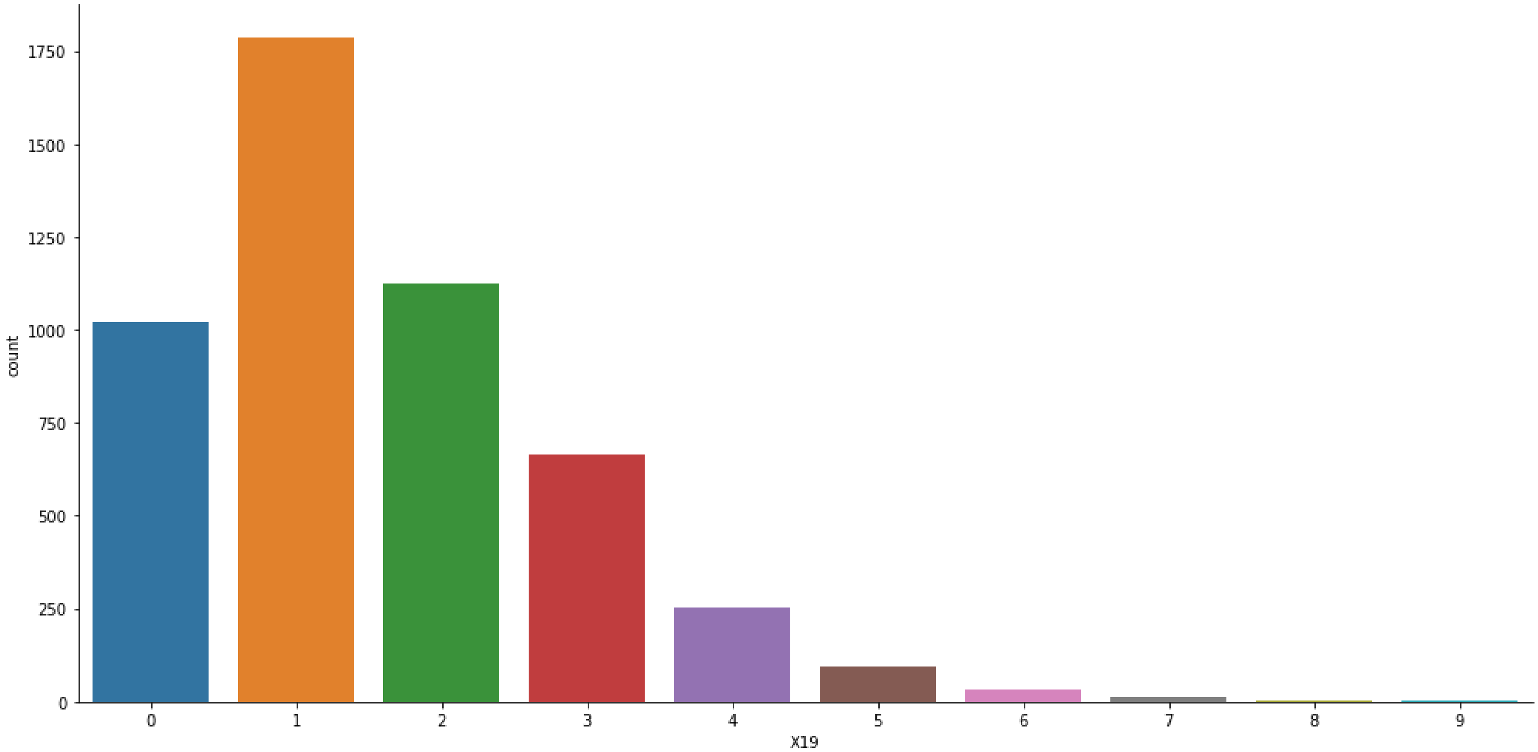



We can see that the values greater than 7 have very low frequencies.

In [14]:

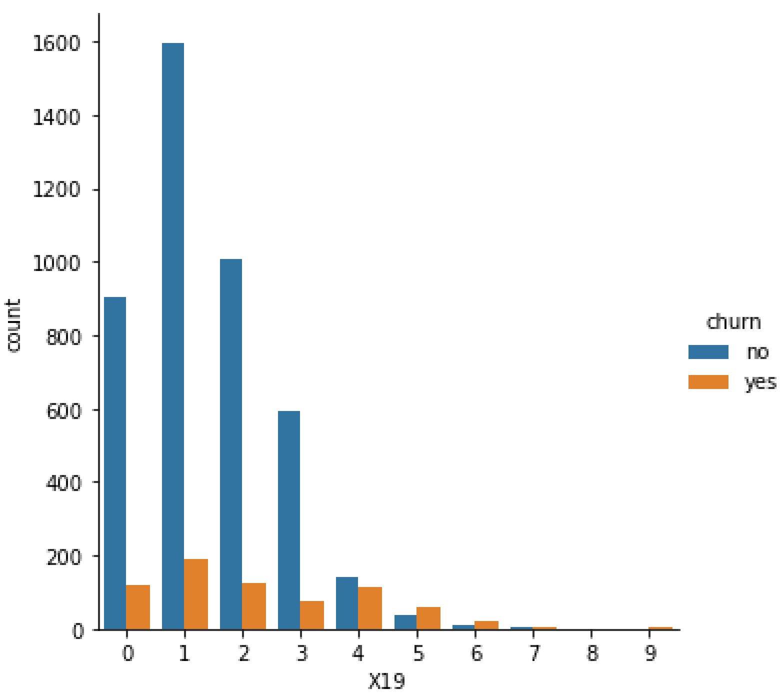
```
sns.catplot(data=df, x='X19', kind='count', height=7, aspect=2)
```

```
plt.show()
```




In [15]:

```
sns.catplot(data=df, x='X19', hue='churn', kind='count')

plt.show()
```




We can either take `X17` and `X19` as continuos or categorical.

If we want then to be continuous we dont have to change anything.

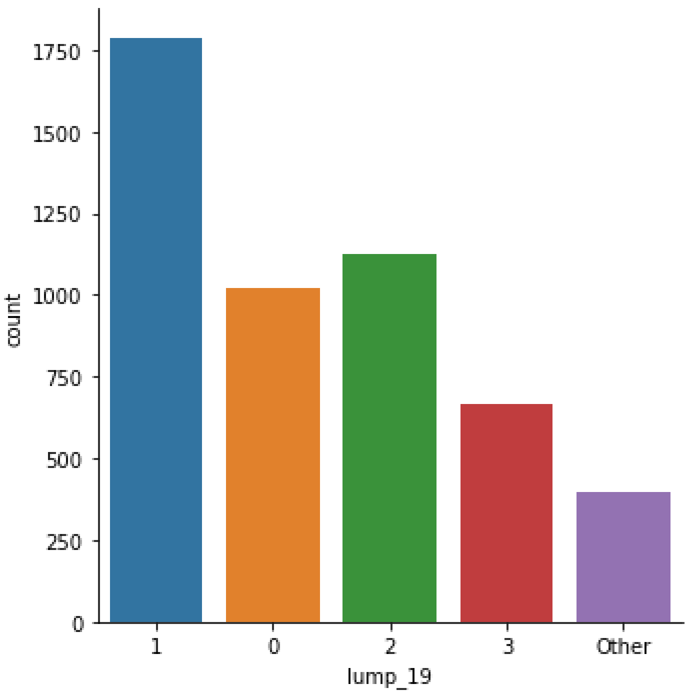
If we want them to be categorical we can lump then.

In [16]:

```
df['lump_19'] = np.where(df.X19 > 3, 'Other', df.X19.astype('str'))
```

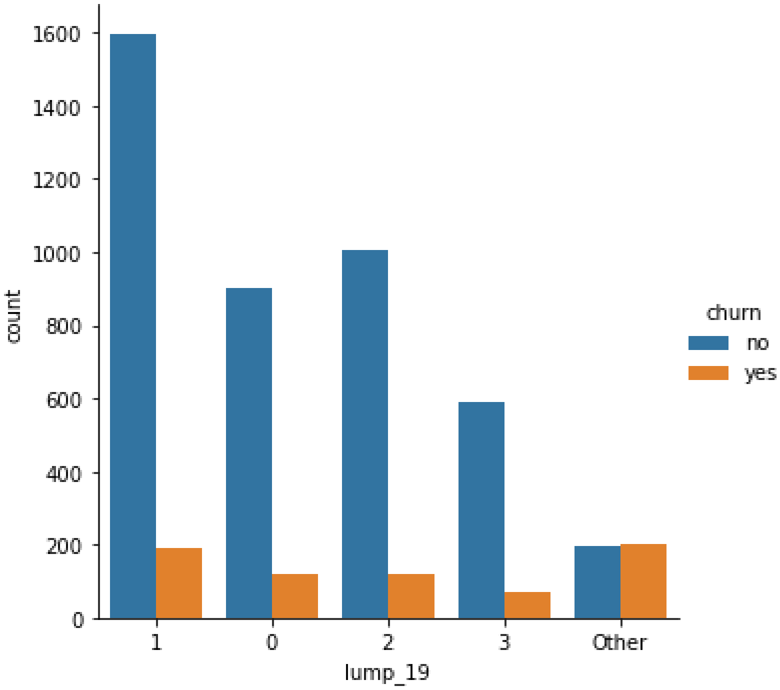
In [17]:

```
sns.catplot(data=df, x='lump_19', kind='count')

plt.show()
```




In [18]:

```
sns.catplot(data=df, x='lump_19', hue='churn', kind='count')

plt.show()
```




**For continuous inputs**

In [19]:

```
num_inputs = df.select_dtypes('number').copy().columns.to_list()
```

In [20]:

```
num_inputs
```

Out[20]:

```
['X02',
 'X06',
 'X07',
 'X08',
 'X09',
 'X10',
 'X11',
 'X12',
 'X13',
 'X14',
 'X15',
 'X16',
 'X17',
 'X18',
 'X19']
```

In [21]:

```
lf_num = df.melt(id_vars=['churn'], value_vars = num_inputs, ignore_index=True)
```

In [22]:

```
lf_num
```

Out[22]:

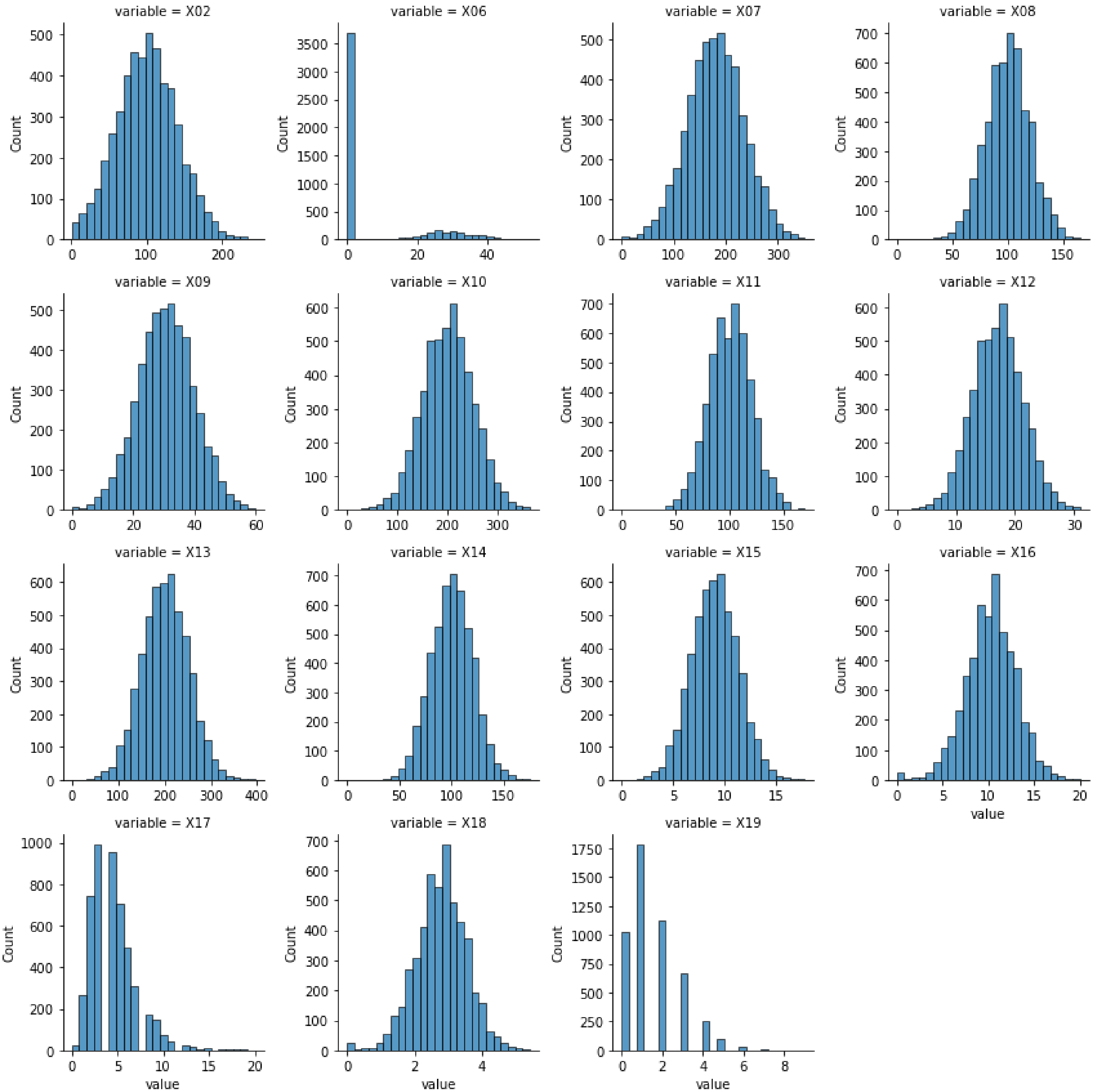
| churn | variable | value |
|---|---|---|

| | | | |
|---|---|---|---|
| **0** | no | X02 | 128.0 |
| **1** | no | X02 | 107.0 |
| **2** | no | X02 | 137.0 |
| **3** | no | X02 | 84.0 |
| **4** | no | X02 | 75.0 |
| **...** | ... | ... | ... |
| **74995** | no | X19 | 2.0 |
| **74996** | yes | X19 | 3.0 |
| **74997** | no | X19 | 1.0 |
| **74998** | no | X19 | 0.0 |
| **74999** | no | X19 | 0.0 |

75000 rows × 3 columns

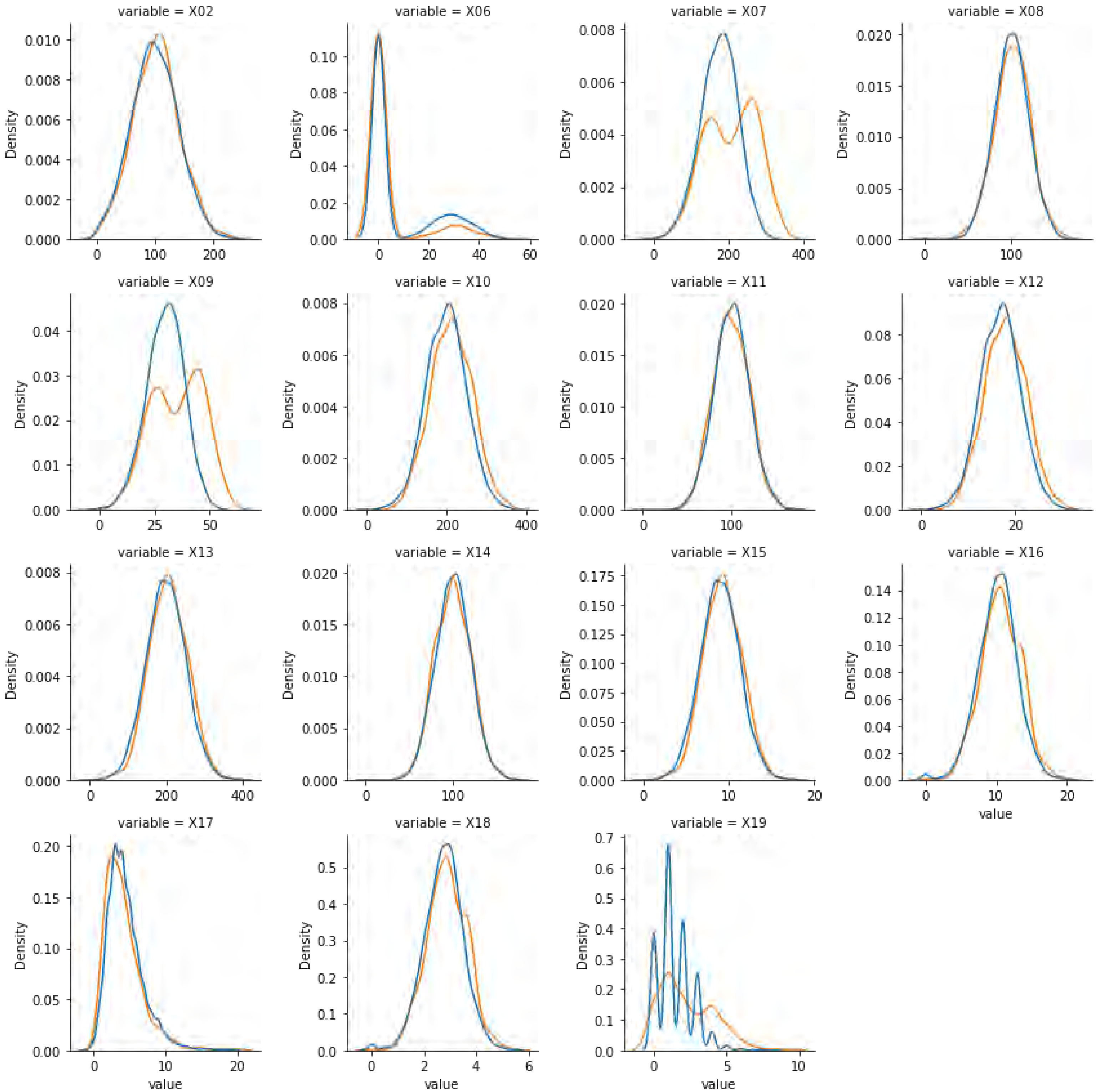
In [23]:

```
g = sns.FacetGrid(data= lf_num, col='variable', col_wrap=4, sharex=False, sharey=False)

g.map_dataframe(sns.histplot, x='value', bins=25)

plt.show()
```




In [24]:

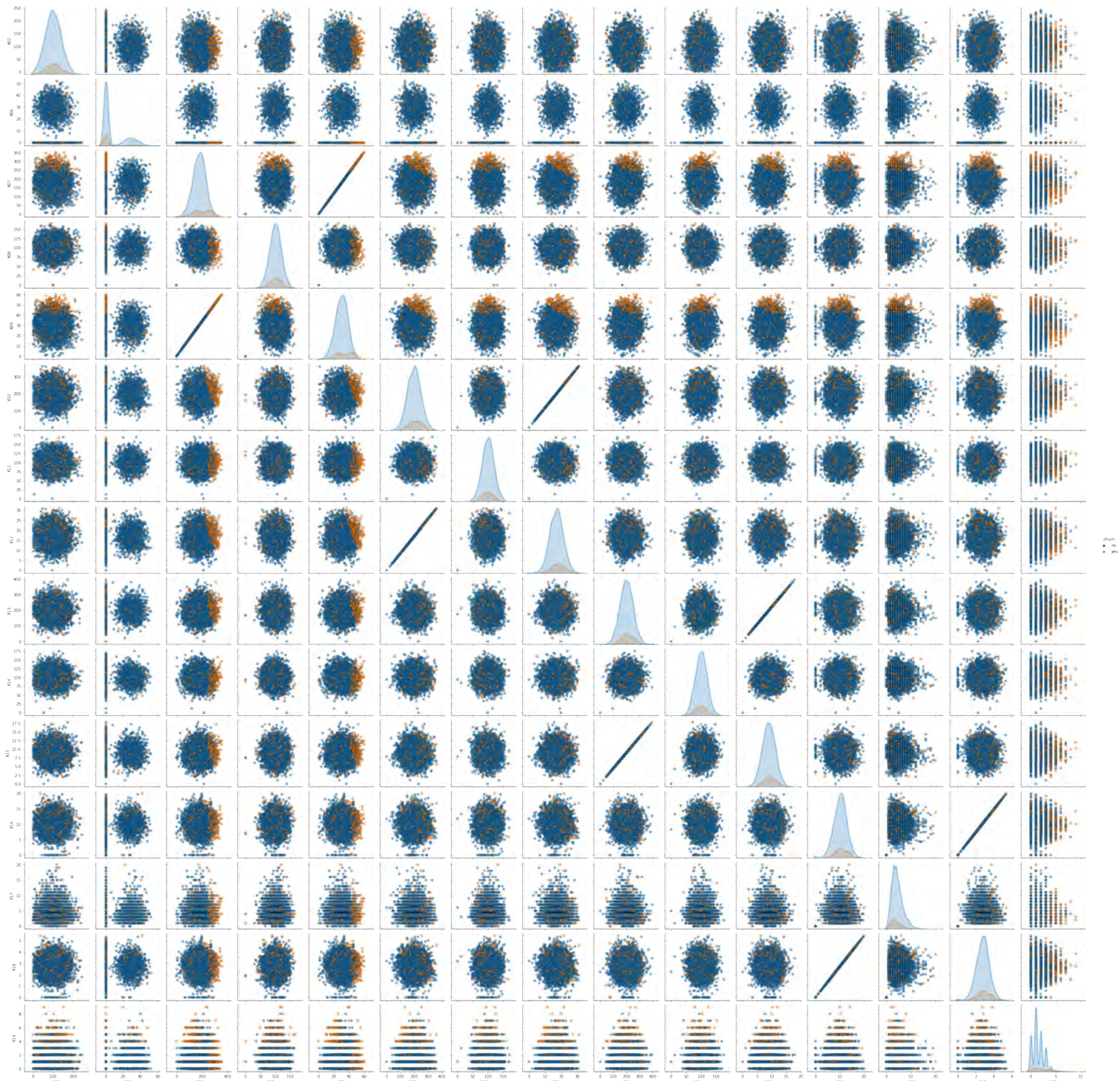
```
g = sns.FacetGrid(data= lf_num, col='variable', col_wrap=4, sharex=False, sharey=False)

g.map_dataframe(sns.kdeplot, x='value', hue='churn' ,common_norm=False)

plt.show()
```




The `X06` distribution is odd!

In [25]:

```
sns.pairplot(data=df, hue='churn', diag_kind = 'kde', plot_kws = {'alpha':0.6, 's':30, 'edgecolor':'k'})

plt.show()
```



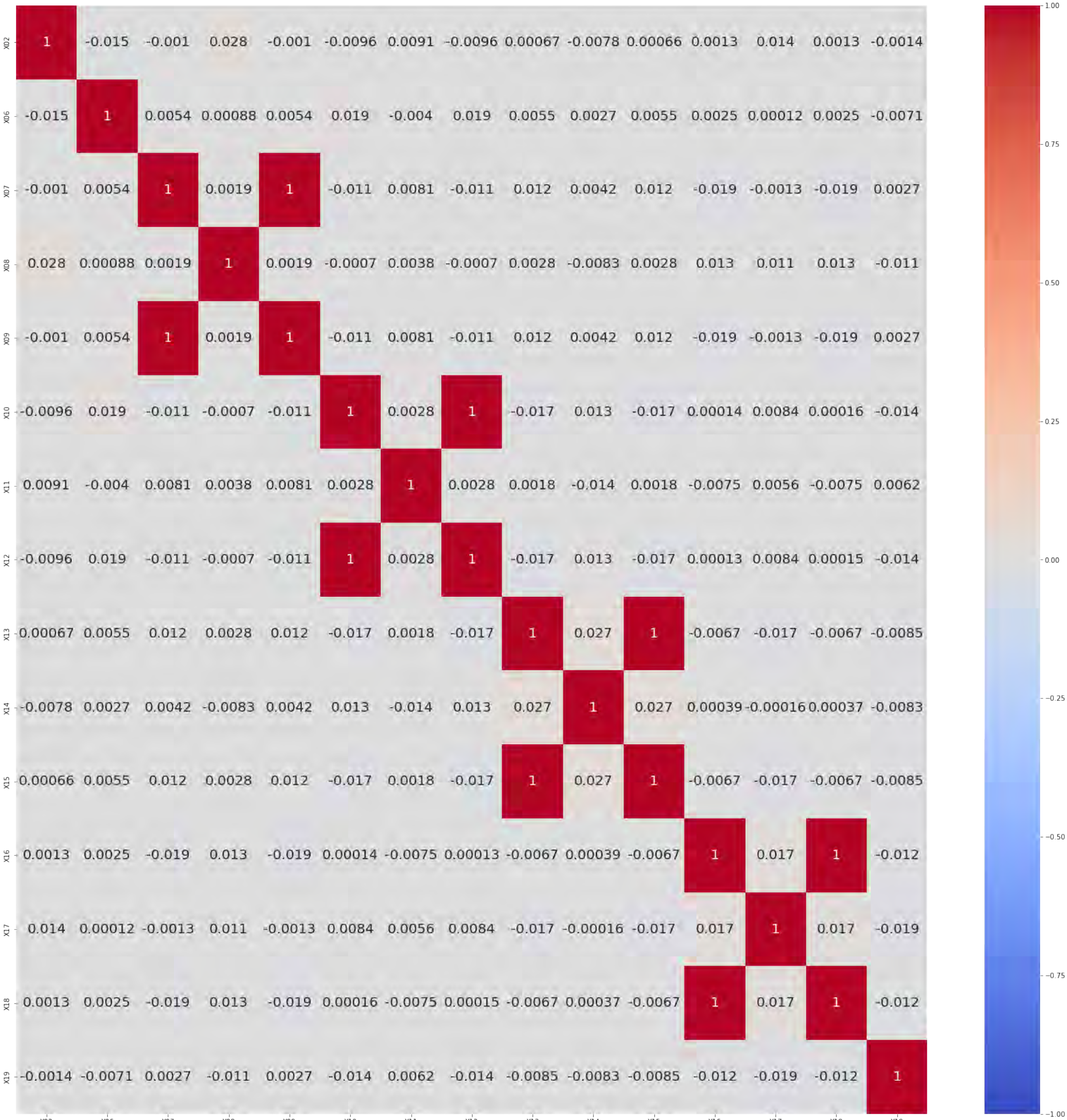
In `X07` and `x09` we can distinguish that the no responses are on the right side

## Correlation

In [26]:

```
fig, ax = plt.subplots(figsize=(30,30))

sns.heatmap(data = df.corr(),
            vmin=-1, vmax=1, center=0,
            cmap='coolwarm',
            annot=True, annot_kws={'size': 20},
```

```
                ax=ax)

plt.show()
```




From the above figure we can observe that

- X07 and X09
- X10 and X12

- X13 and X15
- X16 and X18

are in perfect correlation. That is they move in same direction together.

# Clustering

In [27]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from scipy.cluster import hierarchy
```

Converting the variable `X19` into categorical and treating `X06` and `X17` continuous

In [28]:

```
df['lump_x19'] = np.where( df.X19 > 3, 'Other', df.X19.astype('str'))
```

In [29]:

```
df['X06_trans'] = np.log(df.X06 + 0.001)
```

In [30]:

```
df = df.drop(columns=['X06', 'X19', 'lump_19'])
```

In [31]:

```
df_clean = df.select_dtypes('number').copy()
```

In [32]:

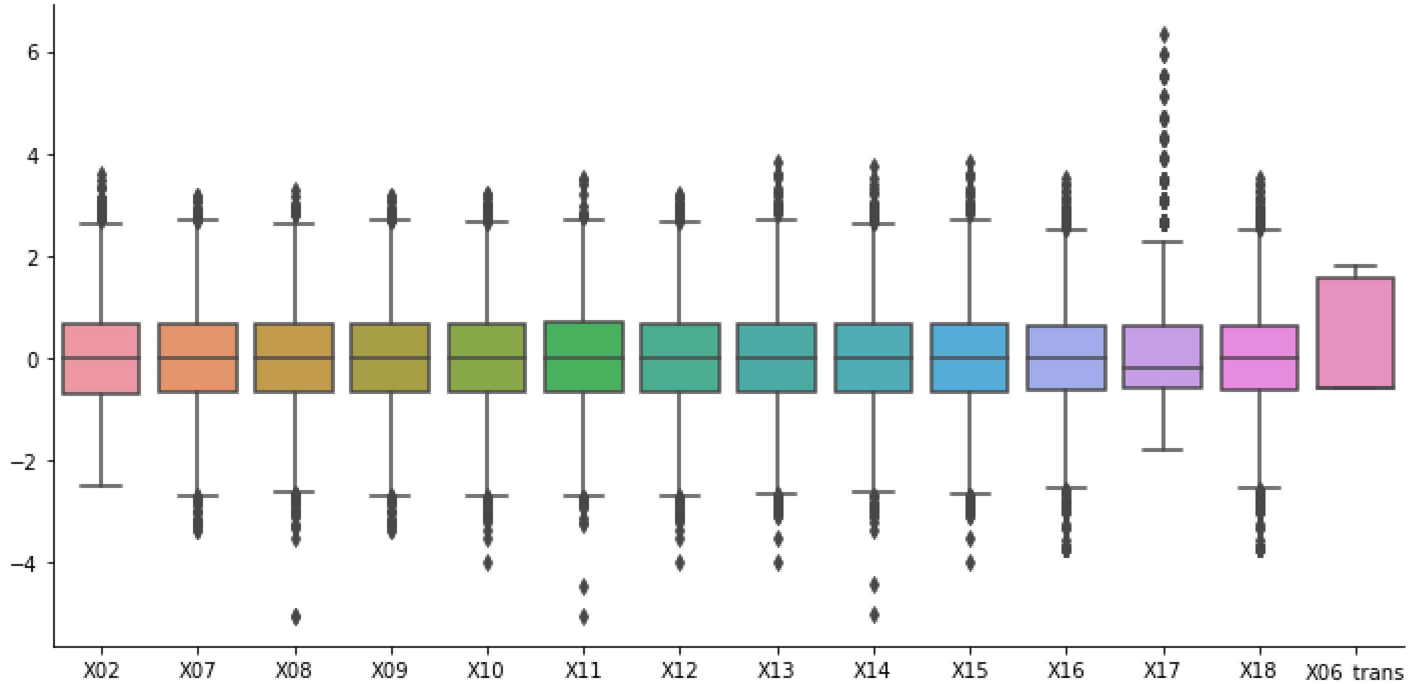
```
X = StandardScaler().fit_transform(df_clean)
```

In [33]:

```
df_stand = pd.DataFrame(X, columns=df_clean.columns)
```

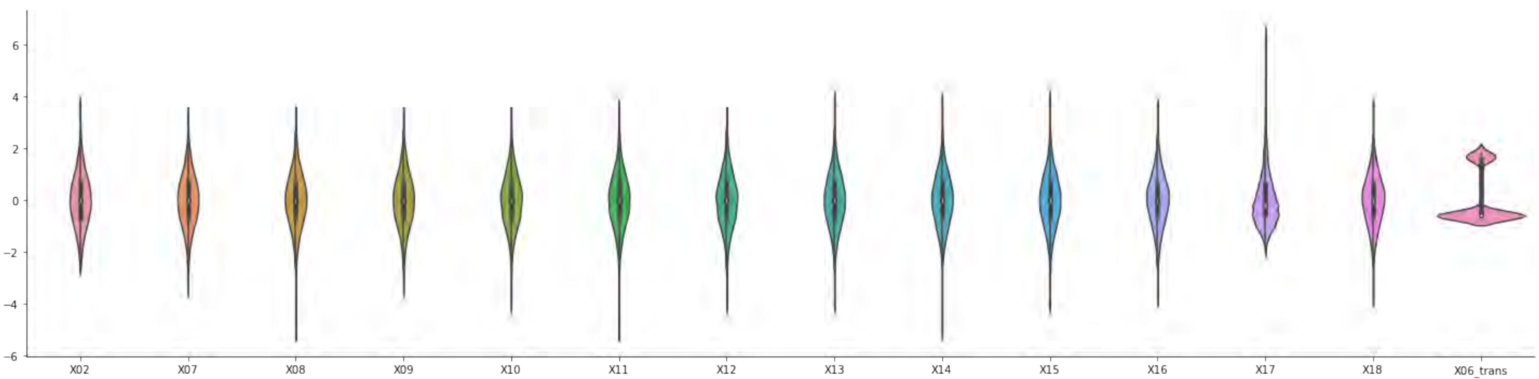
In [34]:

```
sns.catplot(data = pd.DataFrame(X, columns=df_clean.columns), kind='box', aspect=2)

plt.show()
```




In [35]:

```
sns.catplot(data = pd.DataFrame(X, columns=df_clean.columns), kind='violin', aspect=4)

plt.show()
```



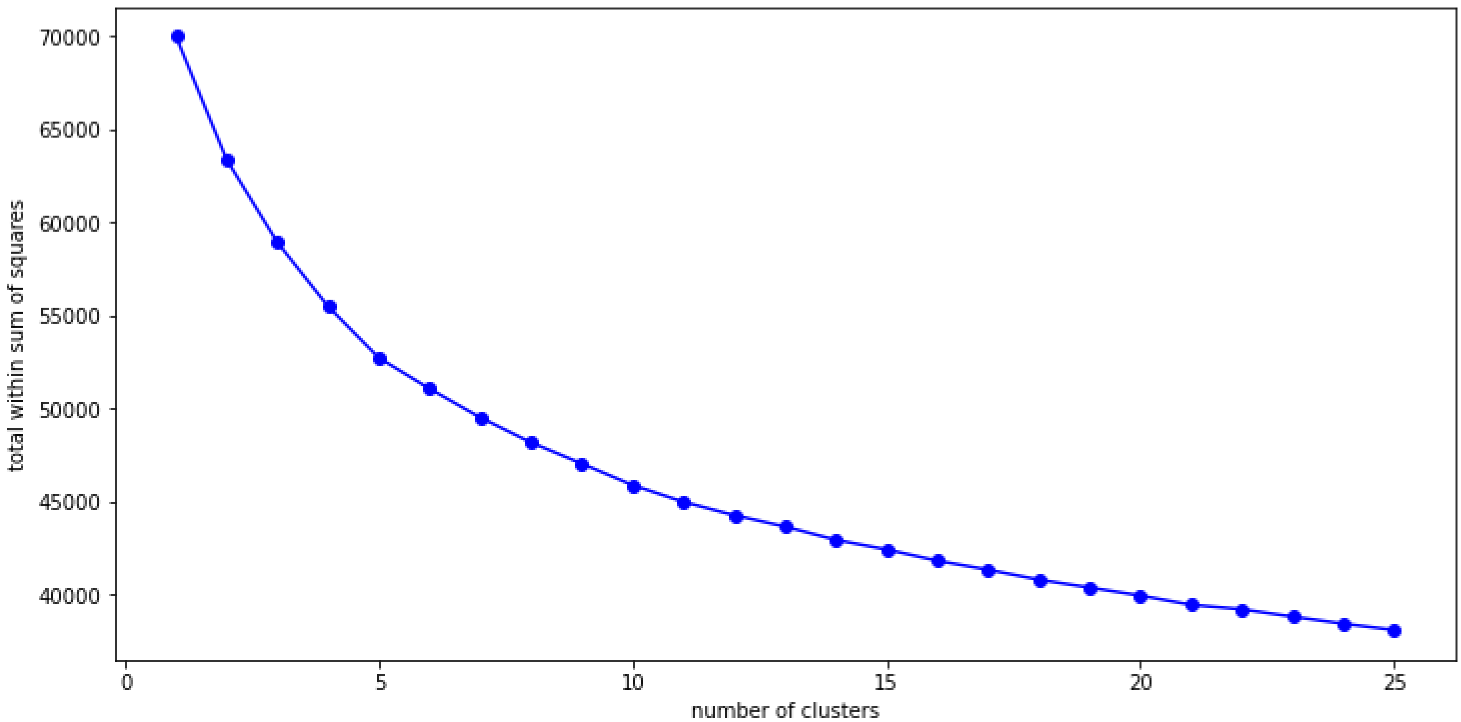

In [36]:

```
tots_within = []

K = range(1, 26)

for k in K:
    km = KMeans(n_clusters=k, random_state=121, n_init=25, max_iter=500)
    km = km.fit(X)
    tots_within.append(km.inertia_)
```

In [37]:

```
fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(K, tots_within, 'bo-')
ax.set_xlabel('number of clusters')
ax.set_ylabel('total within sum of squares')
```

```
plt.show()
```




In [38]:

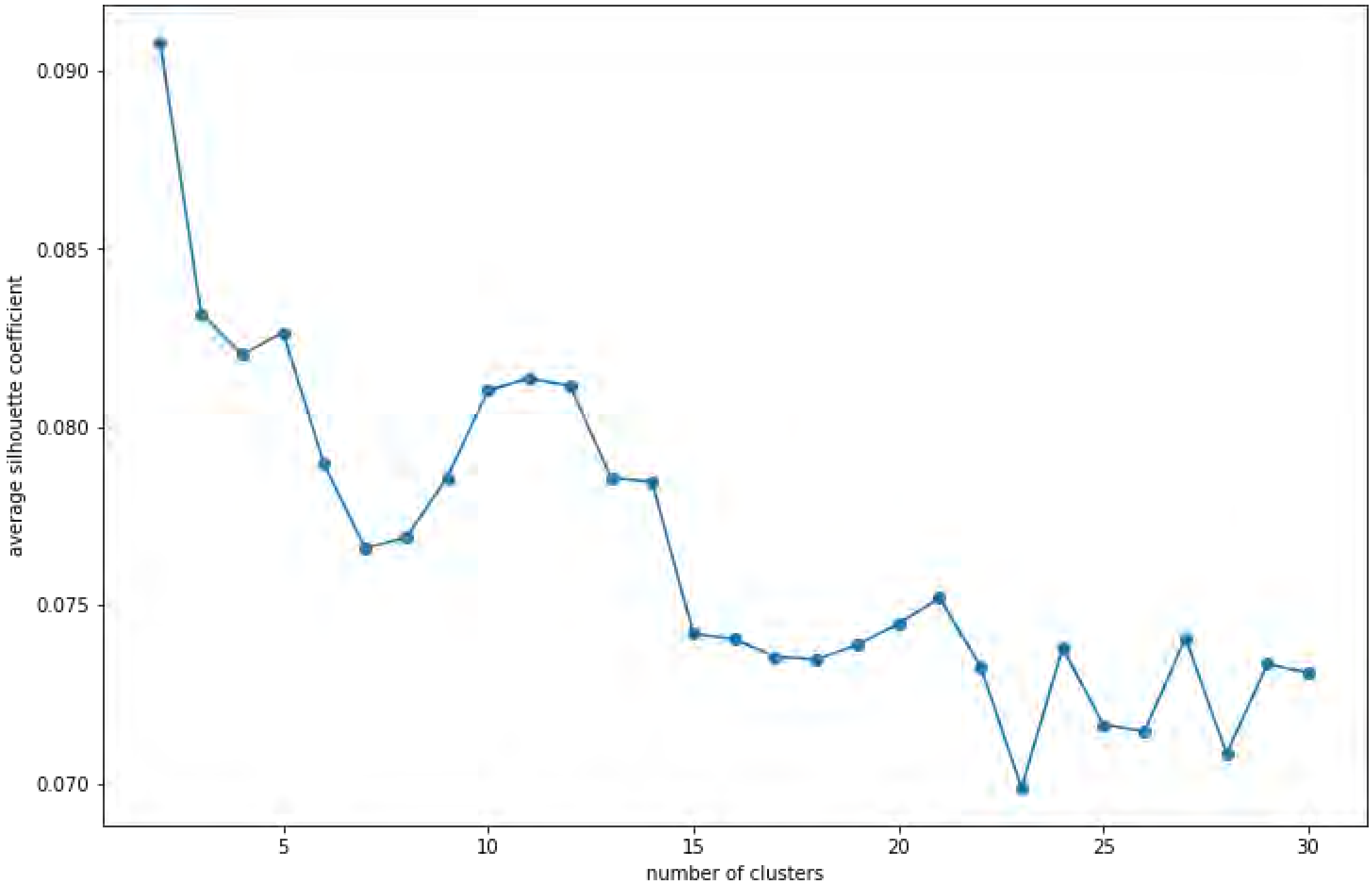
```
from sklearn.metrics import silhouette_score
```

In [39]:

```
sil_coef = []

K = range(2, 31)

for k in K:
    k_label = KMeans(n_clusters=k, random_state=121, n_init=25, max_iter=500).fit_predict(
X )
    sil_coef.append( silhouette_score(X, k_label) )
```

In [40]:

```
fig, ax = plt.subplots(figsize=(12, 8))

ax.plot(K, sil_coef, 'o-')
ax.set_xlabel('number of clusters')
ax.set_ylabel('average silhouette coefficient')

plt.show()
```




From this we cannot confirm how many clusters we have to take.

## PCA

In [41]:

```
from sklearn.decomposition import PCA
```

In [42]:

```
churn_pcs = PCA(n_components=2).fit_transform( X )
```

In [43]:

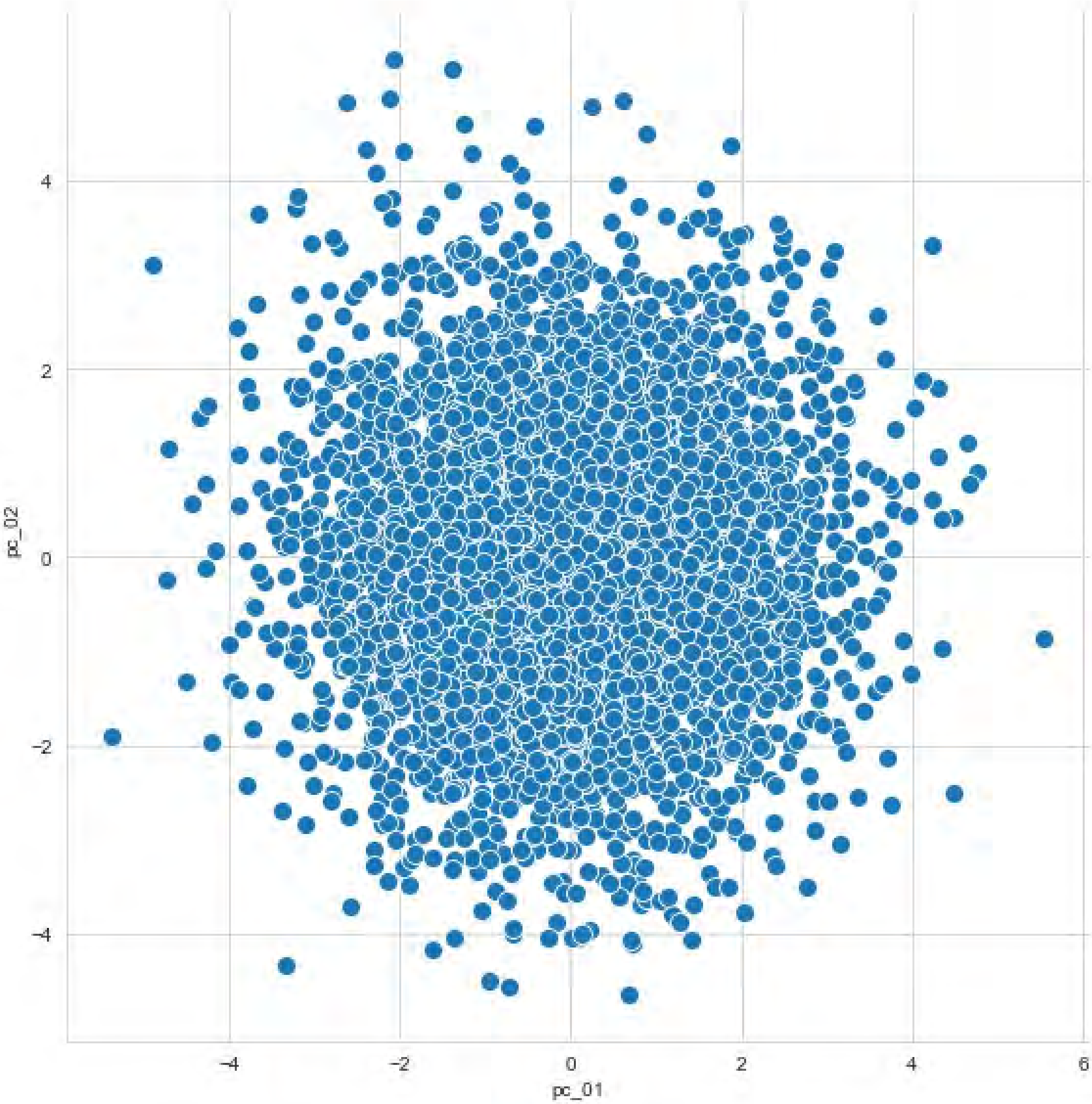
```
churn_pcs_df = pd.DataFrame( churn_pcs, columns=['pc_01', 'pc_02'])
```

In [44]:

```
churn_pcs_df['churn'] = df.churn
```

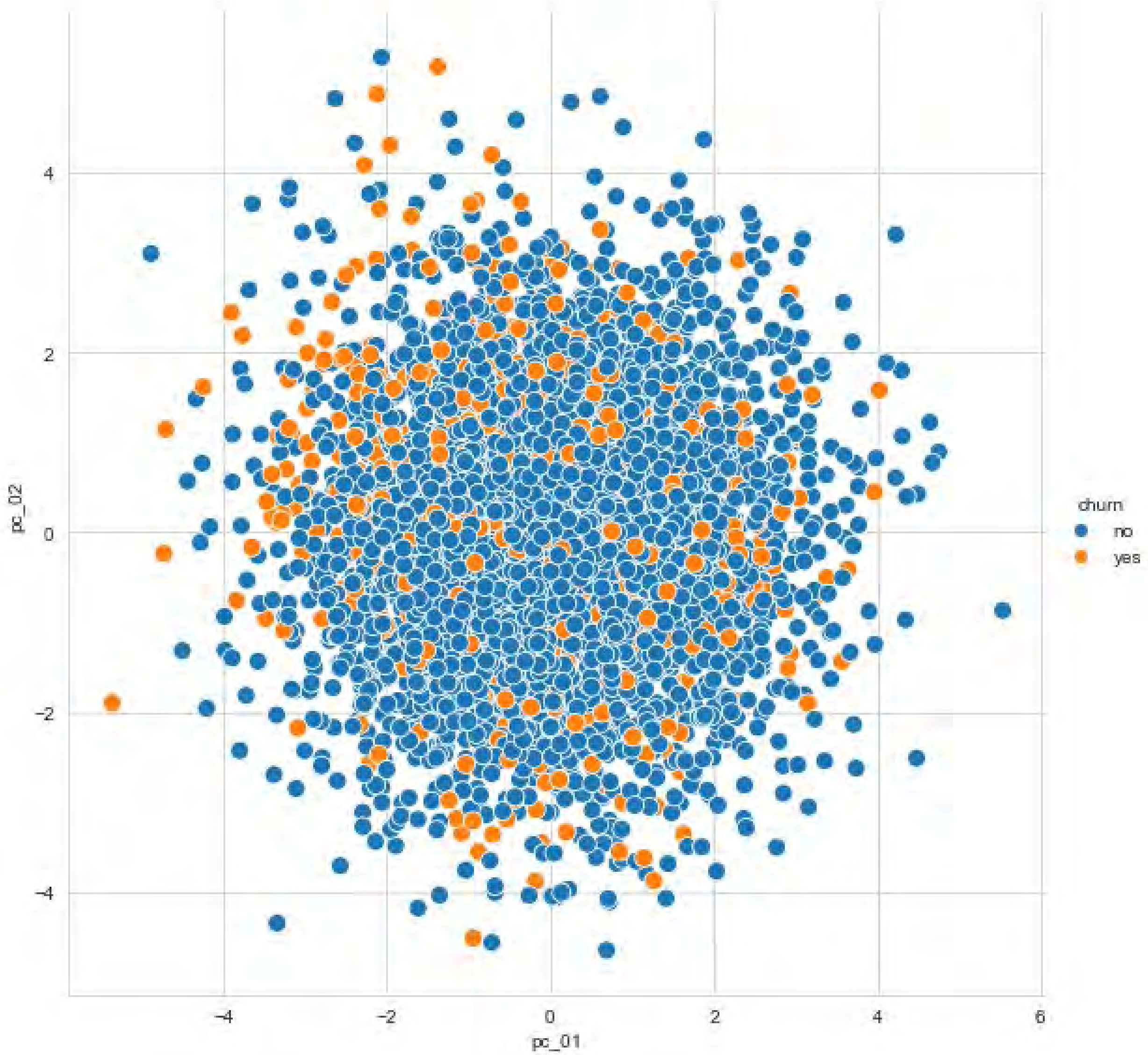
In [45]:

```
sns.set_style('whitegrid')
sns.relplot(data = churn_pcs_df, x='pc_01', y='pc_02', s=100, height=8)
plt.show()
```




In [46]:

```
sns.relplot(data = churn_pcs_df, x='pc_01', y='pc_02', hue='churn', s=100, height=8)

plt.show()
```




## Hierarchical clustering

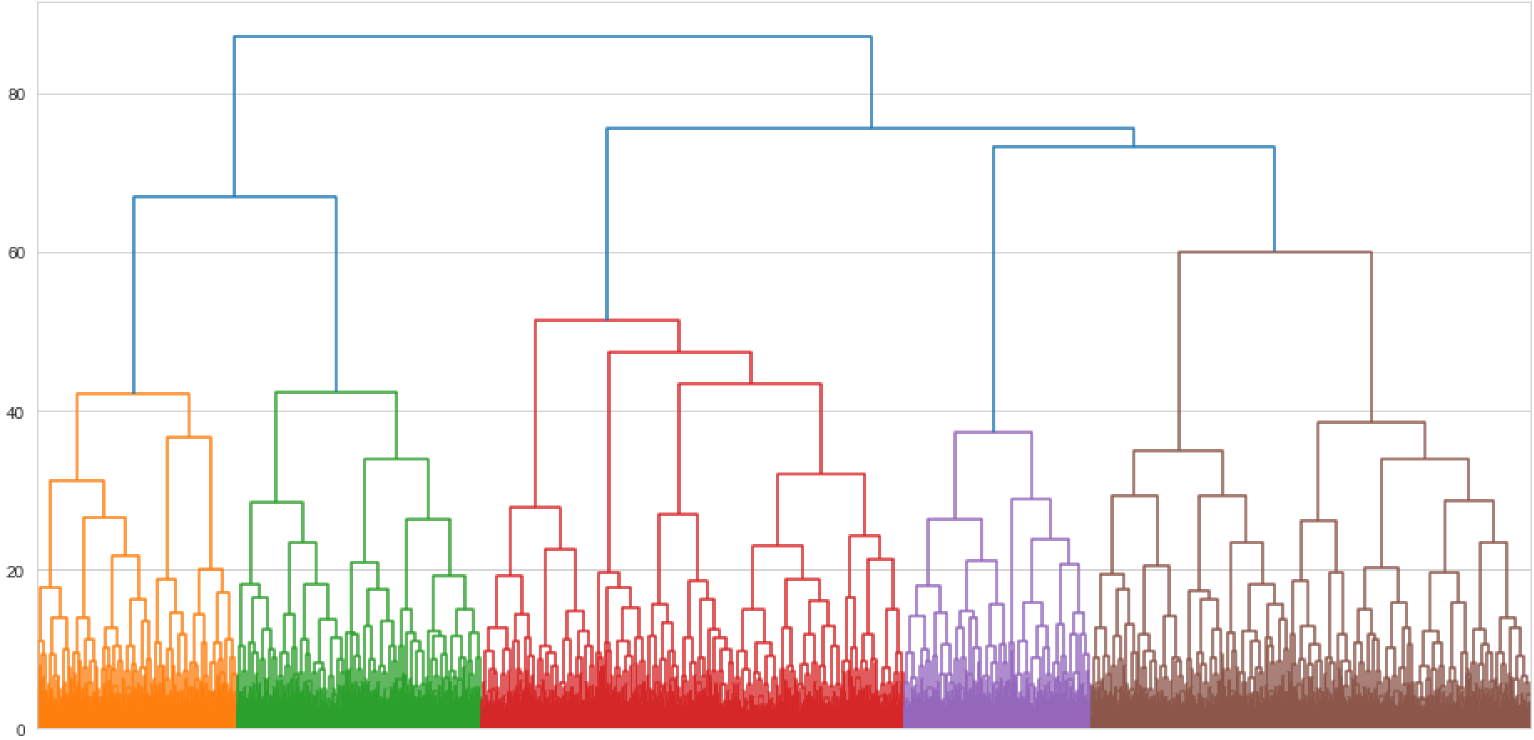
### Ward method

In [47]:

```
hclust_ward = hierarchy.ward( X )
```

In [48]:

```
fig = plt.figure(figsize=(16,8))
dn = hierarchy.dendrogram( hclust_ward, no_labels=True )
plt.show()
```




Form this we can confirm that we have to take 5 clusters.

**Cut the tree**

In [49]:

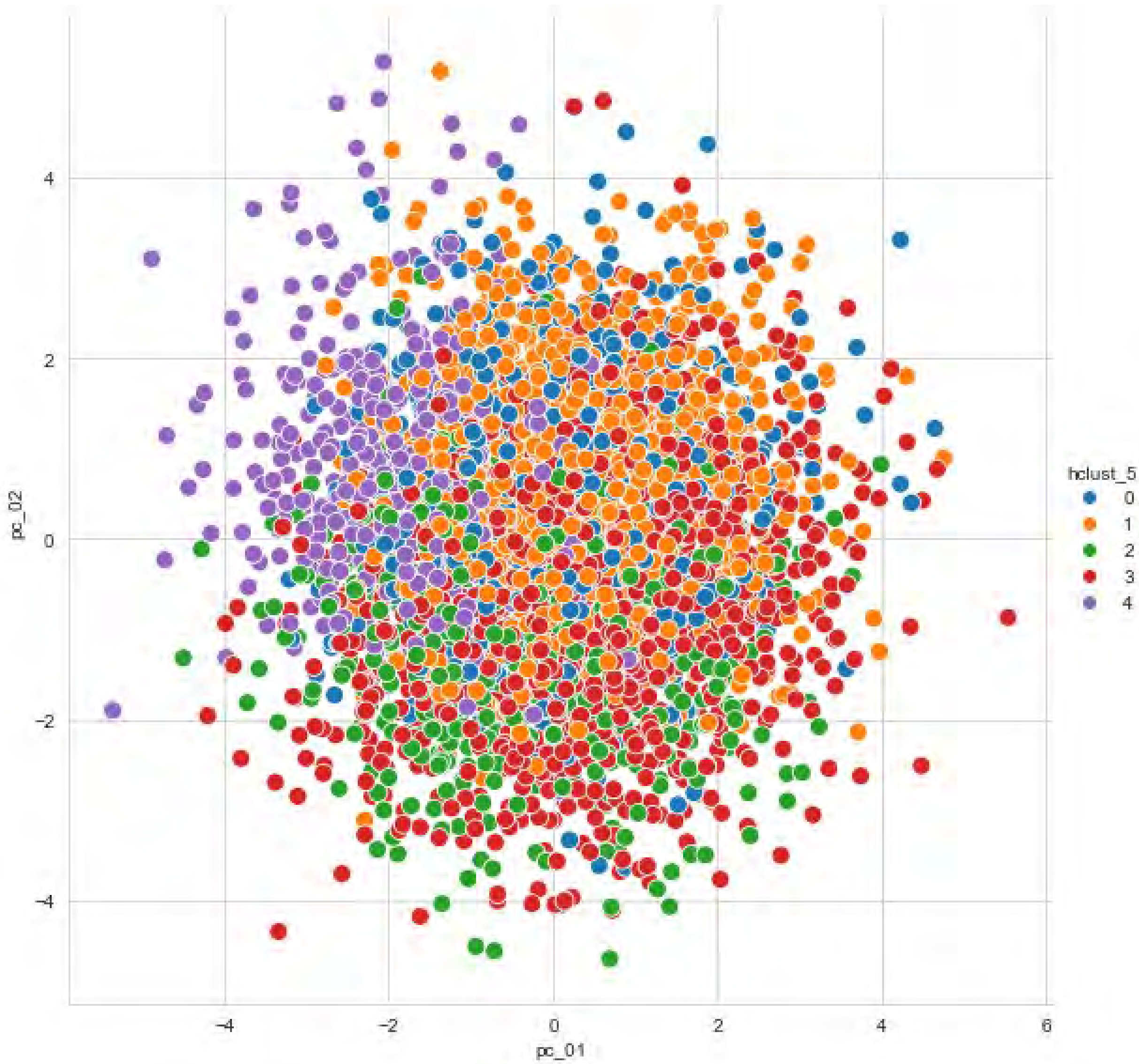
```
ward_cut_5 = hierarchy.cut_tree( hclust_ward, n_clusters=5 )
```

In [50]:

```
churn_pcs_df['hclust_5'] = pd.Series( ward_cut_5.ravel(), index=churn_pcs_df.index)

churn_pcs_df['hclust_5'] = churn_pcs_df.hclust_5.astype('category')
```

In [51]:

```
sns.relplot(data = churn_pcs_df, x='pc_01', y='pc_02', hue='hclust_5', s=100, height=8)

plt.show()
```




# Elastic Net

In [52]:

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
```

In [53]:

```
from sklearn.pipeline import Pipeline
```

In [54]:

```
from sklearn.model_selection import cross_val_score
```

In [55]:

```
my_cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=101)
```

Case : When Treating `X19` as categorical and `X17` and `X06` as continuous

In [56]:

```
df_6 = pd.read_csv('ppg_churn.csv')
```

In [57]:

```
df_6['lump_x19'] = np.where( df_6.X19 > 3, 'Other', df_6.X19.astype('str'))
```

In [58]:

```
df_6['X06_trans'] = np.log(df_6.X06 + 0.001)
```

In [59]:

```
df_6 = df_6.drop(columns=['X06', 'X19'])
```

In [60]:

```
xinputs_6 = df_6.select_dtypes('number').copy()
youtput_6 = df_6.loc[:, ['churn']].copy()

X_train_6 = xinputs_6.to_numpy()
y_train_6 = youtput_6.churn.to_numpy().ravel()
```

In [61]:

```
enet_default_6 = LogisticRegression(penalty='elasticnet', solver='saga', random_state=101,
max_iter=10001,
                                    C=1.0, l1_ratio=0.5)

default_enet_wflow_6 = Pipeline( steps=[('std_inputs', StandardScaler()),
                                        ('enet', enet_default_6)] )

enet_default_cv_6 = cross_val_score(default_enet_wflow_6, X_train_6, y_train_6, cv=my_cv)

enet_default_cv_6

enet_default_cv_6.mean()
```

Out[61]:

```
0.8661333333333333
```

In [62]:

```
from sklearn.model_selection import GridSearchCV
```

In [ ]:

From the Eastic net results we get the beset result when we treat the `X19` variable as categorical and `X17` and `X06` as continuous

# Models Interpretation

In [63]:

```
df_mod = df.copy()
```

In [64]:

```
df_mod['churn'] = np.where(df.churn=='yes', 1, 0)
```

In [65]:

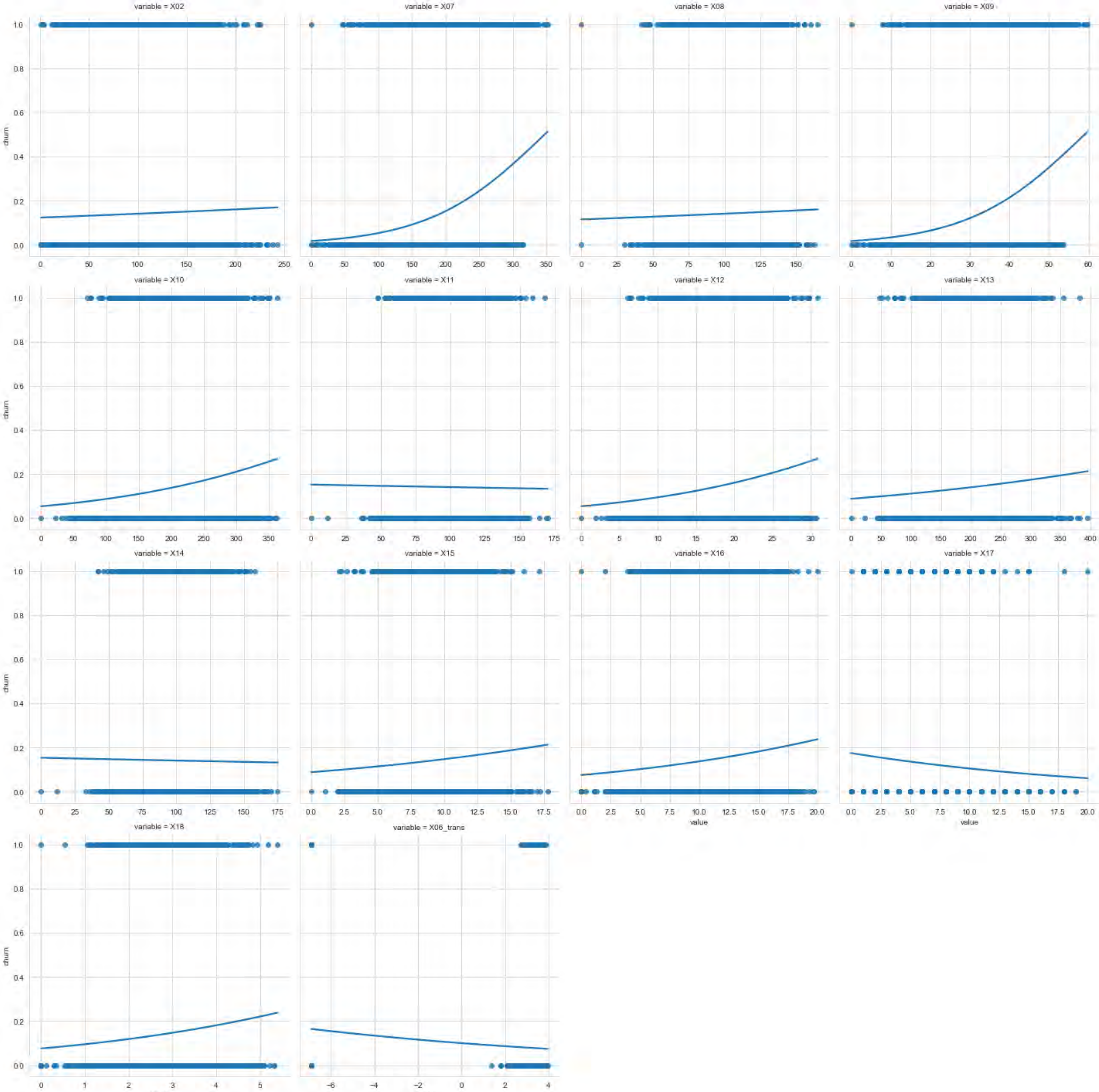
```
num_inputs_1 = df.select_dtypes('number').copy().columns.to_list()
```

In [66]:

```
lf_num_1 = df_mod.melt(id_vars=['churn','X03','X04','X05'], value_vars = num_inputs_1,
ignore_index=True)
```

In [67]:

```
sns.lmplot(data = lf_num_1, x='value', y='churn', col='variable', logistic=True,
col_wrap=4, ci=None,
           facet_kws={'sharey':True, 'sharex':False})

plt.show()
```

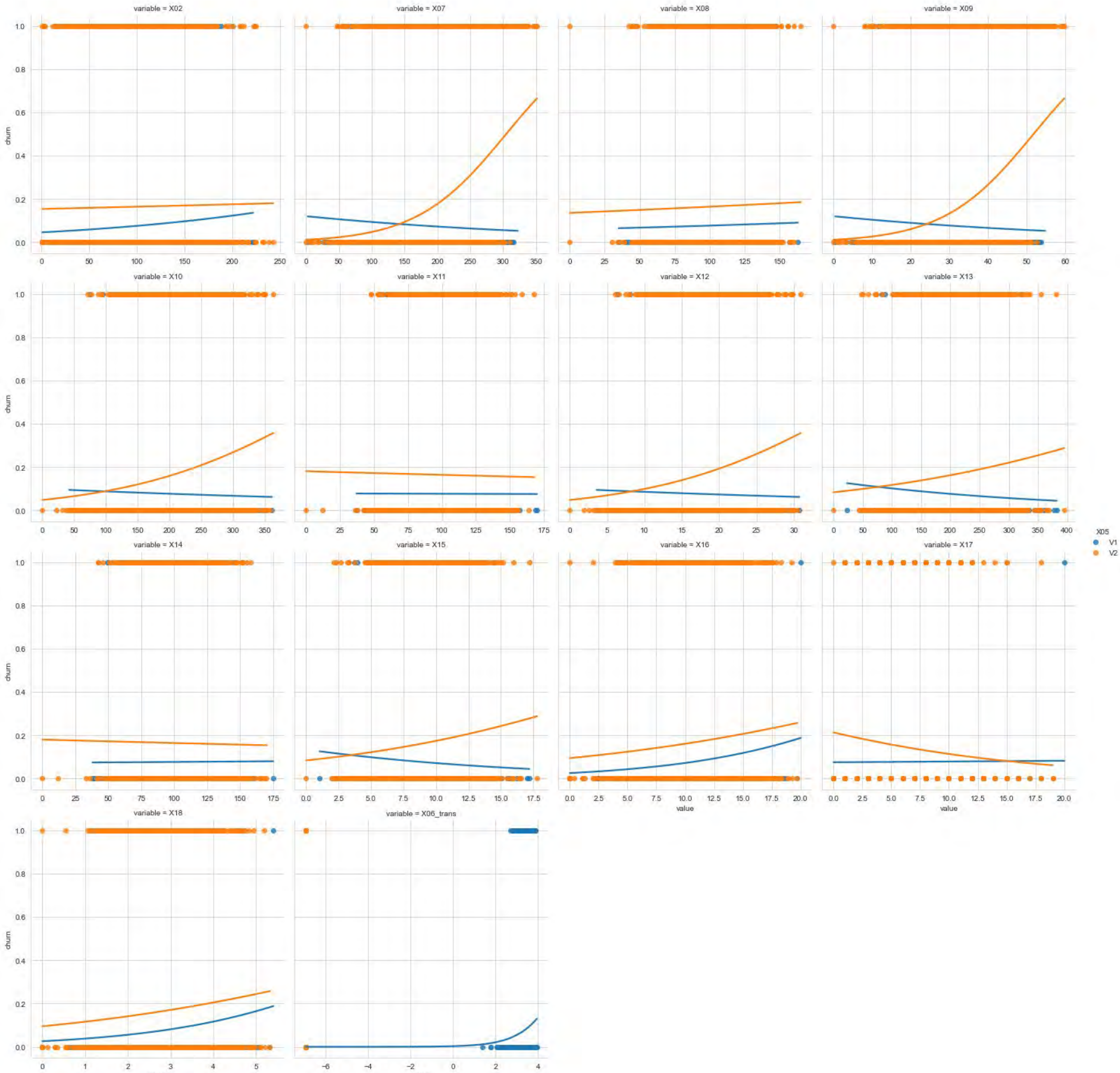



In [68]:

```
sns.lmplot(data = lf_num_1, x='value', y='churn', col='variable', hue='X04',
logistic=True, col_wrap=4, ci=None,
            facet_kws={'sharey':True, 'sharex':False})

plt.show()
```


In [69]:

```
sns.lmplot(data = lf_num_1, x='value', y='churn', col='variable', hue='X05',
logistic=True, col_wrap=4, ci=None,
            facet_kws={'sharey':True, 'sharex':False})

plt.show()
```




In [70]:

```
df_main = df_stand.copy()
```

In [71]:

```
df_main['churn'] = df_mod.churn.copy()
df_main['state'] = df.state.copy()
df_main['X03'] = df.X03.copy()
df_main['X04'] = df.X04.copy()
df_main['X05'] = df.X05.copy()
df_main['lump_X19'] = df.lump_x19.copy()
```

In [72]:

```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   X02        5000 non-null   float64
 1   X07        5000 non-null   float64
 2   X08        5000 non-null   float64
 3   X09        5000 non-null   float64
 4   X10        5000 non-null   float64
 5   X11        5000 non-null   float64
 6   X12        5000 non-null   float64
 7   X13        5000 non-null   float64
 8   X14        5000 non-null   float64
 9   X15        5000 non-null   float64
 10  X16        5000 non-null   float64
 11  X17        5000 non-null   float64
 12  X18        5000 non-null   float64
 13  X06_trans  5000 non-null   float64
 14  churn      5000 non-null   int32  
 15  state      5000 non-null   object 
 16  X03        5000 non-null   object 
 17  X04        5000 non-null   object 
 18  X05        5000 non-null   object 
 19  lump_X19   5000 non-null   object 
dtypes: float64(14), int32(1), object(5)
memory usage: 761.8+ KB
```

In [73]:

```
import statsmodels.formula.api as smf
```

In [74]:

```
formula_1 = 'churn ~ X02 + X07 + X08 + X09 + X10 + X11 + X12 + X13 + X14 + X15 + X16 + X17
+ X18 + X06_trans'

formula_2 = 'churn ~ (X02 + X07 + X08 + X09 + X10 + X11 + X12 + X13 + X14 + X15 + X16 +
X17 + X18 + X06_trans) ** 2'

formula_3 = 'churn ~ state + X03 + X04 + X05 + lump_X19'

formula_4 = 'churn ~ X02 + X07 + X08 + X09 + X10 + X11 + X12 + X13 + X14 + X15 + X16 + X17
+ X18 + X06_trans + state + X03 + X04 + X05 + lump_X19'

formula_5 = 'churn ~ (X02 + X07 + X08 + X09 + X10 + X11 + X12 + X13 + X14 + X15 + X16 +
X17 + X18 + X06_trans) * (state + X03 + X04 + X05 + lump_X19)'

formula_6 = 'churn ~ np.power(X02, 2) + np.power(X07, 2) + np.power(X08, 2) +
np.power(X09, 2) + np.power(X10, 2) + np.power(X11, 2) + np.power(X12, 2) + np.power(X13,
2) + np.power(X14, 2) + np.power(X15, 2) + np.power(X16, 2) + np.power(X17, 2) +
np.power(X18, 2) + np.power(X06_trans, 2) '

formula_7 = 'churn ~ np.power(X02, 4) + np.power(X07, 4) + np.power(X08, 4) +
```

```
np.power(X09, 4) + np.power(X10, 4) + np.power(X11, 4) + np.power(X12, 4) + np.power(X13,
4) + np.power(X14, 4) + np.power(X15, 4) + np.power(X16, 4) + np.power(X17, 4) +
np.power(X18, 4) + np.power(X06_trans, 4)'

formula_8 = 'churn ~ np.power(X02, 2) + np.power(X07, 2) + np.power(X08, 2) +
np.power(X09, 2) + np.power(X10, 2) + np.power(X11, 2) + np.power(X12, 2) + np.power(X13,
2) + np.power(X14, 2) + np.power(X15, 2) + np.power(X16, 2) + np.power(X17, 2) +
np.power(X18, 2) + np.power(X06_trans, 2) + (X02 + X07 + X08 + X09 + X10 + X11 + X12 + X13
+ X14 + X15 + X16 + X17 + X18 + X06_trans ) * (state + X03 + X04 + X05 + lump_X19)'
```

In [75]:

```
def my_coefplot(model_object, figsize_use=(10,5)):
    fig, ax = plt.subplots(figsize=figsize_use)

    ax.errorbar(y = model_object.params.index,
                x = model_object.params,
                xerr = 2 * model_object.bse,
                fmt='o', color='black', ecolor='black',
                elinewidth=3, ms=10)

    ax.axvline(x=0, linestyle='--', linewidth=5, color='grey')

    ax.set_xlabel('coefficient value')

    plt.show()
```

In [76]:

```
fit_01 = smf.logit(formula = formula_1, data = df_main).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.368005
         Iterations 9
```

In [77]:

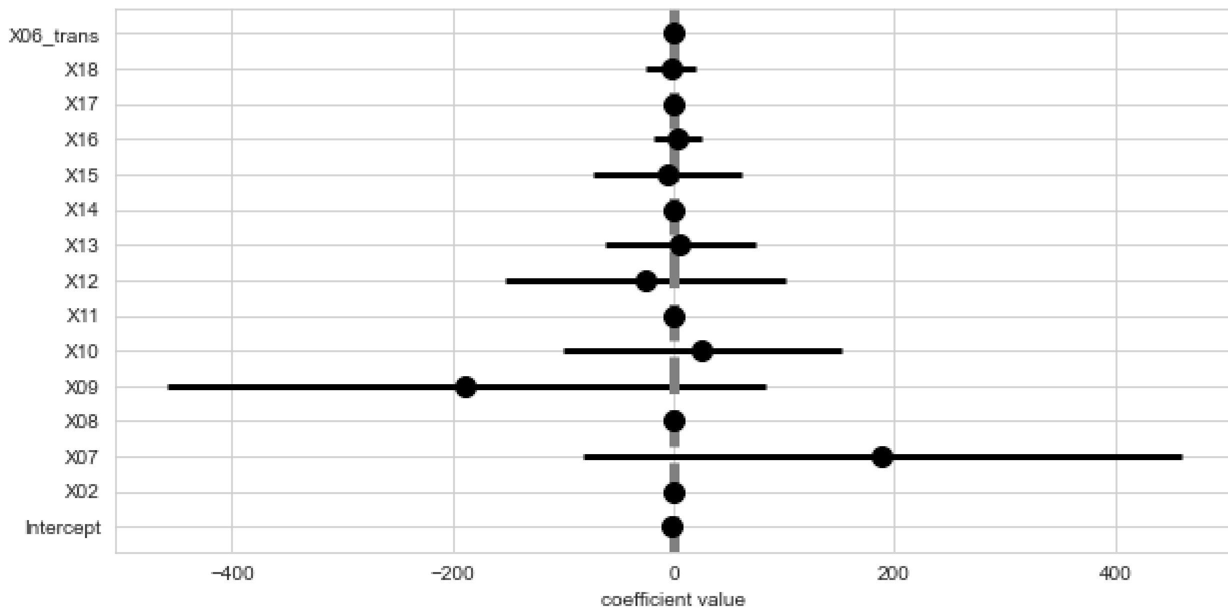
```
print( fit_01.summary() )
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  churn   No. Observations:                 5000
Model:                          Logit   Df Residuals:                     4985
Method:                           MLE   Df Model:                           14
Date:                Tue, 14 Dec 2021   Pseudo R-squ.:                 0.09691
Time:                        19:42:22   Log-Likelihood:                -1840.0
converged:                       True   LL-Null:                       -2037.5
Covariance Type:            nonrobust   LLR p-value:                 1.493e-75
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -2.0569      0.050    -41.536      0.000      -2.154      -1.960
X02            0.0659      0.042      1.559      0.119      -0.017       0.149
X07          188.8316    135.603      1.393      0.164     -76.946     454.609
X08            0.0413      0.042      0.976      0.329      -0.042       0.124
X09         -188.1769    135.603     -1.388      0.165    -453.954      77.600
X10           25.8932     63.370      0.409      0.683     -98.311     150.097
X11           -0.0248      0.043     -0.582      0.561      -0.108       0.059
```

```
X12          -25.5904     63.370     -0.404      0.686   -149.793      98.613
X13            5.8066     33.721      0.172      0.863    -60.286      71.899
X14           -0.0285      0.042     -0.675      0.500     -0.111       0.054
X15           -5.6627     33.721     -0.168      0.867    -71.754      60.429
X16            3.0517     11.083      0.275      0.783    -18.671      24.774
X17           -0.1633      0.046     -3.555      0.000     -0.253      -0.073
X18           -2.8320     11.083     -0.256      0.798    -24.555      18.891
X06_trans     -0.4102      0.051     -8.063      0.000     -0.510      -0.310
==============================================================================
```

In [78]:

```
my_coefplot(fit_01)
```




In [79]:

```
fit_03 = smf.logit(formula = formula_3, data = df_main).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.328347
         Iterations 7
```

In [80]:

```
print( fit_03.summary() )
```

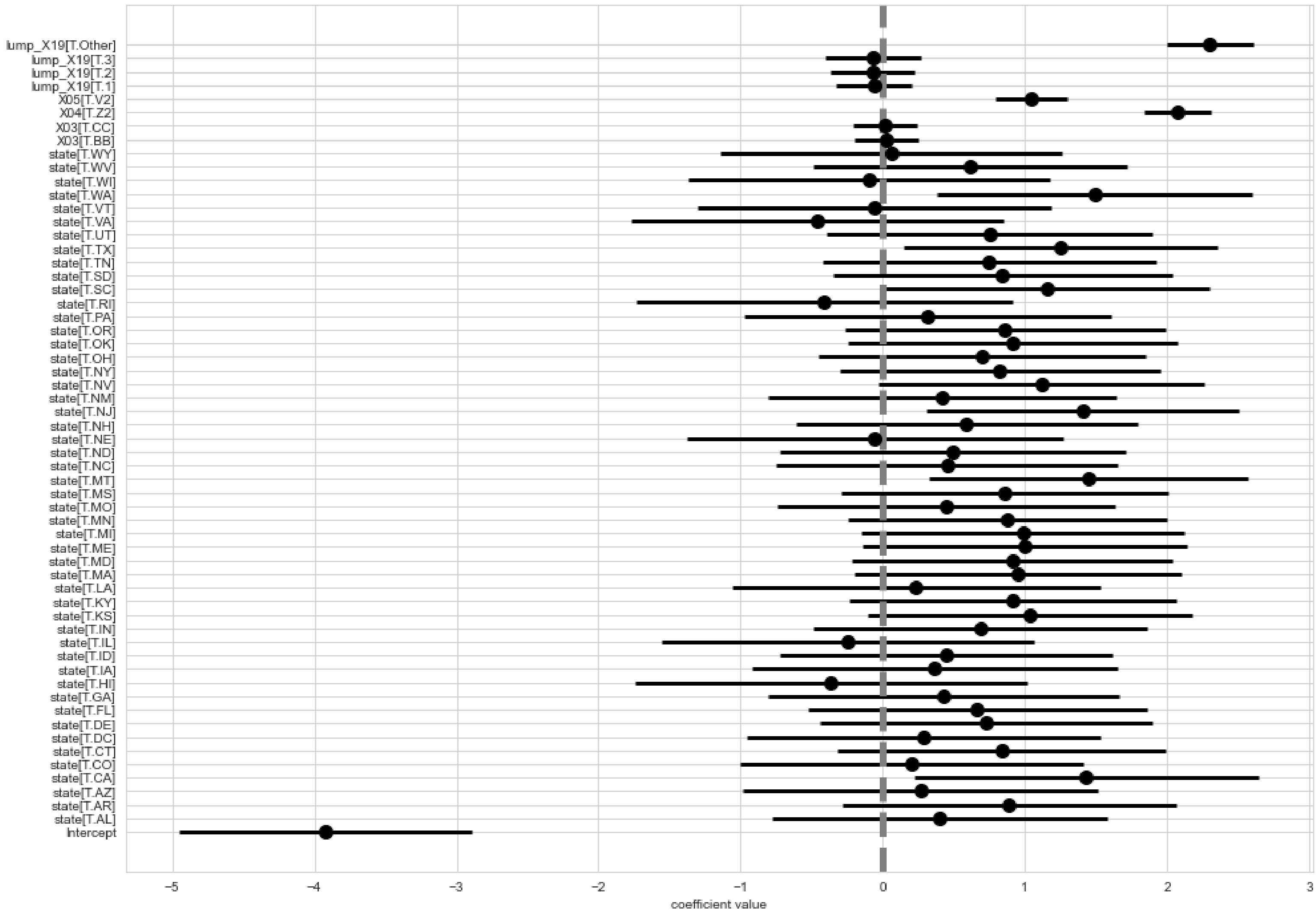
```
                           Logit Regression Results                           
==============================================================================
Dep. Variable:                  churn   No. Observations:                 5000
Model:                          Logit   Df Residuals:                     4941
Method:                           MLE   Df Model:                           58
Date:                Tue, 14 Dec 2021   Pseudo R-squ.:                  0.1942
Time:                        19:42:23   Log-Likelihood:                -1641.7
converged:                       True   LL-Null:                       -2037.5
Covariance Type:            nonrobust   LLR p-value:                2.535e-129
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
Intercept        -3.9189      0.514     -7.618      0.000      -4.927      -2.911
state[T.AL]       0.4004      0.588      0.681      0.496      -0.751       1.552
```

| | | | | | | |
|---|---|---|---|---|---|---|
| state[T.AR] | 0.8905 | 0.588 | 1.514 | 0.130 | -0.262 | 2.043 |
| state[T.AZ] | 0.2674 | 0.624 | 0.428 | 0.668 | -0.956 | 1.491 |
| state[T.CA] | 1.4322 | 0.606 | 2.364 | 0.018 | 0.245 | 2.620 |
| state[T.CO] | 0.2016 | 0.603 | 0.334 | 0.738 | -0.981 | 1.384 |
| state[T.CT] | 0.8350 | 0.577 | 1.448 | 0.148 | -0.295 | 1.965 |
| state[T.DC] | 0.2871 | 0.620 | 0.463 | 0.643 | -0.929 | 1.503 |
| state[T.DE] | 0.7295 | 0.584 | 1.250 | 0.211 | -0.415 | 1.874 |
| state[T.FL] | 0.6655 | 0.595 | 1.119 | 0.263 | -0.500 | 1.831 |
| state[T.GA] | 0.4288 | 0.618 | 0.694 | 0.488 | -0.783 | 1.641 |
| state[T.HI] | -0.3643 | 0.691 | -0.527 | 0.598 | -1.718 | 0.989 |
| state[T.IA] | 0.3670 | 0.641 | 0.572 | 0.567 | -0.890 | 1.624 |
| state[T.ID] | 0.4467 | 0.582 | 0.767 | 0.443 | -0.695 | 1.588 |
| state[T.IL] | -0.2440 | 0.653 | -0.373 | 0.709 | -1.524 | 1.036 |
| state[T.IN] | 0.6891 | 0.587 | 1.174 | 0.240 | -0.461 | 1.839 |
| state[T.KS] | 1.0360 | 0.572 | 1.811 | 0.070 | -0.085 | 2.157 |
| state[T.KY] | 0.9150 | 0.577 | 1.586 | 0.113 | -0.216 | 2.046 |
| state[T.LA] | 0.2357 | 0.648 | 0.364 | 0.716 | -1.034 | 1.506 |
| state[T.MA] | 0.9552 | 0.575 | 1.662 | 0.097 | -0.171 | 2.082 |
| state[T.MD] | 0.9109 | 0.565 | 1.612 | 0.107 | -0.197 | 2.018 |
| state[T.ME] | 0.9973 | 0.570 | 1.749 | 0.080 | -0.120 | 2.115 |
| state[T.MI] | 0.9857 | 0.568 | 1.735 | 0.083 | -0.128 | 2.099 |
| state[T.MN] | 0.8779 | 0.561 | 1.565 | 0.118 | -0.221 | 1.977 |
| state[T.MO] | 0.4468 | 0.595 | 0.750 | 0.453 | -0.720 | 1.614 |
| state[T.MS] | 0.8592 | 0.574 | 1.496 | 0.135 | -0.266 | 1.985 |
| state[T.MT] | 1.4484 | 0.560 | 2.588 | 0.010 | 0.352 | 2.545 |
| state[T.NC] | 0.4520 | 0.601 | 0.752 | 0.452 | -0.725 | 1.629 |
| state[T.ND] | 0.4932 | 0.610 | 0.809 | 0.419 | -0.702 | 1.688 |
| state[T.NE] | -0.0566 | 0.663 | -0.085 | 0.932 | -1.355 | 1.242 |
| state[T.NH] | 0.5911 | 0.601 | 0.984 | 0.325 | -0.587 | 1.769 |
| state[T.NJ] | 1.4058 | 0.549 | 2.561 | 0.010 | 0.330 | 2.482 |
| state[T.NM] | 0.4205 | 0.613 | 0.686 | 0.492 | -0.780 | 1.621 |
| state[T.NV] | 1.1188 | 0.572 | 1.955 | 0.051 | -0.003 | 2.241 |
| state[T.NY] | 0.8248 | 0.564 | 1.462 | 0.144 | -0.281 | 1.931 |
| state[T.OH] | 0.6950 | 0.575 | 1.209 | 0.227 | -0.432 | 1.822 |
| state[T.OK] | 0.9122 | 0.581 | 1.570 | 0.116 | -0.226 | 2.051 |
| state[T.OR] | 0.8625 | 0.565 | 1.526 | 0.127 | -0.245 | 1.971 |
| state[T.PA] | 0.3147 | 0.645 | 0.488 | 0.625 | -0.949 | 1.578 |
| state[T.RI] | -0.4112 | 0.662 | -0.621 | 0.534 | -1.709 | 0.886 |
| state[T.SC] | 1.1549 | 0.574 | 2.012 | 0.044 | 0.030 | 2.280 |
| state[T.SD] | 0.8428 | 0.596 | 1.415 | 0.157 | -0.325 | 2.011 |
| state[T.TN] | 0.7499 | 0.586 | 1.279 | 0.201 | -0.400 | 1.899 |
| state[T.TX] | 1.2548 | 0.552 | 2.273 | 0.023 | 0.173 | 2.337 |
| state[T.UT] | 0.7532 | 0.572 | 1.316 | 0.188 | -0.369 | 1.875 |
| state[T.VA] | -0.4624 | 0.656 | -0.705 | 0.481 | -1.747 | 0.823 |
| state[T.VT] | -0.0598 | 0.624 | -0.096 | 0.924 | -1.282 | 1.163 |
| state[T.WA] | 1.4919 | 0.554 | 2.693 | 0.007 | 0.406 | 2.578 |
| state[T.WI] | -0.0952 | 0.634 | -0.150 | 0.881 | -1.338 | 1.148 |
| state[T.WV] | 0.6185 | 0.552 | 1.120 | 0.263 | -0.464 | 1.701 |
| state[T.WY] | 0.0587 | 0.602 | 0.098 | 0.922 | -1.120 | 1.238 |
| X03[T.BB] | 0.0247 | 0.112 | 0.221 | 0.825 | -0.195 | 0.245 |
| X03[T.CC] | 0.0169 | 0.111 | 0.152 | 0.879 | -0.201 | 0.235 |
| X04[T.Z2] | 2.0748 | 0.118 | 17.598 | 0.000 | 1.844 | 2.306 |
| X05[T.V2] | 1.0460 | 0.125 | 8.354 | 0.000 | 0.801 | 1.291 |
| lump_X19[T.1] | -0.0611 | 0.133 | -0.460 | 0.645 | -0.321 | 0.199 |
| lump_X19[T.2] | -0.0709 | 0.146 | -0.485 | 0.628 | -0.357 | 0.216 |
| lump_X19[T.3] | -0.0706 | 0.169 | -0.417 | 0.676 | -0.402 | 0.261 |
| lump_X19[T.Other] | 2.3020 | 0.153 | 15.035 | 0.000 | 2.002 | 2.602 |

```
==============================================================================
```

In [81]:

```
my_coefplot(fit_03, figsize_use=(16, 12))
```




In [82]:

```
fit_04 = smf.logit(formula = formula_4, data = df_main).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.290285
         Iterations 10
```

In [83]:

```
print( fit_04.summary() )
```

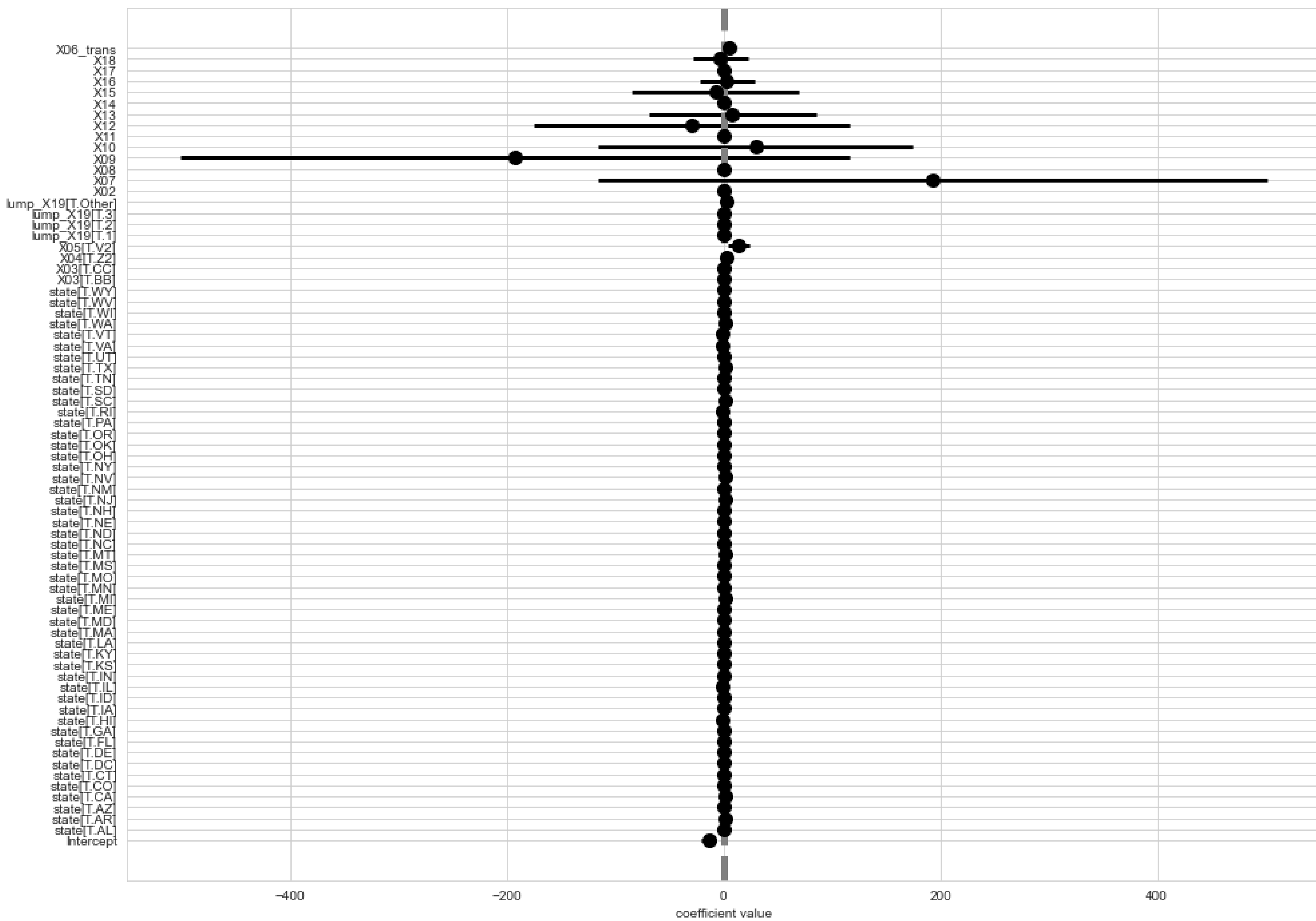
```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  churn   No. Observations:                 5000
Model:                          Logit   Df Residuals:                     4927
Method:                           MLE   Df Model:                           72
Date:                Tue, 14 Dec 2021   Pseudo R-squ.:                  0.2876
Time:                        19:42:24   Log-Likelihood:                -1451.4
converged:                       True   LL-Null:                       -2037.5
Covariance Type:            nonrobust   LLR p-value:                2.337e-198
```

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| Intercept | -13.9097 | 3.575 | -3.891 | 0.000 | -20.917 | -6.903 |
| state[T.AL] | 0.2382 | 0.605 | 0.394 | 0.694 | -0.948 | 1.424 |
| state[T.AR] | 0.9523 | 0.611 | 1.558 | 0.119 | -0.246 | 2.150 |
| state[T.AZ] | 0.1146 | 0.650 | 0.176 | 0.860 | -1.159 | 1.388 |
| state[T.CA] | 1.5372 | 0.636 | 2.415 | 0.016 | 0.290 | 2.785 |
| state[T.CO] | -0.0836 | 0.643 | -0.130 | 0.897 | -1.344 | 1.177 |
| state[T.CT] | 0.6306 | 0.597 | 1.057 | 0.291 | -0.539 | 1.800 |
| state[T.DC] | 0.3102 | 0.641 | 0.484 | 0.628 | -0.946 | 1.566 |
| state[T.DE] | 0.5198 | 0.601 | 0.865 | 0.387 | -0.659 | 1.698 |
| state[T.FL] | 0.6187 | 0.610 | 1.014 | 0.311 | -0.577 | 1.815 |
| state[T.GA] | 0.1999 | 0.652 | 0.307 | 0.759 | -1.078 | 1.478 |
| state[T.HI] | -0.5561 | 0.714 | -0.779 | 0.436 | -1.955 | 0.843 |
| state[T.IA] | 0.3028 | 0.678 | 0.447 | 0.655 | -1.025 | 1.631 |
| state[T.ID] | 0.4042 | 0.600 | 0.673 | 0.501 | -0.773 | 1.581 |
| state[T.IL] | -0.4258 | 0.693 | -0.614 | 0.539 | -1.785 | 0.933 |
| state[T.IN] | 0.3228 | 0.618 | 0.522 | 0.602 | -0.889 | 1.535 |
| state[T.KS] | 0.7037 | 0.595 | 1.183 | 0.237 | -0.462 | 1.869 |
| state[T.KY] | 0.8091 | 0.593 | 1.363 | 0.173 | -0.354 | 1.972 |
| state[T.LA] | 0.4107 | 0.667 | 0.615 | 0.538 | -0.897 | 1.719 |
| state[T.MA] | 0.8718 | 0.594 | 1.468 | 0.142 | -0.292 | 2.036 |
| state[T.MD] | 0.6656 | 0.589 | 1.130 | 0.258 | -0.489 | 1.820 |
| state[T.ME] | 0.8876 | 0.589 | 1.507 | 0.132 | -0.267 | 2.042 |
| state[T.MI] | 1.0276 | 0.586 | 1.753 | 0.080 | -0.121 | 2.176 |
| state[T.MN] | 0.8268 | 0.576 | 1.436 | 0.151 | -0.302 | 1.955 |
| state[T.MO] | 0.3909 | 0.621 | 0.629 | 0.529 | -0.827 | 1.609 |
| state[T.MS] | 0.8050 | 0.593 | 1.357 | 0.175 | -0.358 | 1.968 |
| state[T.MT] | 1.4991 | 0.576 | 2.601 | 0.009 | 0.369 | 2.629 |
| state[T.NC] | 0.2659 | 0.627 | 0.424 | 0.671 | -0.963 | 1.495 |
| state[T.ND] | 0.2317 | 0.639 | 0.363 | 0.717 | -1.020 | 1.484 |
| state[T.NE] | -0.1096 | 0.681 | -0.161 | 0.872 | -1.445 | 1.226 |
| state[T.NH] | 0.5075 | 0.625 | 0.812 | 0.417 | -0.717 | 1.732 |
| state[T.NJ] | 1.1691 | 0.568 | 2.058 | 0.040 | 0.056 | 2.282 |
| state[T.NM] | 0.2894 | 0.634 | 0.456 | 0.648 | -0.953 | 1.532 |
| state[T.NV] | 0.9231 | 0.594 | 1.555 | 0.120 | -0.240 | 2.087 |
| state[T.NY] | 0.8819 | 0.583 | 1.513 | 0.130 | -0.261 | 2.025 |
| state[T.OH] | 0.5097 | 0.599 | 0.852 | 0.394 | -0.663 | 1.683 |
| state[T.OK] | 0.7195 | 0.603 | 1.192 | 0.233 | -0.463 | 1.902 |
| state[T.OR] | 0.8234 | 0.585 | 1.408 | 0.159 | -0.323 | 1.970 |
| state[T.PA] | 0.1136 | 0.671 | 0.169 | 0.865 | -1.201 | 1.428 |
| state[T.RI] | -0.7577 | 0.700 | -1.082 | 0.279 | -2.131 | 0.615 |
| state[T.SC] | 1.0246 | 0.602 | 1.702 | 0.089 | -0.155 | 2.204 |
| state[T.SD] | 0.6562 | 0.620 | 1.059 | 0.290 | -0.558 | 1.870 |
| state[T.TN] | 0.7045 | 0.605 | 1.164 | 0.245 | -0.482 | 1.891 |
| state[T.TX] | 1.1025 | 0.571 | 1.931 | 0.054 | -0.017 | 2.222 |
| state[T.UT] | 0.6559 | 0.594 | 1.104 | 0.270 | -0.509 | 1.821 |
| state[T.VA] | -0.7192 | 0.682 | -1.054 | 0.292 | -2.056 | 0.618 |
| state[T.VT] | -0.4039 | 0.653 | -0.619 | 0.536 | -1.684 | 0.876 |
| state[T.WA] | 1.3605 | 0.577 | 2.359 | 0.018 | 0.230 | 2.491 |
| state[T.WI] | -0.1493 | 0.661 | -0.226 | 0.821 | -1.444 | 1.145 |
| state[T.WV] | 0.5356 | 0.574 | 0.933 | 0.351 | -0.590 | 1.661 |
| state[T.WY] | -0.2739 | 0.633 | -0.433 | 0.665 | -1.514 | 0.967 |
| X03[T.BB] | 0.0641 | 0.120 | 0.536 | 0.592 | -0.170 | 0.298 |
| X03[T.CC] | 0.1083 | 0.118 | 0.915 | 0.360 | -0.124 | 0.340 |
| X04[T.Z2] | 2.2827 | 0.130 | 17.598 | 0.000 | 2.028 | 2.537 |

```
X05[T.V2]              14.1432      4.798      2.948      0.003      4.739     23.548
lump_X19[T.1]           0.0074      0.142      0.052      0.959     -0.271      0.285
lump_X19[T.2]           0.0599      0.155      0.387      0.699     -0.244      0.363
lump_X19[T.3]          -0.0372      0.179     -0.208      0.835     -0.387      0.313
lump_X19[T.Other]       2.7392      0.169     16.233      0.000      2.408      3.070
X02                     0.0714      0.048      1.483      0.138     -0.023      0.166
X07                   192.8078    154.338      1.249      0.212   -109.690    495.306
X08                     0.0447      0.048      0.929      0.353     -0.050      0.139
X09                  -192.0175    154.337     -1.244      0.213   -494.513    110.478
X10                    29.3703     72.764      0.404      0.686   -113.244    171.985
X11                    -0.0418      0.049     -0.850      0.395     -0.138      0.055
X12                   -28.9727     72.764     -0.398      0.691   -171.587    113.642
X13                     7.9402     38.536      0.206      0.837    -67.588     83.469
X14                    -0.0317      0.048     -0.654      0.513     -0.127      0.063
X15                    -7.6965     38.535     -0.200      0.842    -83.224     67.831
X16                     3.2364     12.722      0.254      0.799    -21.698     28.171
X17                    -0.1743      0.052     -3.336      0.001     -0.277     -0.072
X18                    -2.9848     12.722     -0.235      0.815    -27.919     21.949
X06_trans               5.6913      2.101      2.708      0.007      1.573      9.810
======================================================================================
```

In [84]:

```
my_coefplot(fit_04, figsize_use=(16, 12))
```




In [85]:

```
fit_06 = smf.logit(formula = formula_6, data = df_main).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.375739
         Iterations 8
```

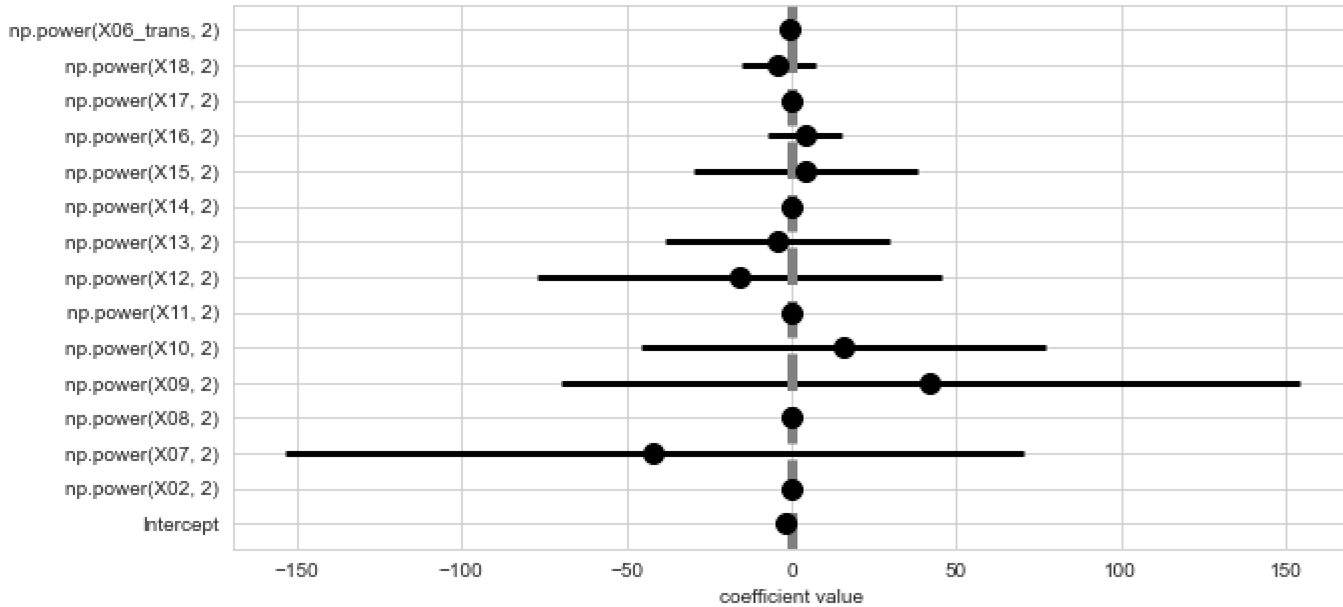
In [86]:

```
print( fit_06.summary() )
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  churn   No. Observations:                 5000
Model:                          Logit   Df Residuals:                     4985
Method:                           MLE   Df Model:                           14
Date:                Tue, 14 Dec 2021   Pseudo R-squ.:                 0.07793
Time:                        19:42:25   Log-Likelihood:                -1878.7
converged:                       True   LL-Null:                       -2037.5
Covariance Type:            nonrobust   LLR p-value:                 2.535e-59
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                 -2.0577      0.101    -20.309      0.000      -2.256      -1.859
np.power(X02, 2)          -0.0062      0.031     -0.202      0.840      -0.066       0.054
np.power(X07, 2)         -41.6645     56.067     -0.743      0.457    -151.555      68.225
np.power(X08, 2)           0.0202      0.029      0.703      0.482      -0.036       0.077
np.power(X09, 2)          42.0464     56.066      0.750      0.453     -67.840     151.933
np.power(X10, 2)          15.9449     30.716      0.519      0.604     -44.258      76.148
np.power(X11, 2)          -0.0109      0.030     -0.367      0.714      -0.069       0.047
np.power(X12, 2)         -15.9052     30.717     -0.518      0.605     -76.110      44.300
np.power(X13, 2)          -4.4132     16.938     -0.261      0.794     -37.612      28.785
np.power(X14, 2)          -0.0166      0.029     -0.565      0.572      -0.074       0.041
np.power(X15, 2)           4.4089     16.938      0.260      0.795     -28.789      37.607
np.power(X16, 2)           4.0894      5.661      0.722      0.470      -7.006      15.185
np.power(X17, 2)           0.0185      0.017      1.095      0.273      -0.015       0.052
np.power(X18, 2)          -4.0634      5.661     -0.718      0.473     -15.158       7.031
np.power(X06_trans, 2)    -0.3413      0.047     -7.239      0.000      -0.434      -0.249
==========================================================================================
```

In [87]:

```
my_coefplot(fit_06)
```




In [88]:

```
fit_07 = smf.logit(formula = formula_7, data = df_main).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.386547
         Iterations 6
```

In [89]:

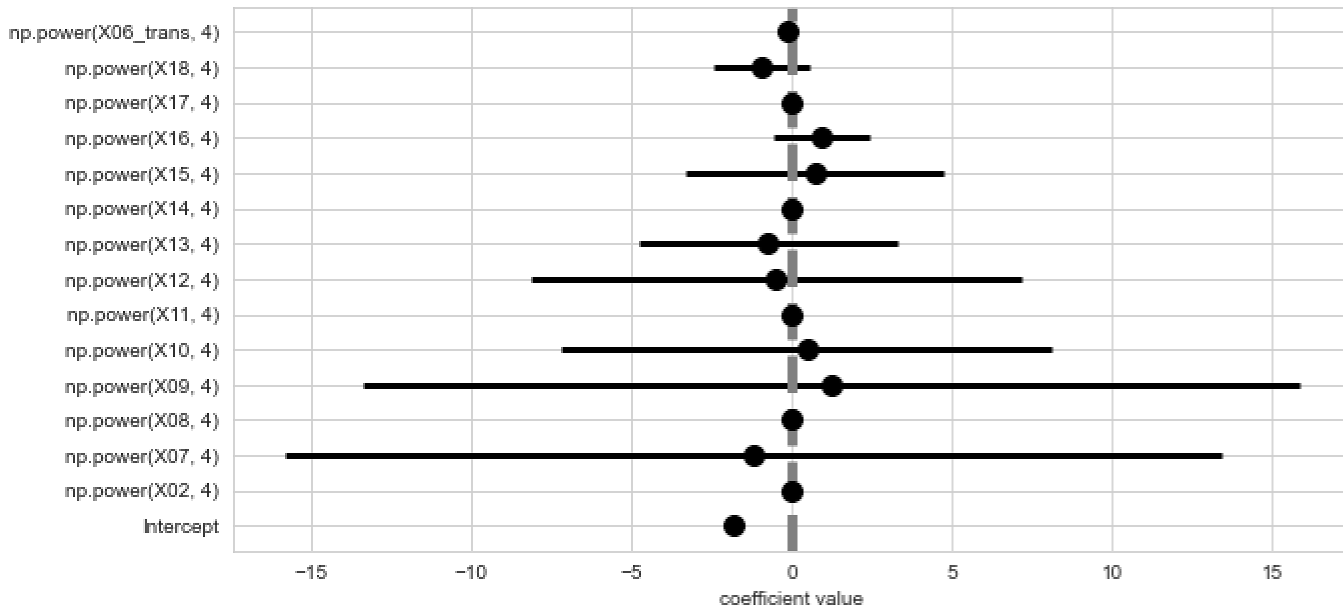
```
print( fit_07.summary() )
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  churn   No. Observations:                 5000
Model:                          Logit   Df Residuals:                     4985
Method:                           MLE   Df Model:                           14
Date:                Tue, 14 Dec 2021   Pseudo R-squ.:                 0.05141
Time:                        19:42:25   Log-Likelihood:                -1932.7
converged:                       True   LL-Null:                       -2037.5
Covariance Type:            nonrobust   LLR p-value:                 6.280e-37
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                 -1.7845      0.059    -30.478      0.000      -1.899      -1.670
np.power(X02, 4)           0.0006      0.005      0.133      0.894      -0.009       0.010
np.power(X07, 4)          -1.2066      7.303     -0.165      0.869     -15.521      13.108
np.power(X08, 4)          -0.0030      0.003     -1.028      0.304      -0.009       0.003
np.power(X09, 4)           1.2524      7.303      0.171      0.864     -13.061      15.566
np.power(X10, 4)           0.4911      3.820      0.129      0.898      -6.996       7.978
np.power(X11, 4)          -0.0035      0.004     -0.813      0.416      -0.012       0.005
np.power(X12, 4)          -0.4882      3.820     -0.128      0.898      -7.976       6.999
np.power(X13, 4)          -0.7202      2.026     -0.356      0.722      -4.690       3.250
np.power(X14, 4)          -0.0051      0.005     -1.079      0.281      -0.014       0.004
np.power(X15, 4)           0.7203      2.026      0.356      0.722      -3.250       4.690
np.power(X16, 4)           0.9411      0.747      1.260      0.208      -0.523       2.405
np.power(X17, 4)           0.0008      0.001      1.145      0.252      -0.001       0.002
np.power(X18, 4)          -0.9420      0.747     -1.261      0.207      -2.406       0.522
np.power(X06_trans, 4)    -0.1022      0.015     -6.981      0.000      -0.131      -0.073
```

==========================================================================================

In [90]:

```
my_coefplot(fit_07)
```




In [91]:

```
mod_list = [fit_01, fit_03, fit_04, fit_06, fit_07]
```

In [92]:

```
df_copy_b = df_main.copy()
```

In [93]:

```
for i, mod in enumerate(mod_list):
    df_copy_b['pred_probability_'+str(i+1).zfill(2)] = mod.predict(df_main)
```

In [94]:

```
for i in range(len(mod_list)):
    df_copy_b['pred_class_'+str(i+1).zfill(2)] = np.where(
df_copy_b['pred_probability_'+str(i+1).zfill(2)] > 0.5, 1, 0)
```

In [95]:

```
model_accuracy = []

for i in range(len(mod_list)):
    model_accuracy.append( df_copy_b.loc[ df_copy_b.churn ==
df_copy_b['pred_class_'+str(i+1).zfill(2)] ].shape[0] / df_copy_b.shape[0])
```

In [96]:

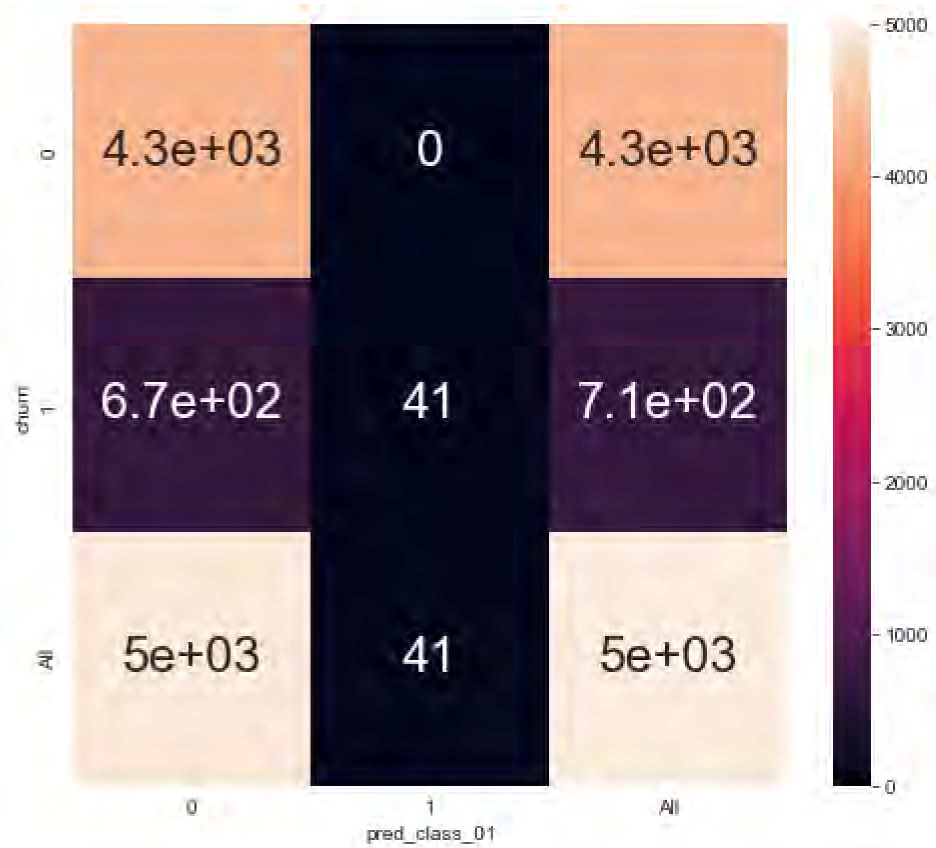
```
model_accuracy
```

Out[96]:

```
[0.8668, 0.8706, 0.8604, 0.8592, 0.859]
```

In [97]:

```
fig, ax = plt.subplots(figsize=(8,7))

sns.heatmap(pd.crosstab(df_copy_b.churn, df_copy_b.pred_class_01, margins=True),
            annot=True, annot_kws={'size': 25},
            ax=ax)

plt.show()
```




**ROC Curve**

In [98]:

```
from sklearn.metrics import roc_curve
```

In [99]:

```
def roc_values(mod_id, df_object):
    fpr, tpr, threshold = roc_curve(df_object.churn.to_numpy(),
df_object['pred_probability_'+str(mod_id+1).zfill(2)].to_numpy())
    res = pd.DataFrame({'fpr': fpr, 'tpr': tpr, 'threshold': threshold})
    res['model_name'] = 'fit_'+str(mod_id+1).zfill(2)

    return res
```

In [100]:

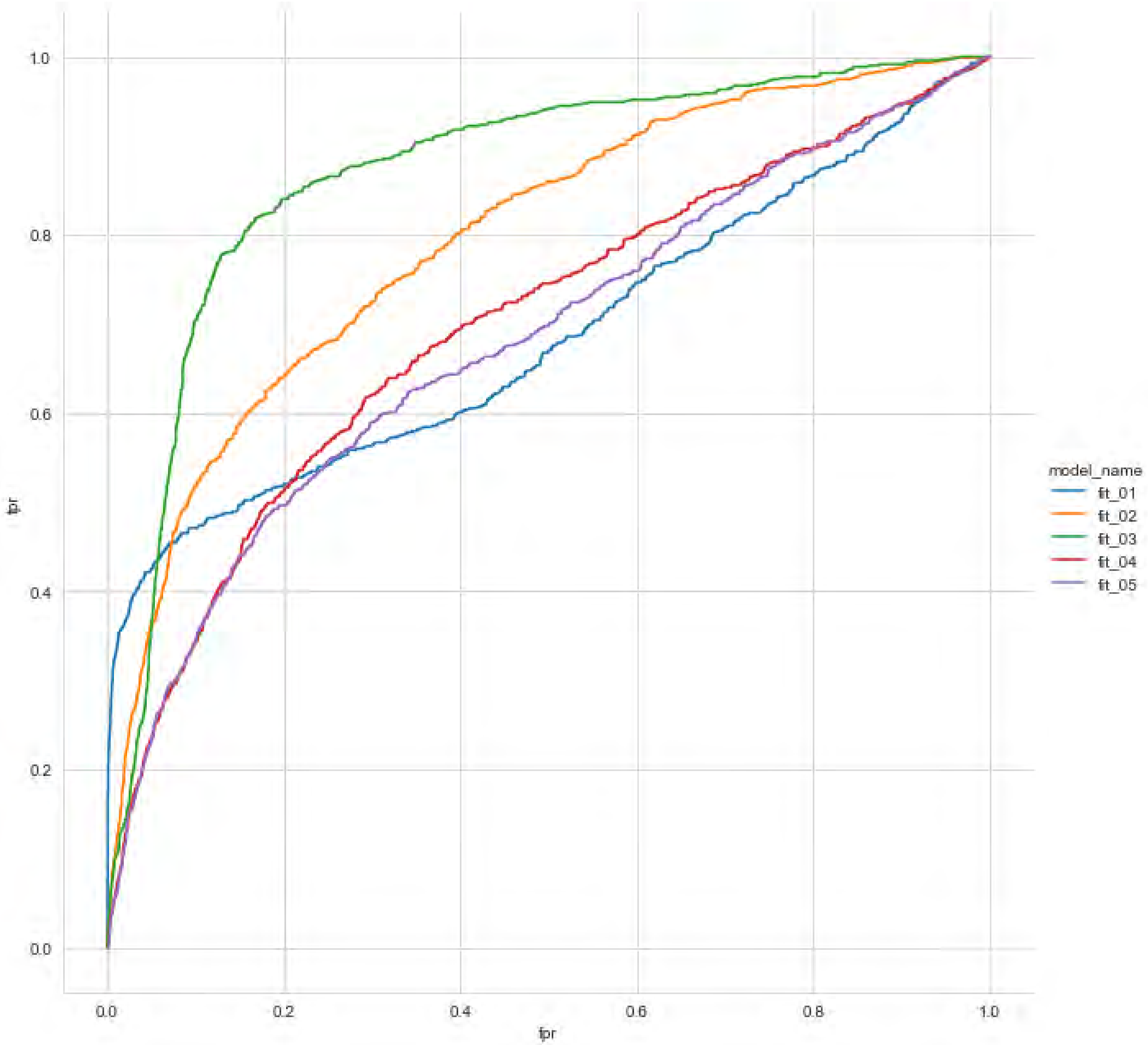
```
all_roc_curves = []

for i in range(len(mod_list)):
    all_roc_curves.append( roc_values(i, df_copy_b) )
```

In [101]:

```
all_roc_df = pd.concat(all_roc_curves).reset_index()
```

In [102]:

```
sns.relplot(data = all_roc_df, x='fpr', y='tpr', hue='model_name',
            estimator=None, units='model_name',
            kind='line',
            drawstyle='steps-post',
            height=9)

plt.show()
```




# Model Performance and Validation

In [103]:

```
from patsy import dmatrices
```

```
from sklearn.linear_model import LogisticRegression
```

In [104]:

```
from sklearn.model_selection import cross_val_score
```

In [105]:

```
y_07, X_07 = dmatrices(formula_7 + ' - 1', data = df_main)
```

In [106]:

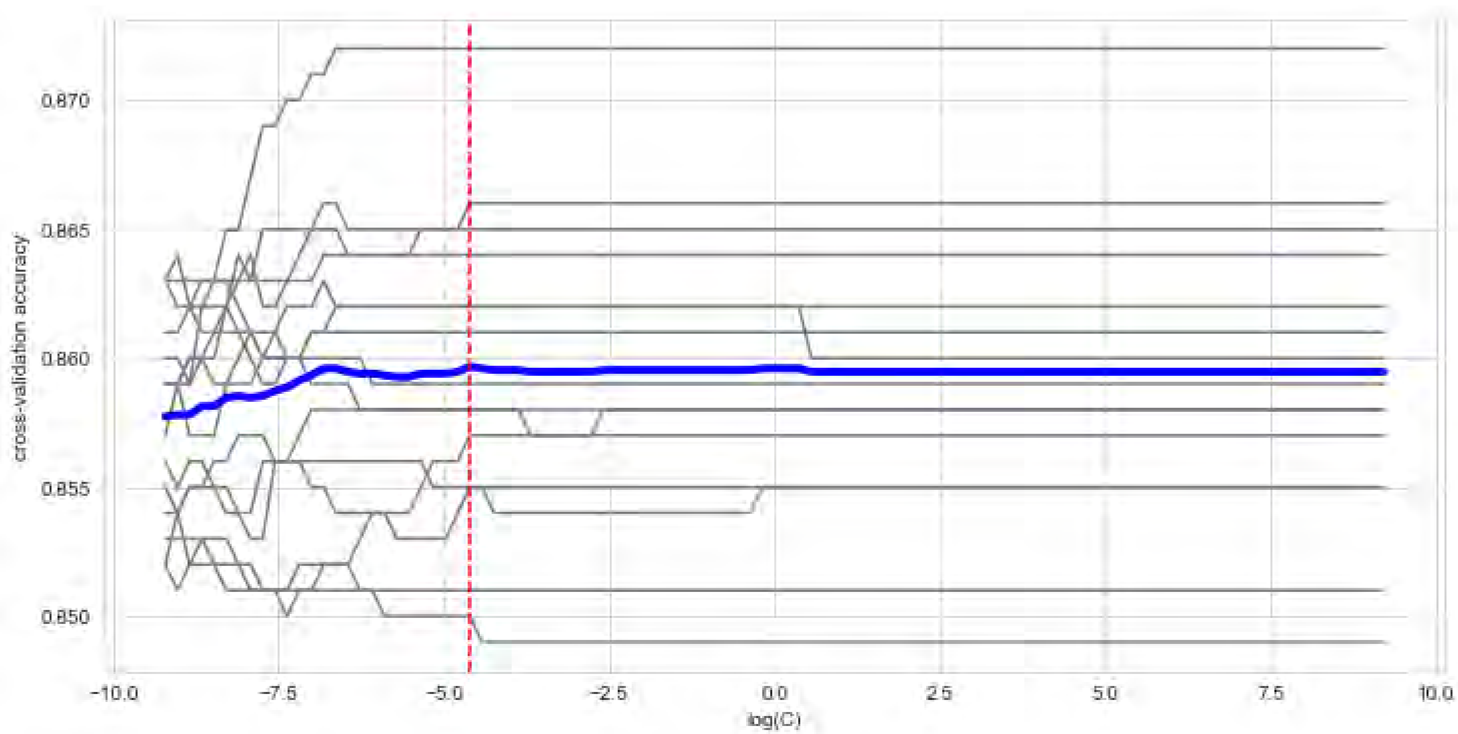
```
from sklearn.linear_model import LogisticRegressionCV
```

In [107]:

```
ridge_tune_results = LogisticRegressionCV(penalty='l2', Cs=101, cv=my_cv, solver='lbfgs',
max_iter=5001, fit_intercept=False).\
fit(X_07, y_07.ravel())
```

In [108]:

```
fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(np.log(ridge_tune_results.Cs_), ridge_tune_results.scores_[1.0].T, color='grey')
ax.plot(np.log(ridge_tune_results.Cs_), ridge_tune_results.scores_[1.0].mean(axis=0),
color='blue', linewidth=4)
ax.axvline(x=np.log(ridge_tune_results.C_), color='red', linestyle='dashed')
ax.set_xlabel('log(C)')
ax.set_ylabel("cross-validation accuracy")

plt.show()
```



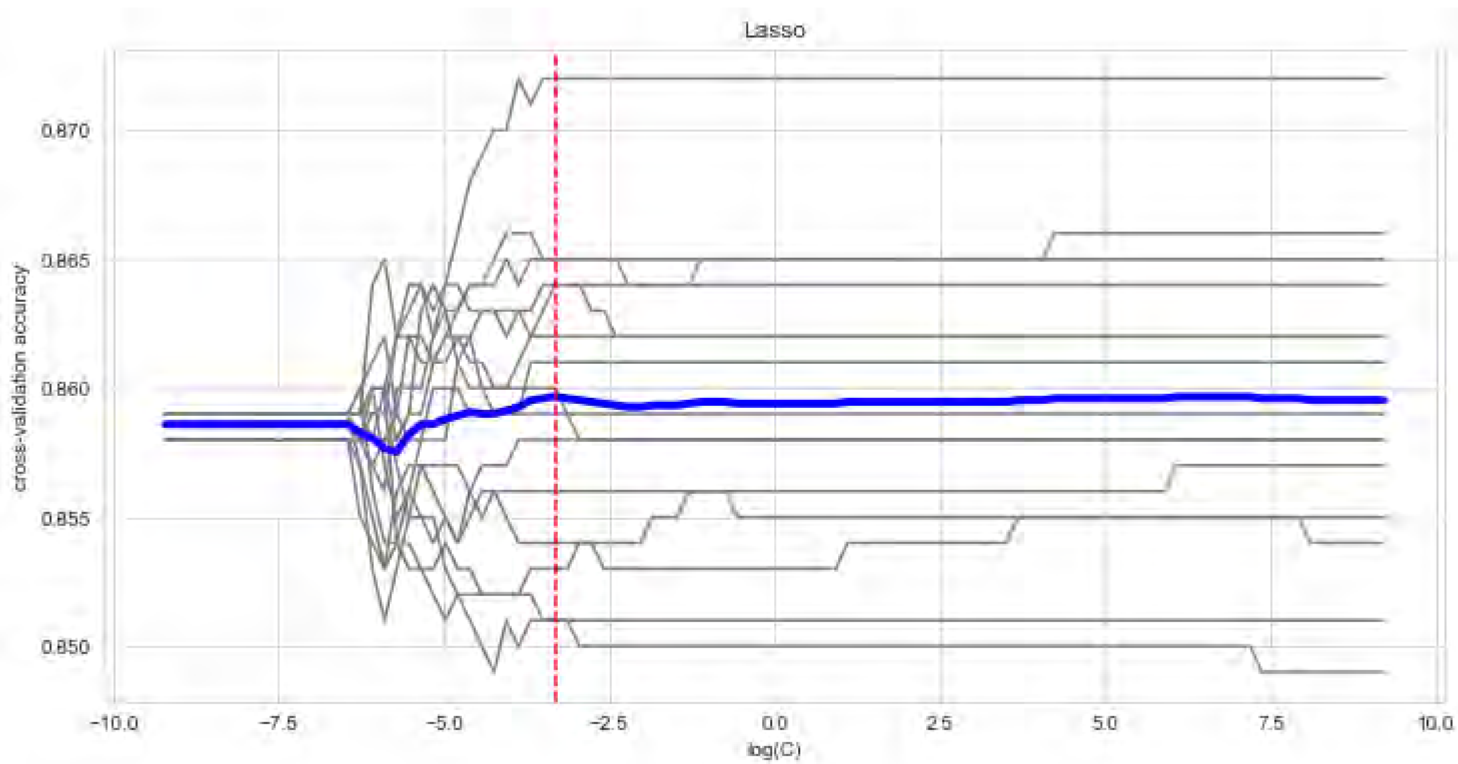

In [109]:

```
lasso_tune_results = LogisticRegressionCV(penalty='l1', Cs=101, cv=my_cv, solver='saga',
```

```
max_iter=5001, fit_intercept=False).\
fit(X_07, y_07.ravel())
```

In [110]:

```
fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(np.log(lasso_tune_results.Cs_), lasso_tune_results.scores_[1.0].T, color='grey')
ax.plot(np.log(lasso_tune_results.Cs_), lasso_tune_results.scores_[1.0].mean(axis=0),
color='blue', linewidth=4)
ax.axvline(x=np.log(lasso_tune_results.C_), color='red', linestyle='dashed')
ax.set_xlabel('log(C)')
ax.set_ylabel("cross-validation accuracy")
ax.set_title("Lasso")

plt.show()
```




In [111]:

```
print( ridge_tune_results.coef_)
```

```
[[-0.05411725  0.00833876 -0.01402398  0.00996263 -0.0187401  -0.0571266
  -0.01910932 -0.025187   -0.05939367 -0.02432039 -0.00739607 -0.00144733
  -0.01872073 -0.25064314]]
```

In [112]:

```
print( lasso_tune_results.coef_ )
```

```
[[-0.05363926  0.00853042 -0.0139784   0.008941   -0.01863751 -0.05632129
  -0.01870365 -0.02478074 -0.05858329 -0.02406857 -0.00782969 -0.00145634
  -0.01795627 -0.24481768]]
```

In [113]:

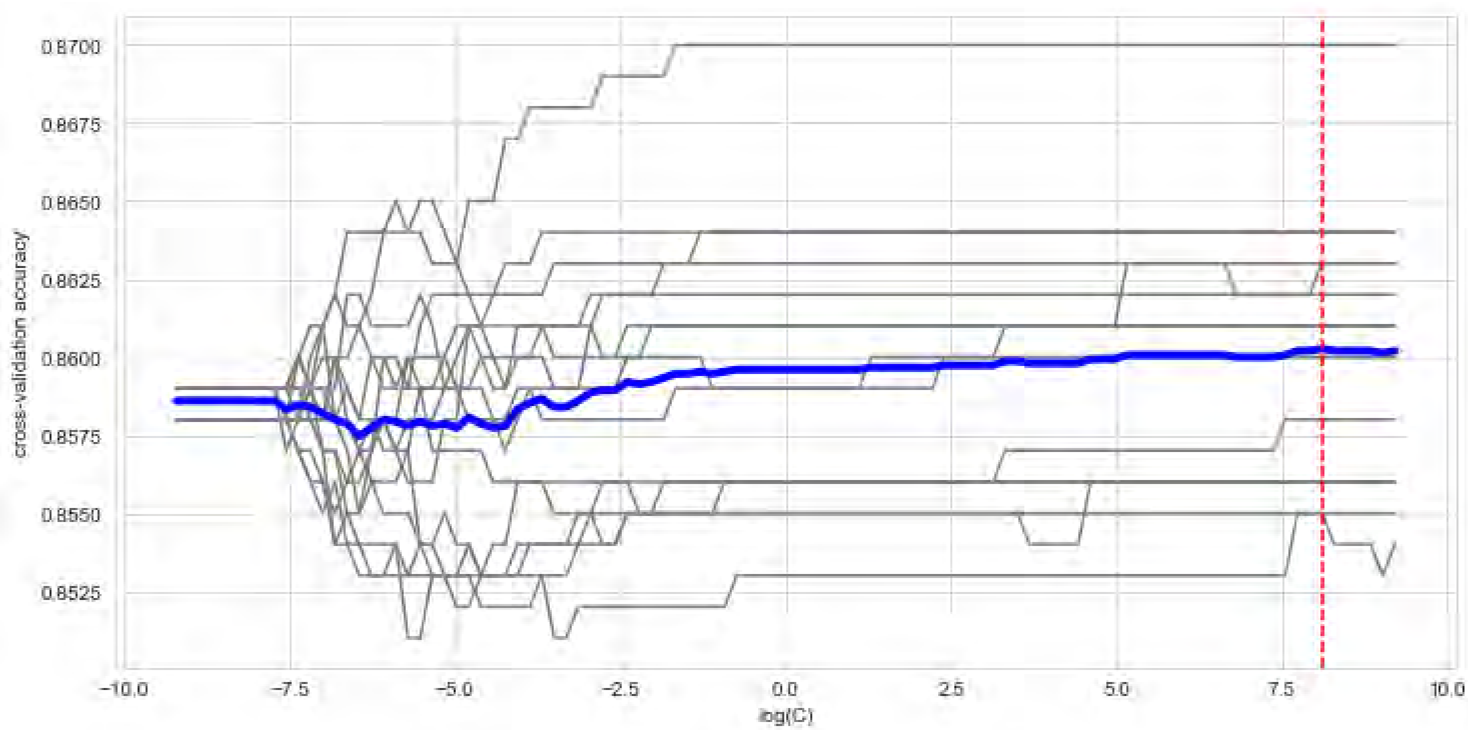
```
y_06, X_06 = dmatrices(formula_6 + ' - 1', data = df_main)
```

In [114]:

```
ridge_tune_results_6 = LogisticRegressionCV(penalty='l2', Cs=101, cv=my_cv,
solver='lbfgs', max_iter=5001, fit_intercept=False).\
fit(X_06, y_06.ravel())
```

In [115]:

```
fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(np.log(ridge_tune_results_6.Cs_), ridge_tune_results_6.scores_[1.0].T,
color='grey')
ax.plot(np.log(ridge_tune_results_6.Cs_), ridge_tune_results_6.scores_[1.0].mean(axis=0),
color='blue', linewidth=4)
ax.axvline(x=np.log(ridge_tune_results_6.C_), color='red', linestyle='dashed')
ax.set_xlabel('log(C)')
ax.set_ylabel("cross-validation accuracy")

plt.show()
```




In [116]:

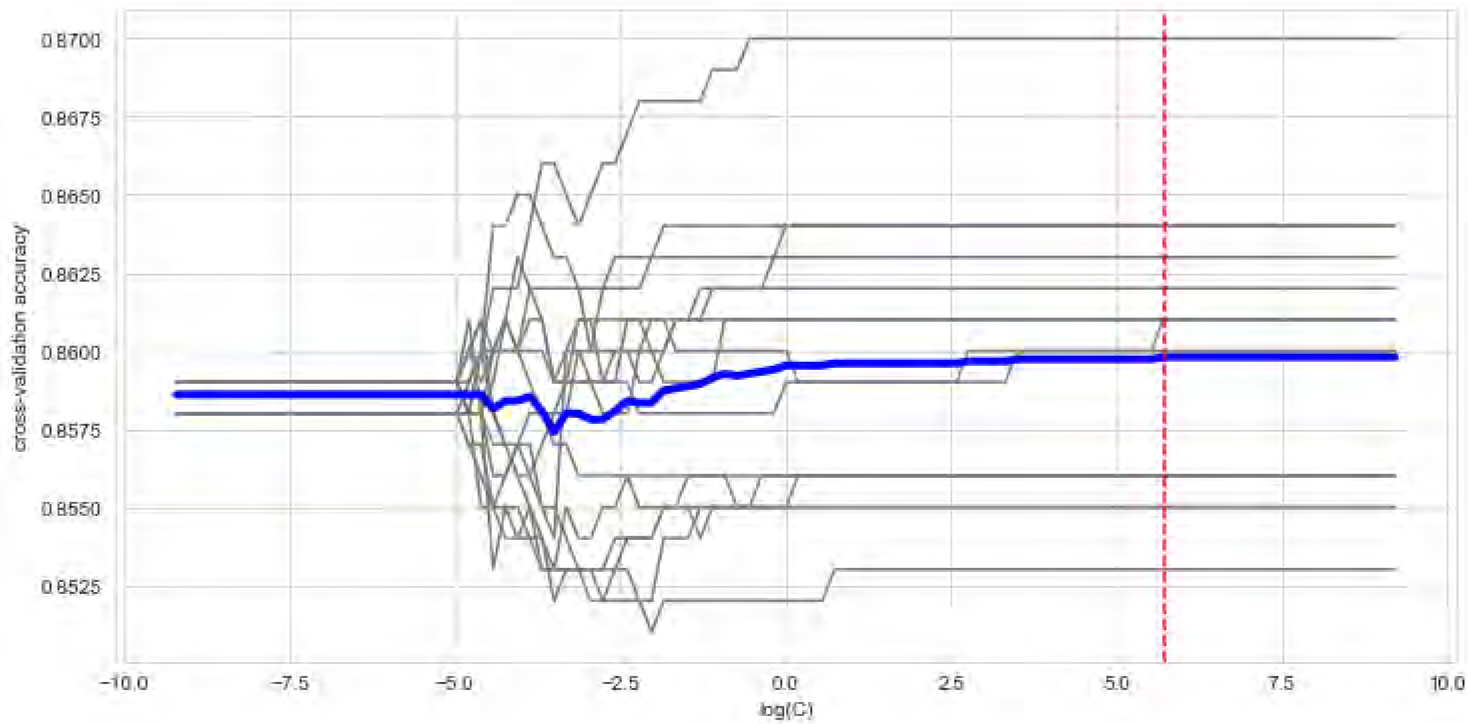
```
lasso_tune_results_6 = LogisticRegressionCV(penalty='l1', Cs=101, cv=my_cv, solver='saga',
max_iter=5001, fit_intercept=False).\
fit(X_06, y_06.ravel())
```

```
C:\Users\Vedant\anaconda3\envs\cmpinf2100\lib\site-packages\sklearn\linear_model\_sag.py:328
: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
C:\Users\Vedant\anaconda3\envs\cmpinf2100\lib\site-packages\sklearn\linear_model\_sag.py:328
: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

In [117]:

```
fig, ax = plt.subplots(figsize=(12, 6))
```

```
ax.plot(np.log(lasso_tune_results_6.Cs_), lasso_tune_results_6.scores_[1.0].T,
color='grey')
ax.plot(np.log(lasso_tune_results_6.Cs_), lasso_tune_results_6.scores_[1.0].mean(axis=0),
color='blue', linewidth=4)
ax.axvline(x=np.log(lasso_tune_results_6.C_), color='red', linestyle='dashed')
ax.set_xlabel('log(C)')
ax.set_ylabel("cross-validation accuracy")

plt.show()
```




In [118]:

```
print( ridge_tune_results.coef_)
```

```
[[-0.05411725  0.00833876 -0.01402398  0.00996263 -0.0187401  -0.0571266
  -0.01910932 -0.025187   -0.05939367 -0.02432039 -0.00739607 -0.00144733
  -0.01872073 -0.25064314]]
```

In [119]:

```
print( lasso_tune_results.coef_ )
```

```
[[-0.05363926  0.00853042 -0.0139784   0.008941   -0.01863751 -0.05632129
  -0.01870365 -0.02478074 -0.05858329 -0.02406857 -0.00782969 -0.00145634
  -0.01795627 -0.24481768]]
```

In [120]:

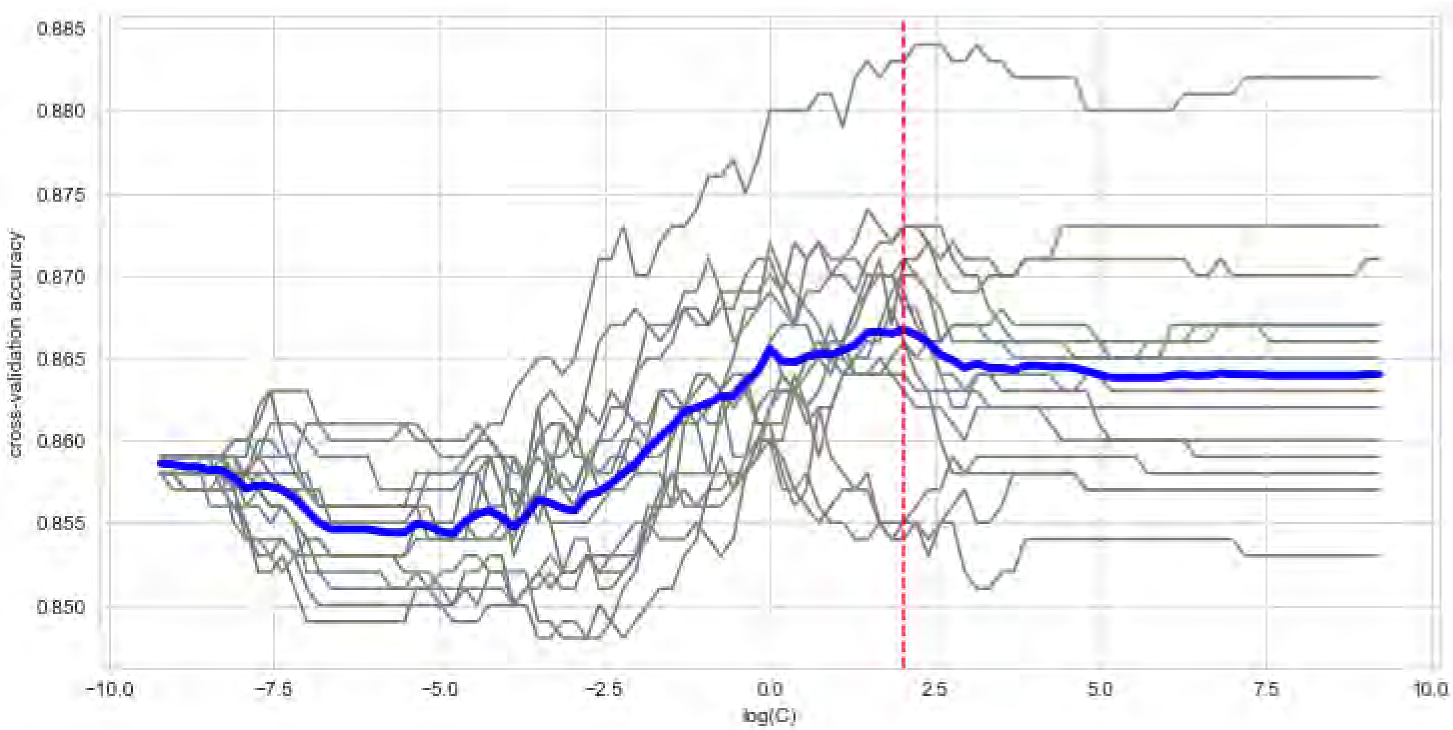
```
y_03, X_03 = dmatrices(formula_3 + ' - 1', data = df_main)
```

In [121]:

```
ridge_tune_results_3 = LogisticRegressionCV(penalty='l2', Cs=101, cv=my_cv,
solver='lbfgs', max_iter=5001, fit_intercept=False).\
fit(X_03, y_03.ravel())
```

In [122]:

```
fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(np.log(ridge_tune_results_3.Cs_), ridge_tune_results_3.scores_[1.0].T,
color='grey')
ax.plot(np.log(ridge_tune_results_3.Cs_), ridge_tune_results_3.scores_[1.0].mean(axis=0),
color='blue', linewidth=4)
ax.axvline(x=np.log(ridge_tune_results_3.C_), color='red', linestyle='dashed')
ax.set_xlabel('log(C)')
ax.set_ylabel("cross-validation accuracy")

plt.show()
```

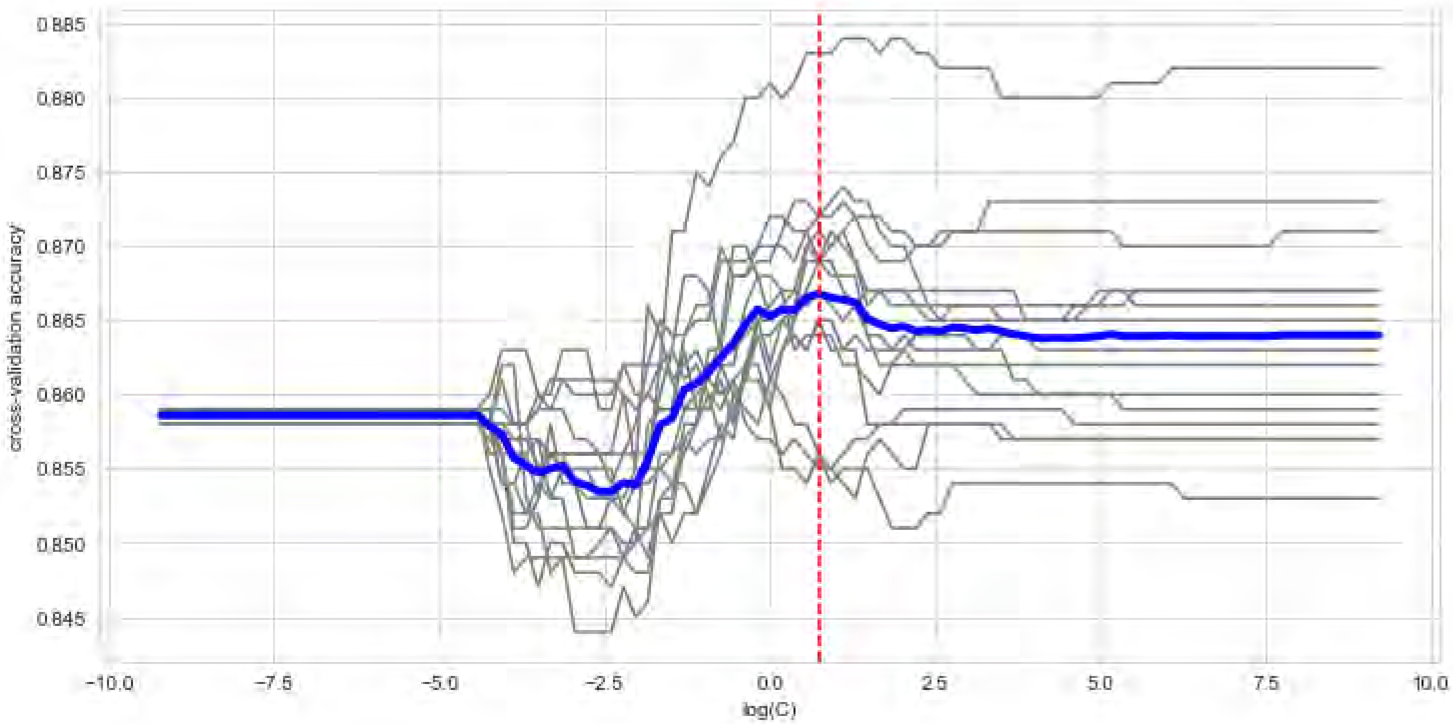



In [123]:

```
lasso_tune_results_3 = LogisticRegressionCV(penalty='l1', Cs=101, cv=my_cv, solver='saga',
max_iter=5001, fit_intercept=False).\
fit(X_03, y_03.ravel())
```

In [124]:

```
fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(np.log(lasso_tune_results_3.Cs_), lasso_tune_results_3.scores_[1.0].T,
color='grey')
ax.plot(np.log(lasso_tune_results_3.Cs_), lasso_tune_results_3.scores_[1.0].mean(axis=0),
color='blue', linewidth=4)
ax.axvline(x=np.log(lasso_tune_results_3.C_), color='red', linestyle='dashed')
ax.set_xlabel('log(C)')
ax.set_ylabel("cross-validation accuracy")

plt.show()
```




In [125]:

```
print( ridge_tune_results.coef_)
```

```
[[-0.05411725  0.00833876 -0.01402398  0.00996263 -0.0187401  -0.0571266
  -0.01910932 -0.025187   -0.05939367 -0.02432039 -0.00739607 -0.00144733
  -0.01872073 -0.25064314]]
```

In [126]:

```
print( lasso_tune_results.coef_ )
```

```
[[-0.05363926  0.00853042 -0.0139784   0.008941   -0.01863751 -0.05632129
  -0.01870365 -0.02478074 -0.05858329 -0.02406857 -0.00782969 -0.00145634
  -0.01795627 -0.24481768]]
```

As the varialbes are highly correlated the lasso penalty does not work properly. We can also validate this result from the result of elastic net where we get the best result when lasso penalty is set to zero.

In [ ]: