Abstract

This Jupyter Notebook is a showcase of various machine learning modelling methods, as well as data treatment methods such as resampling, identification and treatment of outliers, variable selection, and model evaluation.

Dataset

The data is anonymous banking customer data, with variables for transactions (incoming/outgoing), categorical variables such as age, sex and employment sector of customer, and interaction data for logins on banking app and website.

The aim is to predict customer response to marketing campaigns - i.e. which customers invested in a fund after a marketing campaign for that fund.

There are four response variables - dep_response, dep_inflow, ap_response, ap_inflow. We classify these into one kind of response variable by creating a simple binary response variable, binary_response, to indicate whether the customer responded to any marketing campaign at all.

We will predict this response variables with suitable models (i.e. classification), removing the other response variables in order to prevent data leakage/reduce collinearity.

Setup

In [1]:

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

df = pd.read_excel('FIP_RESP_ATTRIBUTES.xlsx',sheet_name='FIP_RESP_ATTRIBUTES')
key = pd.read_excel('FIP_RESP_ATTRIBUTES.xlsx',sheet_name='Sheet1')

df.drop(['kundennr'],axis=1,inplace=True)

binary_response = np.zeros((1,len(df))).T
bin_res = pd.Series(binary_response.squeeze(),dtype='int',name='binary_response')
df2 = pd.concat([bin_res,df],axis=1)

for i in range(1,len(df2)):
    if df2.dep_response.iloc[i]>0 or df2.ap_response.iloc[i]>0:
        df2.binary_response.iloc[i] = 1

df_0 = df2[df2.binary_response==0]
df_1 = df2[df2.binary_response==1]
```

```
C:\Users\gogar\anaconda3\lib\site-packages\pandas\core\indexing.py:670: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

Data Preprocessing

We have categorical variables, so we should create dummy variables for these. We also have outliers, so we will create a subset of the dataset where these are removed, using a cutoff of three times the standard deviation from the mean.

Our minority group (customers who responded) is much smaller than our majority group (customers who didn't respond), so we will resample with a proportion of 10% minority class, and have datasets with outliers included and outliers removed for these also.

The only variable which we will drop for the categorical variables is STAATSBGR, because it has too many categories. We also drop the original response variables.

In [42]:

```
# df2 is dataset with added binary_response variable

# dummy variables
df3 = df2.drop(['STAATSBGR','dep_response','dep_inflow','ap_response','ap_inflow'],axis=1)
df4 = pd.get_dummies(df3)

# remove outliers for numeric variables
df_numeric = df.drop(['SEX','AT_STAATSBGR','BERUFSGRUPPE'],axis=1)
df_standard = (df_numeric - df_numeric.mean())/df_numeric.std()
outlier_list = df_standard[(np.abs(df_standard)>3).any(1)].index.tolist()
df_outliers_removed = df4.drop(index=outlier_list)
```

In [3]:

```
# resampling both datasets
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
```

In [43]:

```
oversamp = SMOTE(sampling_strategy = 0.05)
undersamp = RandomUnderSampler(sampling_strategy = 0.12)
pipeline = Pipeline(steps = [('over', oversamp),('under', undersamp)])

X, y = pipeline.fit_resample(df4.drop(['binary_response'],axis=1),df4.binary_response)
X_outliers_removed, y_outliers_removed = pipeline.fit_resample(df_outliers_removed.drop(['binary_response'],axi
                                                              df_outliers_removed.binary_response)
```

Ten percent of resampled dataset is minority class.

In [5]:

```
print(y.sum()/len(y))
print(y_outliers_removed.sum()/len(y_outliers_removed))
```

```
0.10714335513197708
0.10714285714285714
```

We can now recombine X and y to make datasets with and without outliers. Then we can drop variables as necessary, in order to model.

In [44]:

```
# unresampled_df
unresampled_df = df4
unresampled_df_outliers_removed = df_outliers_removed

# resampled dataset
resampled_df = pd.concat([X, y],axis=1)
resampled_df_outliers_removed = pd.concat([X_outliers_removed, y_outliers_removed],axis=1)
```

Finally, we split the data into train, validation and test sets.

In [45]:

```
from sklearn.model_selection import train_test_split

# unresampled data, with outliers
X_train_FULL, X_NOT_train_FULL, y_train_FULL, y_NOT_train_FULL = train_test_split(unresampled_df.drop(['binary_
                                                                  unresampled_df.binary_response,test_siz
                                                                  random_state=101)
# unresampled data, outliers removed
X_train_OR, X_NOT_train_OR, y_train_OR, y_NOT_train_OR = train_test_split(unresampled_df_outliers_removed.drop
                                                          unresampled_df_outliers_removed.binary_response
                                                          random_state=101)

# resampled data, with outliers
X_train_RESAMP, X_NOT_train_RESAMP, y_train_RESAMP, y_NOT_train_RESAMP = train_test_split(resampled_df.drop(['b
                                                                  resampled_df.binary_response,test_size=
                                                                  random_state=101)
# resampled data, outliers removed
X_train_RESAMP_OR, X_NOT_train_RESAMP_OR, y_train_RESAMP_OR, y_NOT_train_RESAMP_OR = train_test_split(resampled
                                                          resampled_df_outliers_removed.binary_response,t
                                                          random_state=101)



# split 30% groups into validation and test sets
X_val_FULL, X_test_FULL, y_val_FULL, y_test_FULL = train_test_split(X_NOT_train_FULL,y_NOT_train_FULL,test_size
                                                               random_state=101)
X_val_OR, X_test_OR, y_val_OR, y_test_OR = train_test_split(X_NOT_train_OR,y_NOT_train_OR,test_size=0.5,
                                                               random_state=101)
X_val_RESAMP, X_test_RESAMP, y_val_RESAMP, y_test_RESAMP = train_test_split(X_NOT_train_RESAMP, y_NOT_train_RES
                                                                    random_state=101)
X_val_RESAMP_OR, X_test_RESAMP_OR, y_val_RESAMP_OR, y_test_RESAMP_OR = train_test_split(X_NOT_train_RESAMP_OR,
                                                                    random_state=101)
```

Check all the shapes to make sure that everything has worked out okay.

In [46]:

```
print(X_val_FULL.shape)
print(X_test_FULL.shape)
print(y_val_FULL.shape)
print(y_test_FULL.shape)
print(X_val_OR.shape)
print(X_test_OR.shape)
print(y_val_OR.shape)
print(y_test_OR.shape)
print(X_val_RESAMP.shape)
print(X_test_RESAMP.shape)
print(y_val_RESAMP.shape)
print(y_test_RESAMP.shape)
print(X_val_RESAMP_OR.shape)
print(X_test_RESAMP_OR.shape)
print(y_val_RESAMP_OR.shape)
print(y_test_RESAMP_OR.shape)
```

```
(23806, 44)
(23807, 44)
(23806,)
(23807,)
(17326, 44)
(17326, 44)
(17326,)
(17326,)
(10758, 44)
(10758, 44)
(10758,)
(10758,)
(8022, 44)
(8022, 44)
(8022,)
(8022,)
```

Models

The following models will be explored:

- Logistic Regression
- Principal Components Regression
- K Nearest Neighbours
- Decision Trees & Random Forests
- Support Vector Classifier
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis

Logistic Regression

To create a baseline model, we will perform logistic regression on the different datasets, and check the performance of the models.

In [15]:

```
import statsmodels.api as sm

# full dataset, not resampled
full_model = sm.Logit(y_train_FULL, X_train_FULL).fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.099193
         Iterations: 35
```

```
C:\Users\gogar\anaconda3\lib\site-packages\statsmodels\base\model.py:568: ConvergenceWarning: Maximum Likelihoo
d optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

The warning tells us that the model is failing to converge. When we use too many variables in a logistic regression, the probabilities in the Logit function: e^(BX)/(1+e^(BX)) can become to large or too small, and stop the model finding good estimates for the B coefficients.

In [16]:

```
# full dataset, outliers removed
OR_model = sm.Logit(y_train_OR, X_train_OR).fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.036666
         Iterations: 35
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-16-02c815ddcc08> in <module>
      1 # full dataset, outliers removed
----> 2 OR_model = sm.Logit(y_train_OR, X_train_OR).fit()

~\anaconda3\lib\site-packages\statsmodels\discrete\discrete_model.py in fit(self, start_params, method, maxite
r, full_output, disp, callback, **kwargs)
   1911         bnryfit = super(Logit, self).fit(start_params=start_params,
   1912                 method=method, maxiter=maxiter, full_output=full_output,
-> 1913                 disp=disp, callback=callback, **kwargs)
   1914
   1915         discretefit = LogitResults(self, bnryfit)

~\anaconda3\lib\site-packages\statsmodels\discrete\discrete_model.py in fit(self, start_params, method, maxite
r, full_output, disp, callback, **kwargs)
    214             start_params=start_params,
    215             method=method, maxiter=maxiter, full_output=full_output,
--> 216             disp=disp, callback=callback, **kwargs)
    217
    218         return mlefit  # It is up to subclasses to wrap results

~\anaconda3\lib\site-packages\statsmodels\base\model.py in fit(self, start_params, method, maxiter, full_outpu
t, disp, fargs, callback, retall, skip_hessian, **kwargs)
    531             Hinv = cov_params_func(self, xopt, retvals)
    532         elif method == 'newton' and full_output:
--> 533             Hinv = np.linalg.inv(-retvals['Hessian']) / nobs
    534         elif not skip_hessian:
    535             H = -1 * self.hessian(xopt)

<__array_function__ internals> in inv(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in inv(a)
    545     signature = 'D->D' if isComplexType(t) else 'd->d'
    546     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 547     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    548     return wrap(ainv.astype(result_t, copy=False))
    549

~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in _raise_linalgerror_singular(err, flag)
     95
     96 def _raise_linalgerror_singular(err, flag):
---> 97     raise LinAlgError("Singular matrix")
     98
     99 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

We get the warning of a Singular matrix. This is likely because, with a large amount of collinearity in the model, the matrix has too low rank and cannot be used to estimate the coefficients.

In [17]:

```
# resampled dataset
resampled_model = sm.Logit(y_train_RESAMP, X_train_RESAMP).fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.191995
         Iterations: 35
```

```
C:\Users\gogar\anaconda3\lib\site-packages\statsmodels\base\model.py:568: ConvergenceWarning: Maximum Likelihoo
d optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

Same warning as before.

In [18]:

```
# resampled dataset, outliers removed
resamp_OR_model = sm.Logit(y_train_RESAMP_OR, X_train_RESAMP_OR).fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.159823
         Iterations: 35
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-18-e9c49b625a07> in <module>
      1 # resampled dataset, outliers removed
----> 2 resamp_OR_model = sm.Logit(y_train_RESAMP_OR, X_train_RESAMP_OR).fit()

~\anaconda3\lib\site-packages\statsmodels\discrete\discrete_model.py in fit(self, start_params, method, maxite
r, full_output, disp, callback, **kwargs)
   1911         bnryfit = super(Logit, self).fit(start_params=start_params,
   1912                 method=method, maxiter=maxiter, full_output=full_output,
-> 1913                 disp=disp, callback=callback, **kwargs)
   1914
   1915         discretefit = LogitResults(self, bnryfit)

~\anaconda3\lib\site-packages\statsmodels\discrete\discrete_model.py in fit(self, start_params, method, maxite
r, full_output, disp, callback, **kwargs)
    214             start_params=start_params,
    215             method=method, maxiter=maxiter, full_output=full_output,
--> 216             disp=disp, callback=callback, **kwargs)
    217
    218         return mlefit  # It is up to subclasses to wrap results

~\anaconda3\lib\site-packages\statsmodels\base\model.py in fit(self, start_params, method, maxiter, full_outpu
t, disp, fargs, callback, retall, skip_hessian, **kwargs)
    531             Hinv = cov_params_func(self, xopt, retvals)
    532         elif method == 'newton' and full_output:
--> 533             Hinv = np.linalg.inv(-retvals['Hessian']) / nobs
    534         elif not skip_hessian:
    535             H = -1 * self.hessian(xopt)

<__array_function__ internals> in inv(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in inv(a)
    545     signature = 'D->D' if isComplexType(t) else 'd->d'
    546     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 547     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    548     return wrap(ainv.astype(result_t, copy=False))
    549

~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in _raise_linalgerror_singular(err, flag)
     95
     96 def _raise_linalgerror_singular(err, flag):
---> 97     raise LinAlgError("Singular matrix")
     98
     99 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

Same warning as before.

All of these warnings are telling us that we are putting too many variables into the model. We should restrict what we put it.

We could find that out by exploring correlations (to reduce collinearity) or by performing variable selection methods, such as the Lasso.

The Lasso is a linear regression method, and so we need to predict a continuous variable with it, not a categorical variable like binary_response. We will use dep_inflow, as this has a good range of values. Actual predictions are not important, just the variable coefficients which we get.

In [45]:

```
# Lasso with Cross Validation
from sklearn.linear_model import LassoCV
from sklearn.model_selection import cross_val_score, RepeatedKFold
lasso = LassoCV(cv=10)

X = df_numeric.drop(['ap_inflow','ap_response','dep_response','dep_inflow'],axis=1)
y = df_numeric.dep_inflow
```

In [27]:

In [46]:

```
lasso.fit(X,y)
```

Out[46]:

```
LassoCV(cv=10)
```

In [52]:

```
lasso.coef_
```

Out[52]:

```
array([-0.        , -0.        ,  0.00123336,  0.        , -0.        ,
        0.        ,  0.08907592, -0.        , -0.        ,  0.        ,
        0.        , -0.        , -0.        , -0.        , -0.        ,
        0.00718844, -0.        , -0.        , -0.        , -0.        ,
       -0.        , -0.        ,  0.        ,  0.        ,  0.        ,
        0.        ])
```

The Lasso generates coefficients for the variables, and the variables which aren't deemed important are given a coefficient of 0.

In [56]:

```
pd.concat([pd.Series(X.columns),pd.Series(lasso.coef_)],axis=1)
```

Out[56]:

	0	1
0	Emails_Received	-0.000000
1	ANZ_DS	-0.000000
2	VOL_DS	0.001233
3	ANZ_DK	0.000000
4	VOL_DK	-0.000000
5	ANZ_DEP	0.000000
6	VOL_DEP	0.089076
7	ANZ_GIRO	-0.000000
8	VOL_GIRO	-0.000000
9	KSEIT	0.000000
10	ALTER	0.000000
11	STAATSBGR	-0.000000
12	ANZ_DAUERSPAREN	-0.000000
13	DAUERSPAR_BETRAG	-0.000000
14	ANZ_EINGANG_3M	-0.000000
15	BETRAG_EINGANG_3M	0.007188
16	ANZ_AUSGANG_3M	-0.000000
17	BETRAG_AUSGANG_3M	-0.000000
18	ANZ_ANSPARPLAN	-0.000000
19	VOL_ANSPARPLAN	-0.000000
20	LWP_KAUF	-0.000000
21	LWP_VERKAUF	-0.000000
22	ALTER_ERST_DP	0.000000
23	LOGIN_WEB_3M	0.000000
24	LOGIN_MOBILE_B_3M	0.000000
25	LOGIN_MOBILE_APP_3M	0.000000

Three variables is probably too few to get a good model, but we can check a logistic regression using the three variables which the Lasso gave non-zero coefficients to. The models converge, which is an improvement, but their results aren't great.

In [50]:

```
reduced_model = sm.Logit(y_train_FULL, X_train_FULL[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).fit()
reduced_model_OR = sm.Logit(y_train_OR, X_train_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).fit()

resampled_model_full = sm.Logit(y_train_RESAMP, X_train_RESAMP[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).fit()
resampled_model_OR = sm.Logit(y_train_RESAMP_OR, X_train_RESAMP_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).f
```

```
Optimization terminated successfully.
         Current function value: 0.474147
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.346768
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.577098
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.559141
         Iterations 7
```

In [51]:

```
preds1 = reduced_model.predict(X_val_FULL[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).round()
preds2 = reduced_model_OR.predict(X_val_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).round()
preds3 = resampled_model_full.predict(X_val_RESAMP[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).round()
preds4 = resampled_model_OR.predict(X_val_RESAMP_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']]).round()
```

In [52]:

```
from sklearn.metrics import confusion_matrix,classification_report
```

In [53]:

```
print('Full Dataset')
print(confusion_matrix(y_val_FULL,preds1))
print('\n')
print(classification_report(y_val_FULL,preds1))
print('\n')

print('Outliers Removed Dataset')
print(confusion_matrix(y_val_OR,preds2))
print('\n')
print(classification_report(y_val_OR,preds2))
print('\n')

print('Resampled Dataset')
print(confusion_matrix(y_val_RESAMP,preds3))
print('\n')
print(classification_report(y_val_RESAMP,preds3))
print('\n')

print('Resampled Outliers Removed')
print(confusion_matrix(y_val_RESAMP_OR,preds4))
print('\n')
print(classification_report(y_val_RESAMP_OR,preds4))
```

```
Full Dataset
[[22739   317]
 [  626   124]]


              precision    recall  f1-score   support

           0       0.97      0.99      0.98     23056
           1       0.28      0.17      0.21       750

    accuracy                           0.96     23806
   macro avg       0.63      0.58      0.59     23806
weighted avg       0.95      0.96      0.96     23806



Outliers Removed Dataset
[[17169    23]
 [  131     3]]


              precision    recall  f1-score   support

           0       0.99      1.00      1.00     17192
           1       0.12      0.02      0.04       134

    accuracy                           0.99     17326
   macro avg       0.55      0.51      0.52     17326
weighted avg       0.99      0.99      0.99     17326



Resampled Dataset
[[9309  281]
 [ 791  377]]


              precision    recall  f1-score   support

           0       0.92      0.97      0.95      9590
           1       0.57      0.32      0.41      1168

    accuracy                           0.90     10758
   macro avg       0.75      0.65      0.68     10758
weighted avg       0.88      0.90      0.89     10758



Resampled Outliers Removed
[[7084   99]
 [ 623  216]]


              precision    recall  f1-score   support

           0       0.92      0.99      0.95      7183
           1       0.69      0.26      0.37       839

    accuracy                           0.91      8022
   macro avg       0.80      0.62      0.66      8022
weighted avg       0.89      0.91      0.89      8022
```

Let's see what variables we get if we use the Lasso on a standardised version of the dataset with all of the outliers removed.

In [47]:

```
lasso2 = LassoCV(cv=10)
X1 = df_standard[(np.abs(df_standard)<3)].drop(['ap_inflow','ap_response','dep_response'],axis=1).dropna()
y1 = X1.dep_inflow
X1 = X1.drop(['dep_inflow'],axis=1)
lasso2.fit(X1,y1)
lasso2.coef_
```

Out[47]:

```
array([-0.00207218, -0.00126965,  0.00399699, -0.        ,  0.        ,
        0.03118783,  0.18465067,  0.        ,  0.        ,  0.00075931,
        0.0013635 , -0.00160013, -0.00093554, -0.        ,  0.        ,
        0.00041   , -0.00193927, -0.00091861, -0.02488596, -0.        ,
       -0.00572193, -0.00578035, -0.01802819,  0.00826852,  0.00215816,
        0.00106755])
```

In [48]:

```
coefs = pd.concat([pd.Series(X1.columns),pd.Series(lasso2.coef_)],axis=1)
coefs[coefs.loc[:,1]>0]
```

Out[48]:

	0	1
2	VOL_DS	0.003997
5	ANZ_DEP	0.031188
6	VOL_DEP	0.184651
9	KSEIT	0.000759
10	ALTER	0.001364
15	BETRAG_EINGANG_3M	0.000410
23	LOGIN_WEB_3M	0.008269
24	LOGIN_MOBILE_B_3M	0.002158
25	LOGIN_MOBILE_APP_3M	0.001068

Let's try these variables on a Logistic Regression.

In [49]:

```
reduced_model2 = sm.Logit(y_train_FULL, X_train_FULL[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINC
reduced_model_OR2 = sm.Logit(y_train_OR, X_train_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGA

resampled_model_full2 = sm.Logit(y_train_RESAMP, X_train_RESAMP[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','
resampled_model_OR2 = sm.Logit(y_train_RESAMP_OR, X_train_RESAMP_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTE
```

```
Optimization terminated successfully.
         Current function value: 0.127099
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.046047
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.265284
         Iterations 7
Optimization terminated successfully.
         Current function value: 0.294096
         Iterations 7
```

The models all converge, which is a good sign. Let's check their outputs.

In [55]:

```
preds5 = reduced_model2.predict(X_val_FULL[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','I
preds6 = reduced_model_OR2.predict(X_val_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','
preds7 = resampled_model_full2.predict(X_val_RESAMP[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGA
preds8 = resampled_model_OR2.predict(X_val_RESAMP_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINC
```

In [56]:

```
print('Full Dataset')
print(confusion_matrix(y_val_FULL,preds5))
print('\n')
print(classification_report(y_val_FULL,preds5))
print('\n')

print('Outliers Removed Dataset')
print(confusion_matrix(y_val_OR,preds6))
print('\n')
print(classification_report(y_val_OR,preds6))
print('\n')

print('Resampled Dataset')
print(confusion_matrix(y_val_RESAMP,preds7))
print('\n')
print(classification_report(y_val_RESAMP,preds7))
print('\n')

print('Resampled Outliers Removed')
print(confusion_matrix(y_val_RESAMP_OR,preds8))
print('\n')
print(classification_report(y_val_RESAMP_OR,preds8))
```

```
Full Dataset
[[22890   166]
 [  683    67]]


              precision    recall  f1-score   support

           0       0.97      0.99      0.98     23056
           1       0.29      0.09      0.14       750

    accuracy                           0.96     23806
   macro avg       0.63      0.54      0.56     23806
weighted avg       0.95      0.96      0.96     23806



Outliers Removed Dataset
[[17187     5]
 [  134     0]]


              precision    recall  f1-score   support

           0       0.99      1.00      1.00     17192
           1       0.00      0.00      0.00       134

    accuracy                           0.99     17326
   macro avg       0.50      0.50      0.50     17326
weighted avg       0.98      0.99      0.99     17326



Resampled Dataset
[[9235  355]
 [ 808  360]]


              precision    recall  f1-score   support

           0       0.92      0.96      0.94      9590
           1       0.50      0.31      0.38      1168

    accuracy                           0.89     10758
   macro avg       0.71      0.64      0.66     10758
weighted avg       0.87      0.89      0.88     10758



Resampled Outliers Removed
[[7059  124]
 [ 651  188]]


              precision    recall  f1-score   support

           0       0.92      0.98      0.95      7183
           1       0.60      0.22      0.33       839

    accuracy                           0.90      8022
   macro avg       0.76      0.60      0.64      8022
weighted avg       0.88      0.90      0.88      8022
```

Explanations in document.

Principal Components Analysis & Regression

We can explore Principal Components Analysis and Regression. This only works for continuous numeric variables, so we will try to predict dep_inflow, as we did before with the Lasso. We use all variables, and will use outliers and no-outliers data. The data are scaled - as is common in PCA and PCR. We cannot use our resampled data here, as the variable which we are predicting is continuous.

In [92]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline

pcr1 = make_pipeline(StandardScaler(), PCA(n_components=3), LinearRegression())
pcr2 = make_pipeline(StandardScaler(), PCA(n_components=3), LinearRegression())
```

In [102…

```
pcr1.fit(X_train_FULL, df.dep_inflow.iloc[X_train_FULL.index.tolist()])
```

Out[102…

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('pca', PCA(n_components=3)),
                ('linearregression', LinearRegression())])
```

In [105…

```
predsv = pcr1.predict(X_val_FULL)
```

In [110…

```
ys = df.dep_inflow.iloc[X_val_FULL.index.tolist()]
```

In [122…

```
acc = 0

for i in range(1,len(ys)):
    acc += np.square(ys.iloc[i] - predsv.tolist()[i])
```

In [124…

```
np.sqrt(acc/len(ys))
```

Out[124…

```
2461.152453581968
```

In [125…

```
np.mean(ys)
```

Out[125…

```
231.62396580693922
```

We get a mean predicted result which is very far from the mean of dep_inflow. We can try again using the data with outliers removed.

In [129…

```
pcr2.fit(X_train_OR, df.dep_inflow.iloc[X_train_OR.index.tolist()])
```

Out[129…

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('pca', PCA(n_components=3)),
                ('linearregression', LinearRegression())])
```

In [130…

```
predstt = pcr2.predict(X_val_OR)
```

In [132…

```
yst = df.dep_inflow.iloc[X_val_OR.index.tolist()]
```

In [134…

```
acc = 0

for i in range(1,len(yst)):
    acc += np.square(yst.iloc[i] - predstt.tolist()[i])

np.sqrt(acc/len(yst))
```

Out[134…

```
412.6604214420549
```

This is much closer to the mean of dep_inflow, so it's possible that this model has better predictive capabilities. However, in general, PCR would not be a good method for predicting variables in this case for two reasons. The first reason is that the Principal Components Regression model is not easily interpretable, and does not tell us which variables are important.

The second reason is that PCR does not actually optimise its methods based on the predicted variable (like linear/logistic regression). Instead, it finds the directions of most variation, and makes the assumption that these will be highly correlated with the predicted variable. This is not always a good assumption to make, and PCA/PCR should generally be used as an exploratory method, not a predictive method.

KNN, Cohens Kappa

Now we can move on to more classification methods. First we will try KNN and Cohen's Kappa to compare results. We have to use a subset of the training data, as the full training set caused my computer to freeze several times. We just reset the index for all large training sets, and sample 10000 numbers for the index.

In [6]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from random import sample
```

In [14]:

```
X_train_FULL.reset_index(drop=True,inplace=True)
y_train_FULL.reset_index(drop=True,inplace=True)
X_train_OR.reset_index(drop=True,inplace=True)
y_train_OR.reset_index(drop=True,inplace=True)
X_train_RESAMP.reset_index(drop=True,inplace=True)
y_train_RESAMP.reset_index(drop=True,inplace=True)
X_train_RESAMP_OR.reset_index(drop=True,inplace=True)
y_train_RESAMP_OR.reset_index(drop=True,inplace=True)
```

In [15]:

```
sample_index = sample(X_train_FULL.index.tolist(),10000)
```

In [40]:

```
k_range = range(2, 10)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_FULL.iloc[sample_index], y_train_FULL.iloc[sample_index], cv=5, scori
    k_scores.append(scores.mean())
```

In [42]:

```
k_scores
```

Out[42]:

```
[0.9654,
 0.9604000000000001,
 0.9649999999999999,
 0.9625999999999999,
 0.9661,
 0.9643,
 0.9664000000000001,
 0.9649000000000001]
```

This is a strange result. There is no difference between the number of neighbours. This is probably due to the high imbalance in the classes. We can try again with the resampled dataset.

In [23]:

```
sample_index = sample(X_train_RESAMP.index.tolist(),10000)
```

In [18]:

```
k_range = range(2, 10)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_RESAMP.iloc[sample_index], y_train_RESAMP.iloc[sample_index], cv=5, s
    k_scores.append(scores.mean())
```

In [19]:

```
k_scores
```

Out[19]:

```
[0.9089,
 0.9132999999999999,
 0.9131,
 0.9128999999999999,
 0.9149,
 0.9151,
 0.9144,
 0.9149]
```

These look a little more realistic. Let's try with the outliers removed.

In [20]:

```
sample_index = sample(X_train_RESAMP_OR.index.tolist(),10000)
```

In [21]:

```
k_range = range(2, 10)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_RESAMP_OR.iloc[sample_index], y_train_RESAMP_OR.iloc[sample_index], c
    k_scores.append(scores.mean())
```

In [22]:

```
k_scores
```

Out[22]:

```
[0.9137000000000001,
 0.9085000000000001,
 0.9118,
 0.9088999999999998,
 0.9115,
 0.9107,
 0.9105000000000001,
 0.9107]
```

These are very similar scores to above. We can test the validity of these models with the training and validation sets, and Cohen's Kappa. We will select 6 neighbors - though it doesn't seem to matter too much.

Commonly, KNN is fitted with the training set and used to predict the validation set. Then a new KNN is fitted to the validation set, and the two models are compared to see how generaliseable the KNN is.

In [30]:
```
X_val_RESAMP.reset_index(drop=True,inplace=True)
y_val_RESAMP.reset_index(drop=True,inplace=True)
X_val_RESAMP_OR.reset_index(drop=True,inplace=True)
y_val_RESAMP_OR.reset_index(drop=True,inplace=True)

sample_index1 = sample(X_train_RESAMP.index.tolist(),10000)
sample_index2 = sample(X_val_RESAMP.index.tolist(),10000)
```

In [37]:
```
knn1 = KNeighborsClassifier(n_neighbors=6)
knn1.fit(X_train_RESAMP.iloc[sample_index1], y_train_RESAMP.iloc[sample_index1])
```

Out[37]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [38]:
```
preds_RESAMP = knn1.predict(X_val_RESAMP.iloc[sample_index2])
```

In [39]:
```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(preds_RESAMP, y_val_RESAMP.iloc[sample_index2]))
```

```
[[8676  652]
 [ 233  439]]
```

In [41]:
```
knn2 = KNeighborsClassifier(n_neighbors = 6)
knn2.fit(X_val_RESAMP.iloc[sample_index2], y_val_RESAMP.iloc[sample_index2])
```

Out[41]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [43]:
```
preds_VAL = knn2.predict(X_val_RESAMP.iloc[sample_index2])
```

In [44]:
```
print(confusion_matrix(preds_VAL, y_val_RESAMP.iloc[sample_index2]))
```

```
[[8756  592]
 [ 153  499]]
```

Just by comparing the two confusion matrices, we can see that the numbers are similar. There is a small increase in accuracy in the model trained on the validation set, but it is low enough to say that the clustering is generaliseable.

We can also check the predictions with Cohen's Kappa. We get 64% agreement, which is a relatively decent score.

In [57]:
```
cohen_kappa_score(preds_VAL, preds_RESAMP)
```

Out[57]:
```
0.6457374613708246
```

Finally we can do the same for the data with outliers removed, and check if their is any improvement in accuracy.

In [62]:
```
knn3 = KNeighborsClassifier(n_neighbors = 6)
knn3.fit(X_train_RESAMP_OR.iloc[sample_index1], y_train_RESAMP_OR.iloc[sample_index1])
preds_OR = knn2.predict(X_val_RESAMP_OR.iloc[sample_index2])
```

In [63]:
```
knn4 = KNeighborsClassifier(n_neighbors = 6)
knn4.fit(X_val_RESAMP_OR.iloc[sample_index2], y_val_RESAMP_OR.iloc[sample_index2])
preds_VAL_OR = knn4.predict(X_val_RESAMP_OR.iloc[sample_index2])
```

In [64]:
```
cohen_kappa_score(preds_OR, preds_VAL_OR)
```

Out[64]:
```
0.5689034638049917
```

The Kappa score has reduced here - perhaps the outliers were useful.

Decision Trees and Random Forests

A random forest is just a collection of decision trees. The gain in accuracy is offset by a loss of interpretability.

In [68]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [72]:
```
tree1 = DecisionTreeClassifier()
tree1.fit(X_train_FULL, y_train_FULL)

tree2 = DecisionTreeClassifier()
tree2.fit(X_train_OR, y_train_OR)

tree3 = DecisionTreeClassifier()
tree3.fit(X_train_RESAMP, y_train_RESAMP)

tree4 = DecisionTreeClassifier()
tree4.fit(X_train_RESAMP_OR, y_train_RESAMP_OR)
```

Out[72]:
```
DecisionTreeClassifier()
```

In [73]:
```
pred1 = tree1.predict(X_val_FULL)
pred2 = tree2.predict(X_val_OR)
pred3 = tree3.predict(X_val_RESAMP)
pred4 = tree4.predict(X_val_RESAMP_OR)
```

In [76]:
```
print('pred1')
print(classification_report(pred1,y_val_FULL))
print('\n')
print('pred2')
print(classification_report(pred2,y_val_OR))
print('\n')
print('pred3')
print(classification_report(pred3,y_val_RESAMP))
print('\n')
print('pred4')
print(classification_report(pred4,y_val_RESAMP_OR))
print('\n')
```

```
pred1
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     22915
           1       0.42      0.36      0.39       891

    accuracy                           0.96     23806
   macro avg       0.70      0.67      0.68     23806
weighted avg       0.95      0.96      0.96     23806


pred2
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     17167
           1       0.29      0.25      0.27       159

    accuracy                           0.99     17326
   macro avg       0.64      0.62      0.63     17326
weighted avg       0.99      0.99      0.99     17326


pred3
              precision    recall  f1-score   support

           0       0.95      0.96      0.95      9500
           1       0.64      0.59      0.62      1258

    accuracy                           0.91     10758
   macro avg       0.79      0.77      0.78     10758
weighted avg       0.91      0.91      0.91     10758


pred4
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      7088
           1       0.79      0.71      0.74       934

    accuracy                           0.94      8022
   macro avg       0.87      0.84      0.86      8022
weighted avg       0.94      0.94      0.94      8022
```

In [77]:
```
forest1 = RandomForestClassifier()
forest1.fit(X_train_FULL, y_train_FULL)

forest2 = RandomForestClassifier()
forest2.fit(X_train_OR, y_train_OR)

forest3 = RandomForestClassifier()
forest3.fit(X_train_RESAMP, y_train_RESAMP)

forest4 = RandomForestClassifier()
forest4.fit(X_train_RESAMP_OR, y_train_RESAMP_OR)
```

Out[77]:
```
RandomForestClassifier()
```

In [78]:
```
forestpred1 = forest1.predict(X_val_FULL)
forestpred2 = forest2.predict(X_val_OR)
forestpred3 = forest3.predict(X_val_RESAMP)
forestpred4 = forest4.predict(X_val_RESAMP_OR)
```

In [79]:
```
print('forestpred1')
print(classification_report(forestpred1,y_val_FULL))
print('\n')
print('forestpred2')
print(classification_report(forestpred2,y_val_OR))
print('\n')
print('forestpred3')
print(classification_report(forestpred3,y_val_RESAMP))
print('\n')
print('forestpred4')
print(classification_report(forestpred4,y_val_RESAMP_OR))
print('\n')
```

```
forestpred1
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     23559
           1       0.28      0.84      0.42       247

    accuracy                           0.98     23806
   macro avg       0.64      0.91      0.70     23806
weighted avg       0.99      0.98      0.98     23806


forestpred2
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     17302
           1       0.18      1.00      0.30        24

    accuracy                           0.99     17326
   macro avg       0.59      1.00      0.65     17326
weighted avg       1.00      0.99      1.00     17326


forestpred3
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      9820
           1       0.66      0.82      0.73       938

    accuracy                           0.95     10758
   macro avg       0.82      0.89      0.85     10758
weighted avg       0.95      0.95      0.95     10758


forestpred4
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      7308
           1       0.81      0.95      0.88       714

    accuracy                           0.98      8022
   macro avg       0.90      0.97      0.93      8022
weighted avg       0.98      0.98      0.98      8022
```

Explanation in document.

Support Vector Classifier

If we have p variables, we can think of this as a p-dimensional space.

In 3 dimensions, a plane is 2 dimensional and flat in the 3 dimensional space. It separates points from one side of the plane, from points on the other.

This idea generalises to p dimensions. A plane in (p-1) dimensions will be flat in p-dimensional space. This plane is known as a hyperplane.

One issue with SVC is that it requires a flat division of the points from different classes. This is not always the case in datasets.

In [81]:
```
from sklearn.svm import SVC
```

In [82]:
```
svc1 = SVC()
svc1.fit(X_train_FULL, y_train_FULL)

svc2 = SVC()
svc2.fit(X_train_OR, y_train_OR)

svc3 = SVC()
svc3.fit(X_train_RESAMP, y_train_RESAMP)

svc4 = SVC()
svc4.fit(X_train_RESAMP_OR, y_train_RESAMP_OR)
```

Out[82]:
```
SVC()
```

In [83]:
```
svc_preds1 = svc1.predict(X_val_FULL)
svc_preds2 = svc2.predict(X_val_OR)
svc_preds3 = svc3.predict(X_val_RESAMP)
svc_preds4 = svc4.predict(X_val_RESAMP_OR)
```

In [84]:
```
print('svc_preds1')
print(classification_report(svc_preds1,y_val_FULL))
print('\n')
print('svc_preds2')
print(classification_report(svc_preds2,y_val_OR))
print('\n')
print('svc_preds3')
print(classification_report(svc_preds3,y_val_RESAMP))
print('\n')
print('svc_preds4')
print(classification_report(svc_preds4,y_val_RESAMP_OR))
print('\n')
```

```
svc_preds1
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     23776
           1       0.02      0.53      0.04        30

    accuracy                           0.97     23806
   macro avg       0.51      0.75      0.51     23806
weighted avg       1.00      0.97      0.98     23806


svc_preds2
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     17326
           1       0.00      0.00      0.00         0

    accuracy                           0.99     17326
   macro avg       0.50      0.50      0.50     17326
weighted avg       1.00      0.99      1.00     17326


svc_preds3
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      9933
           1       0.46      0.66      0.54       825

    accuracy                           0.92     10758
   macro avg       0.72      0.80      0.75     10758
weighted avg       0.93      0.92      0.92     10758


svc_preds4
              precision    recall  f1-score   support

           0       0.99      0.93      0.96      7653
           1       0.33      0.74      0.45       369

    accuracy                           0.92      8022
   macro avg       0.66      0.83      0.70      8022
weighted avg       0.96      0.92      0.93      8022
```

```
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Rec
all and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Rec
all and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Rec
all and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Explanation in document.

Linear & Quadratic Discriminant Analysis

Unlike Support Vector Classifiers, Discriminant Analysis assumes that the data follow a multivariate normal distribution. This is not always true, but LDA and QDA have a good track-record of predictions, even for data which is not actually normally distributed.

Both methods seek to find the boundary between classes, and both use Bayes' Theorem. LDA assumes that the covariance structure is the same for both classes, whereas QDA allows different classes to have different covariance structures. Because of how this works out mathematically, we get a linear boundary for LDA and a curvy boundary for QDA.

In [58]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
```

In [87]:
```
lda1 = LinearDiscriminantAnalysis()
lda1.fit(X_train_FULL, y_train_FULL)

lda2 = LinearDiscriminantAnalysis()
lda2.fit(X_train_OR, y_train_OR)

lda3 = LinearDiscriminantAnalysis()
lda3.fit(X_train_RESAMP, y_train_RESAMP)

lda4 = LinearDiscriminantAnalysis()
lda4.fit(X_train_RESAMP_OR, y_train_RESAMP_OR)
```

Out[87]:
```
LinearDiscriminantAnalysis()
```

In [88]:
```
lda_preds1 = lda1.predict(X_val_FULL)
lda_preds2 = lda2.predict(X_val_OR)
lda_preds3 = lda3.predict(X_val_RESAMP)
lda_preds4 = lda4.predict(X_val_RESAMP_OR)
```

In [89]:
```
print('lda_preds1')
print(classification_report(lda_preds1,y_val_FULL))
print('\n')
print('lda_preds2')
print(classification_report(lda_preds2,y_val_OR))
print('\n')
print('lda_preds3')
print(classification_report(lda_preds3,y_val_RESAMP))
print('\n')
print('lda_preds4')
print(classification_report(lda_preds4,y_val_RESAMP_OR))
print('\n')
```

```
lda_preds1
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     22768
           1       0.48      0.35      0.40      1038

    accuracy                           0.96     23806
   macro avg       0.73      0.66      0.69     23806
weighted avg       0.95      0.96      0.95     23806


lda_preds2
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     16907
           1       0.46      0.15      0.22       419

    accuracy                           0.98     17326
   macro avg       0.72      0.57      0.61     17326
weighted avg       0.97      0.98      0.97     17326


lda_preds3
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      9888
           1       0.54      0.73      0.62       870

    accuracy                           0.93     10758
   macro avg       0.76      0.84      0.79     10758
weighted avg       0.94      0.93      0.93     10758


lda_preds4
              precision    recall  f1-score   support

           0       1.00      0.95      0.98      7502
           1       0.59      0.96      0.73       520

    accuracy                           0.95      8022
   macro avg       0.79      0.96      0.85      8022
weighted avg       0.97      0.95      0.96      8022
```

These results are not good, and show that perhaps the assumption that the covariance matrix for both classes is the same, is a bad assumption.

In [90]:
```
qda1 = QuadraticDiscriminantAnalysis()
qda1.fit(X_train_FULL, y_train_FULL)

qda2 = QuadraticDiscriminantAnalysis()
qda2.fit(X_train_OR, y_train_OR)

qda3 = QuadraticDiscriminantAnalysis()
qda3.fit(X_train_RESAMP, y_train_RESAMP)

qda4 = QuadraticDiscriminantAnalysis()
qda4.fit(X_train_RESAMP_OR, y_train_RESAMP_OR)
```

```
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are col
linear
  warnings.warn("Variables are collinear")
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are col
linear
  warnings.warn("Variables are collinear")
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are col
linear
  warnings.warn("Variables are collinear")
C:\Users\gogar\anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:808: UserWarning: Variables are col
linear
  warnings.warn("Variables are collinear")
```

Out[90]:
```
QuadraticDiscriminantAnalysis()
```

In [91]:
```
qda_preds1 = qda1.predict(X_val_FULL)
qda_preds2 = qda2.predict(X_val_OR)
qda_preds3 = qda3.predict(X_val_RESAMP)
qda_preds4 = qda4.predict(X_val_RESAMP_OR)
```

In [92]:
```
print('qda_preds1')
print(classification_report(qda_preds1,y_val_FULL))
print('\n')
print('qda_preds2')
print(classification_report(qda_preds2,y_val_OR))
print('\n')
print('qda_prebds3')
print(classification_report(qda_preds3,y_val_RESAMP))
print('\n')
print('qda_preds4')
print(classification_report(qda_preds4,y_val_RESAMP_OR))
print('\n')
```

```
qda_preds1
              precision    recall  f1-score   support

           0       0.02      0.99      0.03       386
           1       0.99      0.03      0.06     23420

    accuracy                           0.05     23806
   macro avg       0.51      0.51      0.05     23806
weighted avg       0.98      0.05      0.06     23806


qda_preds2
              precision    recall  f1-score   support

           0       0.02      0.99      0.04       312
           1       0.98      0.01      0.02     17014

    accuracy                           0.03     17326
   macro avg       0.50      0.50      0.03     17326
weighted avg       0.96      0.03      0.02     17326


qda_preds3
              precision    recall  f1-score   support

           0       0.18      0.97      0.31      1812
           1       0.95      0.12      0.22      8946

    accuracy                           0.27     10758
   macro avg       0.57      0.55      0.26     10758
weighted avg       0.82      0.27      0.24     10758


qda_preds4
              precision    recall  f1-score   support

           0       0.35      0.98      0.52      2551
           1       0.95      0.15      0.25      5471

    accuracy                           0.41      8022
   macro avg       0.65      0.57      0.38      8022
weighted avg       0.76      0.41      0.34      8022
```

The collinearity has caused a lot of problems with the predictions. Two options would be:

- removing variables which we think are collinear (did this before).
- use variables from Lasso method.

In [65]:
```
qda1 = QuadraticDiscriminantAnalysis()
qda1.fit(X_train_FULL[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']], y_train_FULL)

qda2 = QuadraticDiscriminantAnalysis()
qda2.fit(X_train_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']], y_train_OR)

qda3 = QuadraticDiscriminantAnalysis()
qda3.fit(X_train_RESAMP[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']], y_train_RESAMP)

qda4 = QuadraticDiscriminantAnalysis()
qda4.fit(X_train_RESAMP_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']], y_train_RESAMP_OR)
```

Out[65]:
```
QuadraticDiscriminantAnalysis()
```

In [66]:
```
qda_preds1 = qda1.predict(X_val_FULL[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']])
qda_preds2 = qda2.predict(X_val_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']])
qda_preds3 = qda3.predict(X_val_RESAMP[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']])
qda_preds4 = qda4.predict(X_val_RESAMP_OR[['VOL_DS','VOL_DEP','BETRAG_EINGANG_3M']])
```

In [68]:
```
print('qda_preds1')
print(classification_report(qda_preds1,y_val_FULL))
print('\n')
print('qda_preds2')
print(classification_report(qda_preds2,y_val_OR))
print('\n')
print('qda_preds3')
print(classification_report(qda_preds3,y_val_RESAMP))
print('\n')
print('qda_preds4')
print(classification_report(qda_preds4,y_val_RESAMP_OR))
print('\n')
```

```
qda_preds1
              precision    recall  f1-score   support

           0       0.99      0.97      0.98     23378
           1       0.18      0.31      0.23       428

    accuracy                           0.96     23806
   macro avg       0.58      0.64      0.60     23806
weighted avg       0.97      0.96      0.97     23806


qda_preds2
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     17117
           1       0.26      0.17      0.20       209

    accuracy                           0.98     17326
   macro avg       0.63      0.58      0.60     17326
weighted avg       0.98      0.98      0.98     17326


qda_preds3
              precision    recall  f1-score   support

           0       0.98      0.91      0.94     10393
           1       0.18      0.58      0.28       365

    accuracy                           0.90     10758
   macro avg       0.58      0.74      0.61     10758
weighted avg       0.96      0.90      0.92     10758


qda_preds4
              precision    recall  f1-score   support

           0       0.99      0.92      0.95      7743
           1       0.23      0.68      0.34       279

    accuracy                           0.91      8022
   macro avg       0.61      0.80      0.65      8022
weighted avg       0.96      0.91      0.93      8022
```

Results are not as good as when we manually selected the variables to use.

In [60]:
```
qda5 = QuadraticDiscriminantAnalysis()
qda5.fit(X_train_FULL[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN_WEB_3M','LOGIN_M

qda6 = QuadraticDiscriminantAnalysis()
qda6.fit(X_train_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN_WEB_3M','LOGIN_MOB

qda7 = QuadraticDiscriminantAnalysis()
qda7.fit(X_train_RESAMP[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN_WEB_3M','LOGIN

qda8 = QuadraticDiscriminantAnalysis()
qda8.fit(X_train_RESAMP_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN_WEB_3M','LO
```

Out[60]:
```
QuadraticDiscriminantAnalysis()
```

In [61]:
```
qda_preds5 = qda5.predict(X_val_FULL[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN_W
qda_preds6 = qda6.predict(X_val_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN_WEB
qda_preds7 = qda7.predict(X_val_RESAMP[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LOGIN
qda_preds8 = qda8.predict(X_val_RESAMP_OR[['VOL_DS','ANZ_DEP','VOL_DEP','KSEIT','ALTER','BETRAG_EINGANG_3M','LO
```

In [63]:
```
print('qda_preds5')
print(classification_report(qda_preds5,y_val_FULL))
print('\n')
print('qda_preds6')
print(classification_report(qda_preds6,y_val_OR))
print('\n')
print('qda_preds7')
print(classification_report(qda_preds7,y_val_RESAMP))
print('\n')
print('qda_preds8')
print(classification_report(qda_preds8,y_val_RESAMP_OR))
print('\n')
```

```
qda_preds5
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     22775
           1       0.29      0.21      0.24      1031

    accuracy                           0.94     23806
   macro avg       0.63      0.59      0.61     23806
weighted avg       0.94      0.94      0.94     23806


qda_preds6
              precision    recall  f1-score   support

           0       0.94      1.00      0.97     16171
           1       0.56      0.06      0.12      1155

    accuracy                           0.93     17326
   macro avg       0.75      0.53      0.54     17326
weighted avg       0.91      0.93      0.91     17326


qda_preds7
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      9935
           1       0.34      0.48      0.40       823

    accuracy                           0.89     10758
   macro avg       0.65      0.70      0.67     10758
weighted avg       0.91      0.89      0.90     10758


qda_preds8
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      7198
           1       0.42      0.43      0.43       824

    accuracy                           0.88      8022
   macro avg       0.68      0.68      0.68      8022
weighted avg       0.88      0.88      0.88      8022
```

Usually the next point would be to use the test set, but the models here have been run so differently that it is not very sensible to compare them to test set predictions.

Rather, comparisons should be made on the classification reports (explanations in the PDF), and a decision made based on a bias-variance trade off.

My own opinion would be that a random-forest style classifier is likely a good choice for this data, and with more tuning, it could provide very good and generiseable results. The downside of this is that a random forest is a black box method.