Configuration Manual

MSc Research Project
Data Analytics

Merve Baskan
Student ID: 20238096

School of Computing
National College of Ireland

Supervisor: Paul Stynes, Musfira Jilani and Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Merve Baskan
Student ID:	20238096
Programme:	Data Analytics
Year:	2021-22
Module:	MSc Research Project
Supervisor:	Paul Stynes, Musfira Jilani and Pramod Pathak
Submission Due Date:	19/09/2022
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	19th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: A Machine Learning Framework to Address Customer Churn Problem Using Uplift Modelling and Prescriptive Analysis

Merve Baskan
20238096

19/09/2022

1 Introduction

This document provides an overview of all the processes followed in the research project. The performance of each machine learning model is assessed, and the best model is chosen. The tools, approaches, and libraries utilized are explained in the following sections of this document. This research aims to compare the uplift model and the conventional customer churn prediction model. The ability to target the right customer group is evaluated for both models.

As stated in the research report, Experiment 1 is a replica of state-of-the-art research and the implementation is provided in open source by the author. Experiment 2 and Experiment 3 were explained in this document. [1]

2 Hardware and Software Specifications

Software Specifications: The Integrated Development Environment (IDE) used for the implementation of this research is Google Colaboratory and the programming language used is Python (v.3.7.13). The main libraries that utilized are:Matplotlib (v.3.2.2), Pandas (v.1.3.5), Xgboost (v.0.90), Seaborn (v.0.11.2) and Sci-kit learn (v.1.0.2)

Hardware specifications: ASUS ZenBook UM425UA-AM164T, Storage: 512GB M.2 NVMe™ PCIe® 3.0 SSD, RAM: 8.0 GB, Processor: AMD Ryzen™ 5 5500U Mobile Processor (6-core/12-thread, 11MB cache, up to 4.0 GHz max boost), Operating System: Windows 10 Home.

3 Data Preprocessing

Figure 1 shows the Python libraries and packages required for the project. Since the code block belongs to Experiment 2.2, it contains the XGBoost package. "**from** sk-

[1] website: `https://www.kaggle.com/code/davinwijaya/why-you-should-start-using-uplift-modeling/notebook`

learn.linear_model **import** LogisticRegression" is used in Experiment 3.1 and 3.2, which includes Logistic Regression application.

```
# Importing the  packages and libraries
import matplotlib as mpl, matplotlib.pyplot as plt, \
pandas as pd, seaborn as sns, xgboost as xgb, sklearn as sk
from sklearn.metrics import accuracy_score, \
confusion_matrix, multilabel_confusion_matrix
from sklearn.model_selection import train_test_split
```

Figure 1: Python libraries and packages

In Figure 2, 21572 null values in "reamining contract", 381 null values in "download avg" and "upload avg" can be seen. Columns that will not be used for analysis and 381 null values are cleaned. "reamining contract" means that the customer has never preferred the contract. Therefore null values can be filled with 0. Also, a new column "has contract" is created to show whether the customer has already selected the contract or not (0 or 1). The new dataset consists of 71893 non-null rows and 11 columns.(Figure 3)

```
[ ] df.isna().sum()

id                             0
is_tv_subscriber               0
is_movie_package_subscriber    0
subscription_age               0
bill_avg                       0
reamining_contract         21572
service_failure_count          0
download_avg                 381
upload_avg                   381
download_over_limit            0
churn                          0
dtype: int64
```

```
[ ] import numpy as np
df['download_avg'].replace('', np.nan, inplace=True)
df['upload_avg'].replace('',np.nan, inplace=True)
df.dropna(subset=['download_avg'], inplace=True)
df.dropna(subset=['upload_avg'], inplace=True)
```

```
test_cols = df.columns.tolist()
test_cols.insert(5, 'has_contract')

# Creating is_contract column
df['has_contract'] = df['reamining_contract'].apply(lambda x: 0 if pd.isna(x) else 1)
# Imputing null values with 0
df['reamining_contract'].replace(np.nan, 0, inplace=True)
# Rearranging columns
test_prepared = df[test_cols]
column_names = ['is_tv_subscriber','is_movie_package_subscriber','subscription_age','bill_avg','reamining_contract',
                'has_contract','service_failure_count','download_avg','upload_avg','download_over_limit','churn']

df = df.reindex(columns=column_names)
```

Figure 2: Data Cleaning and Feature Engineering

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 71893 entries, 0 to 72273
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   is_tv_subscriber             71893 non-null  int64
 1   is_movie_package_subscriber  71893 non-null  int64
 2   subscription_age             71893 non-null  float64
 3   bill_avg                     71893 non-null  int64
 4   reamining_contract           71893 non-null  float64
 5   has_contract                 71893 non-null  int64
 6   service_failure_count        71893 non-null  int64
 7   download_avg                 71893 non-null  float64
 8   upload_avg                   71893 non-null  float64
 9   download_over_limit          71893 non-null  int64
 10  churn                        71893 non-null  int64
dtypes: float64(4), int64(7)
memory usage: 6.6 MB
```

Figure 3: Processed Data Before Uplift Model Applications

As a requirement of the uplift model, there should be a control group and a treatment group. In the dataset in this research, churn(1) and not churn(0) already serve as control groups. However, the treatment group should be selected. The treatment group to be selected should also provide a binary classification as yes=1, no=0. As seen in Figure 4, the first selected treatment column is ”has contract” and the second is ”is tv subscriber”. These two treatments were used in experiments independently of each other, and treatment correlations(%) were compared in the research report. Figure 5

```
[ ] def correlation_treatment(df:pd.DataFrame):
        """Function to calculate the treatment's correlation
        """
        correlation = df[['treatment','churn']].corr(method ='pearson')
        print(correlation)
        return(pd.DataFrame(round(correlation.loc['churn'] * 100,2)))
```

Modification in addition to previous uploaded code: Pearson correlation function from Pandas is used. The results that are in the range are provided in the matrix. Then treatment correlation score as a percentage is provided.

```
[ ] print("Treatment correlation in dataset 1 (%):", correlation_treatment(df).iloc[0,0])

                treatment     churn
    treatment    1.000000 -0.472771
    churn       -0.472771  1.000000
    Treatment correlation in dataset 1 (%): -47.28
```

Figure 4: Treatment identification and treatment correlation of ”has contract” **(The percentage print code was revised after the first submission**

```
def correlation_treatment(df:pd.DataFrame):
    """Function to calculate the treatment's correlation
    """
    correlation = df[['treatment','churn']].corr(method ='pearson')
    print(correlation)
    return(pd.DataFrame(round(correlation.loc['churn'] * 100,2)))
```

Modification in addition to previous uploadded code: Pearson correlation function from Pandas is used. The results that are in the range are provided in the matrix. Then treatment correlation score as a percentage is provided.

```
[ ] print("Treatment correlation :", correlation_treatment(df).iloc[0,0])

                treatment     churn
    treatment    1.000000 -0.329417
    churn       -0.329417  1.000000
    Treatment correlation in dataset 1: -32.94
```

Figure 5: Treatment identification and treatment correlation of ”is tv subscriber” **(The percentage print code was revised after the first submission**

After determining the control and treatment groups, the next step is to determine the 4 target classes. In the research report, it is explained how the target classes are determined.(Figure 6)

```
[ ] def declare_target_class(df:pd.DataFrame):
        """Function for declare the target class
        """
        #CN:
        df['target_class'] = 0
        #CR:
        df.loc[(df.treatment == 0) & (df.churn == 0),'target_class'] = 1
        #TN:
        df.loc[(df.treatment == 1) & (df.churn == 1),'target_class'] = 2
        #TR:
        df.loc[(df.treatment == 1) & (df.churn == 0),'target_class'] = 3
        return df
```

Figure 6: Identification of target classes

4 Machine Learning

Figure 7 shows that 30% of the data used for testing, and the remaining 70% of the data for training. XGBoost training-test steps for both the uplift model and the conventional churn model are shown. "prediction_results" output can be seen in Figure 11.
In the conventional churn model, **'prediction_churn'** is used for accuracy. However, **'proba_churn'** is used to calculate the uplift for prescriptive analysis. For the Uplift modeling prediction analysis, 4 different confusion matrixes are created for 4 different target classes. Also, while XGBoost is used for Experiment 2, Logistic Regression is used for Experiment 3. (Figure 8)

```
def split_data(df:pd.DataFrame):
    """Split data into training data and testing data
    """
    X = df.drop(['churn','target_class'],axis=1)
    y = df.churn
    z = df.target_class
    X_train, X_test, \
    y_train, y_test, \
    z_train, z_test = train_test_split(X,
                                       y,
                                       z,
                                       test_size=0.3,
                                       random_state=42)

    return X_train,X_test, y_train, y_test, z_train, z_test


def machine_learning(X_train:pd.DataFrame,
                     X_test:pd.DataFrame,
                     y_train:pd.DataFrame,
                     y_test:pd.DataFrame,
                     z_train:pd.DataFrame,
                     z_test:pd.DataFrame):
    """Machine learning process consists of
    data training, and data testing process (i.e. prediction) with XGBoost (XGB) Algorithm
    """
    # prepare a new DataFrame
    prediction_results = pd.DataFrame(X_test).copy()
```

```
# training of the conventional churn model
model_tp \
= xgb.XGBClassifier().fit(X_train.drop('treatment', axis=1), y_train)
# prediction steps of the conventional churn model
prediction_tp \
= model_tp.predict(X_test.drop('treatment',axis=1))
probability__tp \
= model_tp.predict_proba(X_test.drop('treatment', axis=1))
prediction_results['prediction_churn'] = prediction_tp
prediction_results['proba_churn'] = probability__tp[:,1]


# training of the uplifted churn model
model_etu \
= xgb.XGBClassifier().fit(X_train.drop('treatment', axis=1), z_train)
# prediction steps of the uplifted churn model
prediction_etu \
= model_etu.predict(X_test.drop('treatment', axis=1))
probability__etu \
= model_etu.predict_proba(X_test.drop('treatment', axis=1))
prediction_results['prediction_target_class'] = prediction_etu
prediction_results['proba_CN'] = probability__etu[:,0]
prediction_results['proba_CR'] = probability__etu[:,1]
prediction_results['proba_TN'] = probability__etu[:,2]
prediction_results['proba_TR'] = probability__etu[:,3]
```

Figure 7: Machine Learning using XGBoost

```
def split_data(df:pd.DataFrame):
    """Split data into training data and testing data
    """
    X = df.drop(['churn','target_class'],axis=1)
    y = df.churn
    z = df.target_class
    X_train, X_test, \
    y_train, y_test, \
    z_train, z_test = train_test_split(X,
                                       y,
                                       z,
                                       test_size=0.3,
                                       random_state=42,
                                       stratify=df['treatment'])

    return X_train, X_test, y_train, y_test, z_train, z_test


def machine_learning(X_train:pd.DataFrame,
                     X_test:pd.DataFrame,
                     y_train:pd.DataFrame,
                     y_test:pd.DataFrame,
                     z_train:pd.DataFrame,
                     z_test:pd.DataFrame):
    """Machine learning process consists of
    data training, and data testing process (i.e. prediction) with Logistic Regression Algorithm
    """
    # prepare a new DataFrame
    prediction_results = pd.DataFrame(X_test).copy()

     #training of the conventional churn model
    model_tp \
    = LogisticRegression(max_iter=1000).fit(X_train.drop('treatment', axis=1), y_train)
    # prediction steps of the conventional churn model
```

```
 #training of the conventional churn model
model_tp \
= LogisticRegression(max_iter=1000).fit(X_train.drop('treatment', axis=1), y_train)
# prediction steps of the conventional churn model
prediction_tp \
= model_tp.predict(X_test.drop('treatment',axis=1))
probability__tp \
= model_tp.predict_proba(X_test.drop('treatment', axis=1))
prediction_results['prediction_churn'] = prediction_tp
prediction_results['proba_churn'] = probability__tp[:,1]


# training of the uplift model
model_etu \
= LogisticRegression(max_iter=10000).fit(X_train.drop('treatment', axis=1), z_train)
# prediction Process for ETU model
prediction_etu \
= model_etu.predict(X_test.drop('treatment', axis=1))
probability__etu \
= model_etu.predict_proba(X_test.drop('treatment', axis=1))
prediction_results['prediction_target_class'] = prediction_etu
prediction_results['proba_CN'] = probability__etu[:,0]
prediction_results['proba_CR'] = probability__etu[:,1]
prediction_results['proba_TN'] = probability__etu[:,2]
prediction_results['proba_TR'] = probability__etu[:,3]
prediction_results['score_etu'] = prediction_results.eval('\
proba_CN/(proba_CN+proba_CR) \
+ proba_TR/(proba_TN+proba_TR) \
- proba_TN/(proba_TN+proba_TR) \
- proba_CR/(proba_CN+proba_CR)')
```

Figure 8: Machine Learning using Logistic Regression **(revised after first submission)**

At first, logistic regression models do not converge, therefore, the maximum number of iterations for logistic regression are increased to solve this problem (1000 max. iterations for the conventional churn model, and 10000 max. iterations for the uplift model).Figure 8

After the iteration change, model fitting is provided. Also, "stratify" parameter is added in order to avoid bias. One of the disadvantages of logistic regression emerges as the dataset is linearly separable. As seen in Figure 12, in the application performed with treatment 1, "CN" and "CR" probabilities are close to 0, while "TN" and "TR" has values close to 1. Although using "stratify" mitigates this condition, negative uplift scores still occur.

```
    prediction_results['score_etu'] = prediction_results.eval('\
    proba_CN/(proba_CN+proba_CR) \
    + proba_TR/(proba_TN+proba_TR) \
    - proba_TN/(proba_TN+proba_TR) \
    - proba_CR/(proba_CN+proba_CR)')


    # add the churn and target class into dataframe as validation data
    prediction_results['churn'] = y_test
    prediction_results['target_class'] = z_test
    return prediction_results, model_etu.feature_importances_, model_tp.feature_importances_


def predict(df:pd.DataFrame):
    """Combining data split and machine learning process with XGB
    """
    X_train, X_test, y_train, y_test, z_train, z_test = split_data(df)
    prediction_results, feat_import_etu, feat_import_tp = machine_learning(X_train,
                                                                          X_test,
                                                                          y_train,
                                                                          y_test,
                                                                          z_train,
                                                                          z_test)
    print("Prediction succeeded")
    return prediction_results, feat_import_etu, feat_import_tp
```

Figure 9: The rest of machine learning process and uplift score calculation for uplift model

```
# plot feature importance manually
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot

# feature importance
print(feat_import_tp)
# plot
pyplot.bar(range(len(feat_import_tp)), feat_import_tp)
pyplot.show()

[0.00219546 0.01518508 0.01607754 0.7402843  0.05948396 0.00524637
 0.11764301 0.03344858 0.01043561]
```

```
# plot feature importance manually
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot

# feature importance
print(feat_import_etu)
# plot
pyplot.bar(range(len(feat_import_etu)), feat_import_etu)
pyplot.show()

[0.12755027 0.02027129 0.0309301  0.60202736 0.07955728 0.0011285
 0.10603139 0.0274068  0.00509698]
```

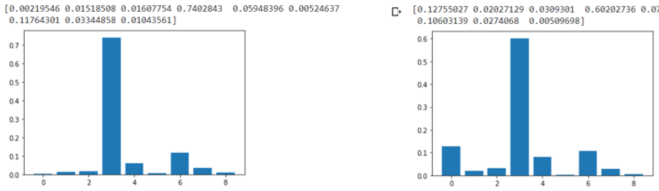

Figure 10: Feature Importance Plots

In Figure 9, the final part of the machine learning code block is presented and uplift score is calculated for the uplift model using Lai's generalized weighed uplift method (LGWUM). In Figure 10, feature importance plots for conventional churn prediction model and uplift model are shown when XGboost is used. Although this step is not used in research, this step will be important when a similar study is done with a dataset with many attributes.

5 Evaluation

Figure 11 is shown to explain the results more clearly: The uplift score calculated from the target class prediction probabilities for uplift model and the churn probability is used

```
[ ] prediction_results
```

ervice_failure_count	download_avg	upload_avg	download_over_limit	prediction_churn	proba_churn	prediction_target_class	proba_CN	proba_CR	proba_TN	proba_TR	score_etu	churn	target_class
0	106.3	14.2	0	0	0.055426	3	0.000956	0.001095	0.068535	0.929414	0.794526	0	3
0	23.8	7.6	0	0	0.056250	3	0.014565	0.021434	0.046430	0.917572	0.712861	0	1
0	5.0	0.3	0	1	0.959713	2	0.423940	0.013739	0.534654	0.027667	0.035619	1	0
0	31.0	4.9	0	1	0.983340	2	0.137084	0.006705	0.837242	0.018969	-0.048946	1	2

Figure 11: Prediction Results

```
[ ] prediction_results
```

ice_failure_count	download_avg	upload_avg	download_over_limit	prediction_churn	proba_churn	prediction_target_class	proba_CN	proba_CR	proba_TN	proba_TR	score_etu	churn	target_class
0	106.3	14.2	0	0	0.005623	3	1.459282e-23	5.266617e-17	0.007226	0.992774	-0.014451	0	3
0	23.8	7.6	0	0	0.041279	3	2.308331e-22	6.057866e-16	0.056961	0.943039	-0.113920	0	3
0	5.0	0.3	0	1	0.982915	2	4.580691e-01	8.886593e-03	0.524809	0.008235	-0.007164	1	0
0	31.0	4.9	0	1	0.885177	2	2.485795e-01	1.074265e-02	0.633962	0.106716	0.205305	1	2
2	120.9	5.4	0	0	0.003905	3	2.874503e-	1.356699e-	0.004332	0.995668	-0.008663	1	2

Figure 12: Prediction Results- Logistic Regression (without stratify)

for the conventional model. **'churn'** and **'prediction_churn'** are used for conventional customer churn prediction accuracy. **'target_class'** and **'prediction_target_class'** are used for the uplift model accuracy. Therefore, while the conventional model predicts 2 outcomes, the uplift model predicts 4 outcomes.(Figure 13)

```
===================================
Conventional churn confusion matrix:
              Predicted True  Predicted False
Actual True             8893              526
Actual False             789            11360
-----------------------------------
Uplifted churn confusion matrix:
a. CN's confusion matrix:
              Predicted True  Predicted False
Actual True            16809             1094
Actual False            2055             1610
b. CR's confusion matrix:
              Predicted True  Predicted False
Actual True            21139               26
Actual False             304               99
c. TN's confusion matrix:
              Predicted True  Predicted False
Actual True            10598             2486
Actual False            1741             6743
d. TR's confusion matrix:
              Predicted True  Predicted False
Actual True            11616              936
Actual False             442             8574
===================================
```

```
[ ] def accuracy_evaluation(df:pd.DataFrame):
        """Accuracy evaluation
        """
        akurasi_cp = accuracy_score(df['churn'],
                                    df['prediction_churn'])
        print('Conventional churn model accuracy: %.2f%%' % (akurasi_cp * 100.0))


        akurasi_uplift = accuracy_score(df['target_class'],
                                        df['prediction_target_class'])
        print('Uplifted churn model accuracy: %.2f%%' % (akurasi_uplift * 100.0))

[ ] accuracy_evaluation(prediction_results)

    Conventional churn model accuracy: 87.79%
    Uplifted churn model accuracy: 70.88%
```

Figure 13: Confusion matrix and accuracy results

In Figure 14 function ranks the churn probabilities and uplift scores to plot the Qini curve, and the steps for obtaining the Qini curve, Qini coefficient are also shown. The Qini-Coefficient is defined as the difference between the area under the Uplift Curve and the area under the random curve. The calculation below is also included in the code block. *x= population with treatment, N= total number of customers, uplift(x) = Nx[(TR/T)-(CR/C)]* Calculating and adding the Qini value into dataframe in the code block includes the formula below:

$$qini\ coefficient = \sum_{n=0}^{N-1} uplift - random\ model\ curve$$

Finally, as seen in Figure 15 uplift model's curve illustrated as "UPLIFT" with red line and conventional customer churn model's curve illustrated as "CHURN" with blue

```python
def sorting_data(df:pd.DataFrame):
    """Function to sort data
    """
    # Set up new DataFrames for the models
    df_c = pd.DataFrame({'n':[], 'target_class':[]})
    df_u = df_c.copy()
    df_c['target_class'] = df['target_class']
    df_u['target_class'] = df['target_class']


    # Add quantiles
    df_c['n'] = df.proba_churn.rank(pct=True, ascending=False)
    df_u['n'] = df.score_etu.rank(pct=True, ascending=False)
    df_c['score'] = df['proba_churn']
    df_u['score'] = df['score_etu']


    # Ranking the data by deciles
    df_c = df_c.sort_values(by='n').reset_index(drop=True)
    df_u = df_u.sort_values(by='n').reset_index(drop=True)
    df_c['model'], df_u['model'] = 'CP', 'Uplift'
    return df_c, df_u
```

```python
def calculating_qini(df:pd.DataFrame):
    """Function to measure the Qini value
    """
    # Calculate the C, T, CR, and TR
    C, T = sum(df['target_class'] <= 1), sum(df['target_class'] >= 2)
    df['cr'] = 0
    df['tr'] = 0
    df.loc[df.target_class  == 1,'cr'] = 1
    df.loc[df.target_class  == 3,'tr'] = 1
    df['cr/c'] = df.cr.cumsum() / C
    df['tr/t'] = df.tr.cumsum() / T


    # Calculate & add the qini value into the Dataframe
    df['uplift'] = df['tr/t'] - df['cr/c']
    df['random'] = df['n'] * df['uplift'].iloc[-1]
    qini_coef= df['uplift'].sum(skipna = True) - df['random'].sum(skipna = True)

    # Print the Qini coefficient
    print('Qini coefficient = {} {}'.format(round(qini_coef, 2), '%'))

    # Add q0 into the Dataframe
    q0 = pd.DataFrame({'n':0, 'uplift':0, 'target_class': None}, index =[0])
    qini = pd.concat([q0, df]).reset_index(drop = True)
    return qini
```

Figure 14: Qini Curve Process and Qini coefficient Part 1

line.('deepskyblue'). The random model is indicated by the gray line and is considered the baseline for the evaluation section. Figure 16 shows the outputs of Qini curve plots, Qini coefficient results for XGBoost with treatment 1(Experiment 2.1) and treatment 2(Experiment 2.2).

```python
def merging_data(df_c:pd.DataFrame, df_u:pd.DataFrame):
    """Function to add the 'Model' column and merge the dataframe into one
    """
    df_u['model'] = 'UPLIFT'
    df_c['model'] = 'CHURN'
    df = pd.concat([df_u, df_c]).sort_values(by='n').reset_index(drop = True)
    return df


def plot_qini(df:pd.DataFrame):
    """Function to plot the qini curve
    """
    print('\nPlotting the qini curve...')

    # Define the data that will be plotted
    order = ['UPLIFT','CHURN']
    ax = sns.lineplot(x='n', y=df.uplift, hue='model', data=df,
                      style='model', palette=['red','deepskyblue'],
                      style_order=order, hue_order = order)
```

```python
    # Additional plot display settings
    handles, labels = ax.get_legend_handles_labels()
    plt.xlabel('Proportion targeted',fontsize=30)
    plt.ylabel('Uplift',fontsize=30)
    plt.subplots_adjust(right=1)
    plt.subplots_adjust(top=1)
    plt.legend(fontsize=30)
    ax.tick_params(labelsize=24)
    ax.legend(handles=handles[:], labels=labels[:])
    ax.plot([0,1], [0,df.loc[len(df) - 1,'uplift']],'--', color='grey')
    print('Successfully plot the qini curve')
    return ax


def evaluation_qini(prediction_results:pd.DataFrame):
    """Function to combine all qini evaluation processes
    """
    df_c, df_u = sorting_data(prediction_results)
    print('Conventional Model')
    qini_c = calculating_qini(df_c)
    print('\nUplifted model:')
    qini_u = calculating_qini(df_u)
    qini = merging_data(qini_c, qini_u)
    ax = plot_qini(qini)
    return ax, qini
```

Figure 15: Qini Curve Process and Qini coefficient Part 2

As explained, the logistic regression models do not converge without editing, however, results are stable, when the same steps as XGBoost are applied. Due to the large number of poorly fitting observations, there is a lack of convergence, that means the data does not fit the model properly. Therefore, the maximum number of iterations for logistic regression are increased.

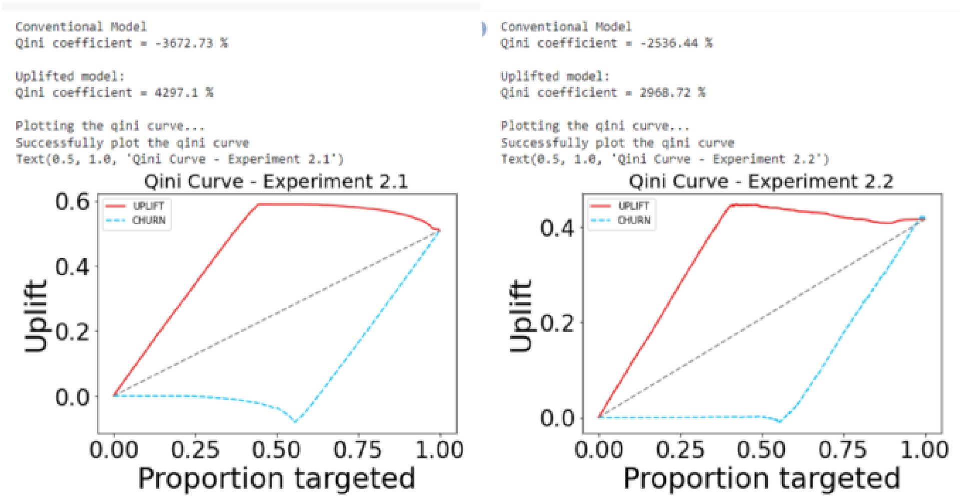

Figure 16: XGBoost - Qini Plot and Qini coeffficient Results

XGBoost successfully manages to deliver a positive uplift for the customer churn without fail in all experiments.(Figure 16) Even though the logistic regression models have a successful uplift and Qini curve with the same application, the models do not converge. The Qini curves and Qini coefficients of the logistic regression models before convergence are shown in Figure 17. The convergence problem in logistic regression is fixed and the effect of the stratify parameter on the result is examined. After adding the parameter, approximately 129% Qini coefficient increase is observed in experiment 3.2, therefore, it is used in the project.Figure 19

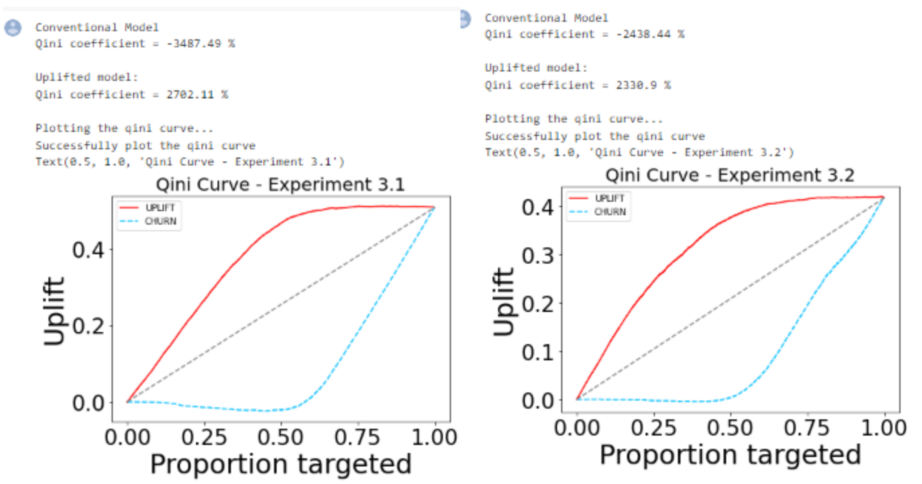

Figure 17: Logistic Regression- Qini Plot and Qini coeffficient Results**(before revision)**

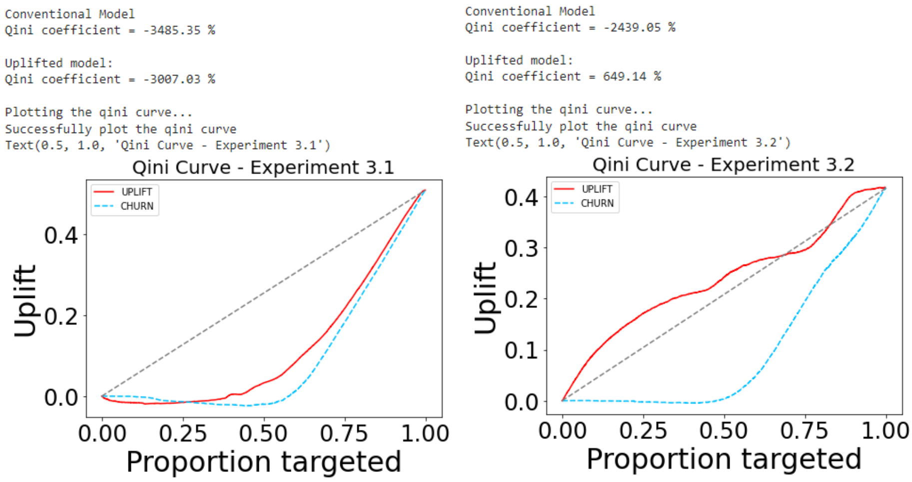

Figure 18: Logistic Regression- Qini Plot and Qini coeffficient Results **(after the maximum iteration increase)**

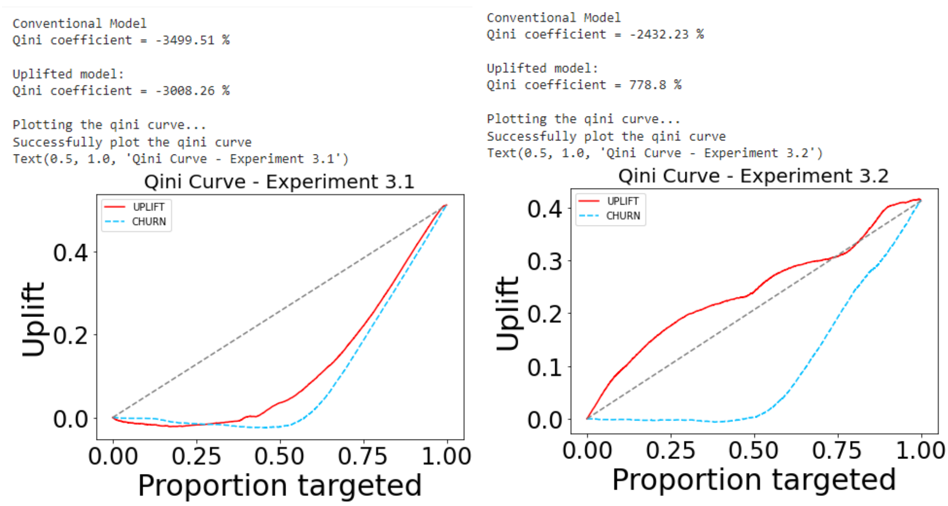

Figure 19: Logistic Regression- Qini Plot and Qini coeffficient Results **(after the maximum iteration increase and using stratify)**