Configuration Manual

MSc Research Project
Data Analytics

Prajwal Keshav Kongi
Student ID: 22205314

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Prajwal Keshav Kongi
Student ID:	22205314
Programme:	Data Analytics
Year:	2023-2024
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	10

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:	16th September 2024

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

Prajwal Keshav Kongi
22205314

1 Introduction

The configuration manual outlines the system specifications, environment setups, and methodologies used in implementing uplift modeling to target potential customers in the fashion e-commerce domain in Russia. The research was carried out on Jupyter Notebook using Python. New packages and libraries for Uplift modeling were installed like scikit-uplift. Data processing and Feature engineering are carried out in this project. Four uplift models are trained, tested, and evaluated when there are multiple treatments in the data. Two-Model with CatBoost, LGBM classifier, Class Transformation, and T-learner with LGBM models are applied. This document provides detailed descriptions and information on the tools and technologies used in developing uplift modeling.

2 System Specifications

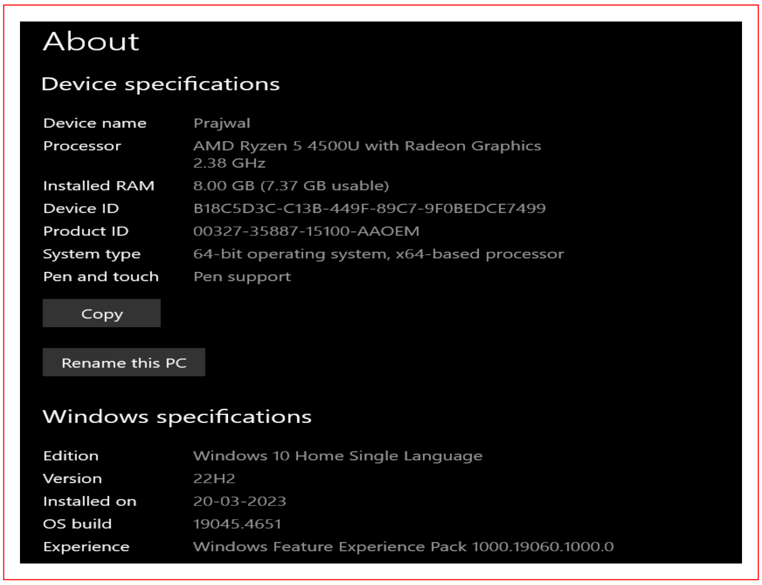

Figure 1: System Specifications

Figure 1 shows the details about the Device Specifications and Windows specifications used for this research.

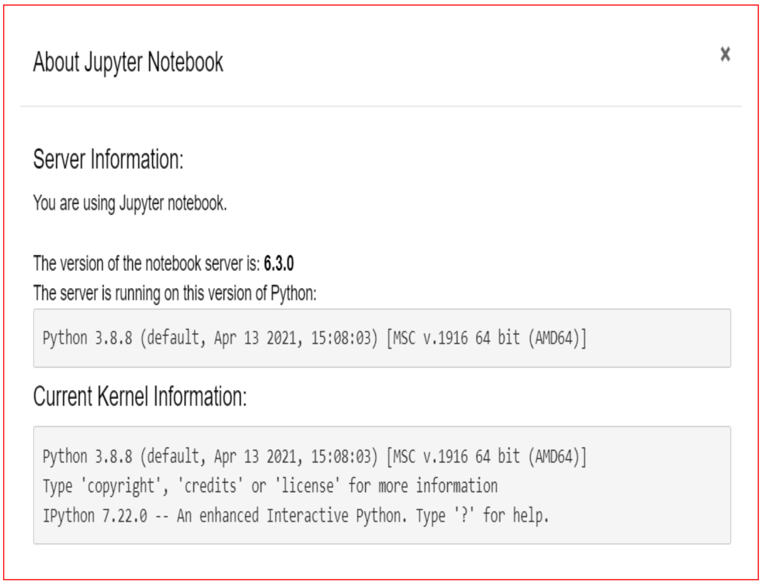

Figure 2: Jupyter and Python Versions

Figure 2 shows the details about the Jupyter Notebook and Python versions used in implementing this project.

3 Fashion E-Commerce Dataset

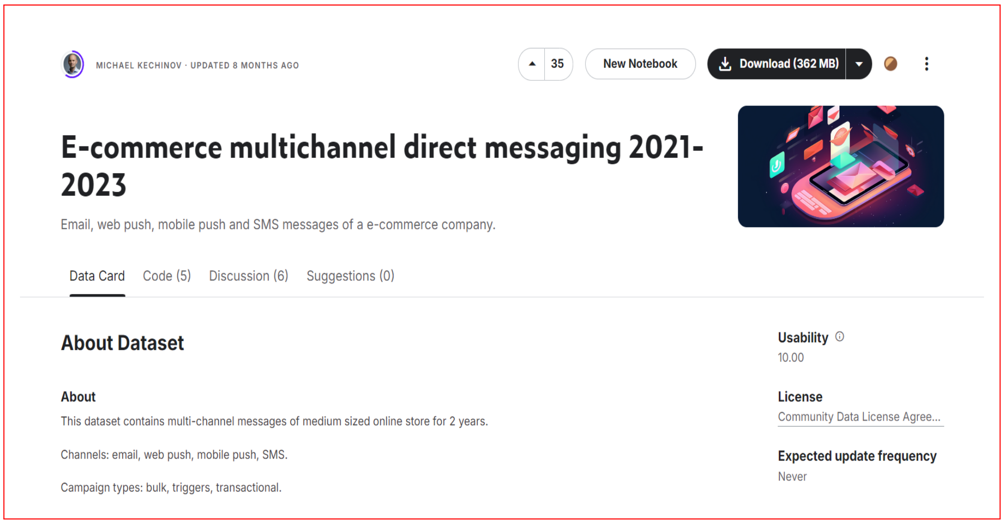

Figure 3: Fashion E-Commerce Direct Messaging Data

Figure 3 shows the source of the E-commerce data used in the project which is derived from Kaggle platform.

4 Dataset Preparation

In this section, data pre-processing is carried out for both message and campaign datasets.

Installing New Libraries

```
pip install pandas scikit-learn scikit-uplift xgboost
```

```
pip install --upgrade scikit-uplift
```

```
pip install catboost
```

```
!{sys.executable} -m pip install scikit-uplift dill lightgbm
```

Figure 4: Installation of New libraries and packages

In Figure 4, new libraries and packages are installed for this project for the implementation of uplift modeling.

Importing Libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from pymongo import MongoClient
import sqlalchemy as db

import sys
from sklearn.model_selection import train_test_split
from sklift.models import TwoModels
from sklift.models import SoloModel
from sklift.models import ClassTransformation
from sklearn.ensemble import RandomForestClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

from sklift.metrics import uplift_at_k, uplift_auc_score, qini_auc_score
from sklift.viz import plot_qini_curve
```

Figure 5: Libraries used in this project

Figure 5 shows the libraries used in this project.

```python
# Converting string to int

df['campaign_id'] = df['campaign_id'].astype('Int32')
```

```python
# Function to convert 't' or 'f' to boolean values (0 and 1)

def convert_to_bool(value):
    if value == 't':
        return True
    else:
        return False

convert_to_bool_cols = ['is_opened', 'is_clicked', 'is_unsubscribed', 'is_purchased']


df[convert_to_bool_cols] = df[convert_to_bool_cols].applymap(convert_to_bool)
```

```python
# Converting date columns to datetime format

date_cols = ['date', 'sent_at', 'opened_first_time_at', 'opened_last_time_at', 'clicked_first_time_at', 'clicked_last_time_a
             'unsubscribed_at', 'hard_bounced_at', 'soft_bounced_at', 'complained_at', 'blocked_at', 'purchased_at']

df[date_cols] = df[date_cols].apply(pd.to_datetime)

df[date_cols].head(2)

```

Figure 6: Fixing Datatypes

In Figure 6, the features are converted to appropriate datatypes using apply, and applymap functions.

```python
# Replacing missing values with Mode of each column

mode_cols = ['subject_with_personalization', 'subject_with_deadline', 'subject_with_emoji',
             'subject_with_bonuses', 'subject_with_discount', 'subject_with_saleout']

for column in mode_cols:
    mode_value = df3[column].mode()[0]  # Getting the most frequent value
    df3[column] = df3[column].fillna(mode_value)
```

```python
# Filling subject_length feature with mean

df3['subject_length'] = df3['subject_length'].fillna(df3['subject_length'].mean())

df3['total_count'] = df3['total_count'].fillna(df3['total_count'].mean())
```

Figure 7: Handling Missing Values

The handling of missing values for both Numerical and Categorical variables is shown in Figure 7.

```python
# Taking only top 7 email_providers and renaming remaining email_providers with 'others'

df['email_provider'] = df['email_provider'].apply(lambda x: x if x in ['mail.ru', 'gmail.com', 'yandex.ru', 'bk.ru',
                                                                        'list.ru', 'inbox.ru', 'rambler.ru'] else 'others')
```

Figure 8: Grouping Email Providers

Figure 8 shows the apply function used with lambda to keep only the top 7 email providers and group the remaining email providers as ’others’.

Pre-processing of Campaigns Data

```python
df_camp1 = df_camp.copy()

# Dropping null values from total_count feature
df_camp1 = df_camp1.dropna(subset=['total_count'])

# Removing duplicates
df_camp1 = df_camp1.drop_duplicates(subset=['id'])

# From ab_test column, we'll get to know which users were in treatment and control groups
treatment_campaign_ids = list(df_camp1[df_camp1['ab_test'] == True]['id'].unique())

# Dropping irrelevant columns
df_camp1.drop(columns = ['topic', 'started_at', 'finished_at', 'ab_test', 'hour_limit',
                         'is_test', 'position'], inplace=True, axis=1)

# Fixing datatypes
# Convert boolean columns to 0 and 1

camp_binary_cols = ['warmup_mode', 'subject_with_personalization', 'subject_with_deadline',
                    'subject_with_emoji', 'subject_with_bonuses', 'subject_with_discount', 'subject_with_saleout']

df_camp1[camp_binary_cols] = df_camp1[camp_binary_cols].apply(lambda x: x.astype(int))

# Merging Message and Campaigns Data
df3 = pd.merge(df3, df_camp1, left_on='campaign_id', right_on='id', how='left')

```

Figure 9: Processing of Campaigns Data

Figure 9 outlines the key pre-processing steps undertaken for campaign data.

5 Feature Engineering

```python
# Applying log transformation for total_count column

df_camp_log_transformed = df_camp4.copy()
df_camp_log_transformed['total_count'] = np.log1p(df_camp_log_transformed['total_count'])
```

```python
# Function to clip outliers based on the 5th and 95th percentile for subject_length

def cap_outliers(df, column):
    lower_cap = df[column].quantile(0.05)
    upper_cap = df[column].quantile(0.95)
    df[column] = df[column].clip(lower=lower_cap, upper=upper_cap)
    return df

df_camp_capped = cap_outliers(df_camp4, 'subject_length')
```

Figure 10: Treating Outliers

In Figure 10, functions to detect and treat outliers are shown. The methods used to treat outliers of variables are Log Transformation and Clipping outliers using percentiles.

```
# Creating new column with the difference in seconds between message 'sent_at' and 'opened_at', 'clicked_at', 'purchased_at'

df1['opened_seconds'] = (df1['opened_first_time_at'] - df1['sent_at']).dt.total_seconds()
df1['clicked_seconds'] = (df1['clicked_first_time_at'] - df1['sent_at']).dt.total_seconds()
df1['purchased_seconds'] = (df1['purchased_at'] - df1['sent_at']).dt.total_seconds()
```

```
# Replacing NaN values in these columns with 0

df1['opened_seconds'].fillna(0, inplace=True)
df1['clicked_seconds'].fillna(0, inplace=True)
df1['purchased_seconds'].fillna(0, inplace=True)
```

```
# Segregating the seconds column's data into 10 segments

num_bins = 10

df1['opened_recency'] = pd.cut(df1['opened_seconds'],
                               bins=num_bins, labels=list(range(num_bins-1, -1, -1)), include_lowest=True)

df1['clicked_recency'] = pd.cut(df1['clicked_seconds'],
                                bins=num_bins, labels=list(range(num_bins-1, -1, -1)), include_lowest=True)

df1['purchased_recency'] = pd.cut(df1['purchased_seconds'],
                                  bins=num_bins, labels=list(range(num_bins-1, -1, -1)), include_lowest=True)
```

```
# Manually filling 0 where message opened_at, clicked_at and purchased_at are null values

df1.loc[df['opened_first_time_at'].isna(), 'opened_recency'] = 0
df1.loc[df['clicked_first_time_at'].isna(), 'clicked_recency'] = 0
df1.loc[df['purchased_at'].isna(), 'purchased_recency'] = 0
```

Figure 11: Creating Recency Score

Figure 11 shows the steps taken in the creation of recency scores for 3 features ie, message opened, clicked, and purchased.

```
# One-hot Encoding

df3.drop(columns = ['warmup_mode', 'platform', 'id', 'campaign_type', 'channel_y'], inplace=True, axis=1)

df4 = df3.copy()
df4 = pd.get_dummies(df4, columns = ['channel_x', 'message_type', 'email_provider'], drop_first=True, dtype = int)
```

Figure 12: One-hot Encoding

One-hot encoding is implemented in Figure 12, to transform categorical features into numerical features using the 'get_dummies' function.

6 Modeling

In this research, four uplift models are trained, tested, and fine-tuned. The following contains the codes of four uplift models implemented on the fashion e-commerce dataset.

6.1 Two Model with CatBoost

Uplift Modeling

1. Two Models Approach

```python
from sklift.models import TwoModels
from xgboost import XGBClassifier
from sklift.metrics import uplift_at_k, qini_auc_score, uplift_auc_score
from sklearn.model_selection import GridSearchCV
```

```python
# Two model - an uplift model with CatBoost classifier

estimator_trmnt = CatBoostClassifier(silent=True, thread_count=2, random_state=42)
estimator_ctrl = CatBoostClassifier(silent=True, thread_count=2, random_state=42)

two_model = TwoModels(
    estimator_trmnt = estimator_trmnt,
    estimator_ctrl = estimator_ctrl,
    method='ddr_control'
)

# Fit and Predict the model
two_model.fit(X_train, y_train, t_train)
uplift_two_model = two_model.predict(X_test)
```

```python
# Evaluation

uplift_score = uplift_at_k(y_test, uplift_two_model, t_test, strategy='overall', k=0.3)
auuc = uplift_auc_score(y_test, uplift_two_model, t_test)
auqc = qini_auc_score(y_test, uplift_two_model, t_test)

print(f'Two-Model Approach - Uplift score at 30%: {uplift_score}')
print(f'Two-Model Approach - Area Under Uplift Curve (AUUC): {auuc}')
print(f'Two-Model Approach - Area Under Qini Curve (AUQC): {auqc}')
```

```
Two-Model Approach - Uplift score at 30%: 0.0015407564272344187
Two-Model Approach - Area Under Uplift Curve (AUUC): -0.0010886121763667507
Two-Model Approach - Area Under Qini Curve (AUQC): -0.001431795914293869
```

Figure 13: Two Model approach with CatBoost

Figure 13 shows the implementation of the Two-model approach using the CatBoost classifier.

6.2 Class Transformation

2. Class Transformation

```python
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklift.metrics import uplift_at_k
from sklift.models import ClassTransformation
```

```python
df4['target'] = df4['is_clicked']

X = df4.drop(columns=['campaign_id', 'is_clicked', 'target'])
y = df4['target']
treatment = df4['treatment']

# Stratify split
stratify_cols = pd.concat([treatment, y], axis=1)

X_train, X_val, trmnt_train, trmnt_val, y_train, y_val = train_test_split(X,
                                                                          treatment,
                                                                          y,
                                                                          stratify=stratify_cols,
                                                                          test_size=0.3,
                                                                          random_state=42
)

print(f"Train shape: {X_train.shape}")
print(f"Validation shape: {X_val.shape}")
```

```
Train shape: (700000, 26)
Validation shape: (300000, 26)
```

```python
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
estimator = CatBoostClassifier(verbose=100, random_state=42, thread_count=1)

# Class Transformation model
ct_model = ClassTransformation(estimator=estimator)

my_pipeline = Pipeline([('imputer', imp_mode), ('model', ct_model)
])

# Fit & predict the uplift
my_pipeline = my_pipeline.fit(X=X_train, y=y_train, model__treatment=trmnt_train)
uplift_predictions = my_pipeline.predict(X_val)

uplift_30 = uplift_at_k(y_val, uplift_predictions, trmnt_val, strategy='overall')
```

Figure 14: Class Transformation Model

In Figure 14, Class Transformation using the CatBoost Classifier is trained on the dataset Zhao and Harinen (2019). The pipeline is used in this code for smooth implementation. The data is split using stratify split method where treatment and control groups are equally distributed among the train and test datasets.

3. Two Model Approach with LGBM Classifier

```
X = df4.drop(columns=['is_clicked'])
y = df4['is_clicked']
treatment = df4['treatment']

# Stratify split
stratify_cols = pd.concat([treatment, y], axis=1)
X_train, X_val, trmnt_train, trmnt_val, y_train, y_val = train_test_split(X,
                                                                          treatment,
                                                                          y,
                                                                          stratify=stratify_cols,
                                                                          test_size=0.3,
                                                                          random_state=31)
print(f"Train shape: {X_train.shape}")
print(f"Validation shape: {X_val.shape}")
```

```
Train shape: (700000, 28)
Validation shape: (300000, 28)
```

```
# Initialize the models with regularization
treatment_model = LGBMClassifier(
                    random_state=31,
                    n_estimators=100,
                    learning_rate=0.05,
                    lambda_l1=1.0,  # L1 regularization
                    lambda_l2=1.0,  # L2 regularization
                    min_split_gain=0.01,
                    min_child_weight=1,
                    subsample=0.8,
                    colsample_bytree=0.8,
                    objective='binary')

control_model = LGBMClassifier(
                random_state=31,
                n_estimators=100,
                learning_rate=0.05,
                lambda_l1=1.0,
                lambda_l2=1.0,
                min_split_gain=0.01,
                min_child_weight=1,
                subsample=0.8,
                colsample_bytree=0.8,
                objective='binary')

# Initialize the TwoModels approach
tm = TwoModels(estimator_trmnt=treatment_model, estimator_ctrl=control_model, method='vanilla')

# Fit and Predict
tm = tm.fit(X_train, y_train, trmnt_train)
uplift_tm = tm.predict(X_val)
```

Figure 15: Two Model with LGBM

Figure 15 shows the code of Two model approach using LGBM Classifier. Regularization techniques and fine-tuning of the models for both treatment and control groups is carried out in this code.

4. T-Learner (Meta Learner) with LGBM Regressor

Treatment

```python
# Treated Units
df_treated = df4[df4['treatment'] == 1]

# Features
features_treated = df_treated.drop(columns=['campaign_id', 'is_clicked', 'treatment'], axis = 1)

# Target action
y_treated = df_treated.loc[:, ['is_clicked']]
```

Control

```python
# Control Units
df_control = df4[df4['treatment'] == 0]

# Features
features_control = df_control.drop(columns=['campaign_id', 'is_clicked', 'treatment'], axis = 1)

# Target action
y_control = df_control.loc[:, ['is_clicked']]
```

```python
# features for all the samples

features = df4.drop(columns=['campaign_id', 'is_clicked', 'treatment'], axis = 1)
```

```python
# LGBM Regressor

optimized_lgbm = LGBMRegressor(
    random_state=42,
    n_estimators=500,
    learning_rate=0.05,
    lambda_l1=1.0,           # L1 regularization
    lambda_l2=1.0,           # L2 regularization
    subsample=0.8,           # Subsample to prevent overfitting
    colsample_bytree=0.8,    # Feature sampling
    early_stopping_round=10  # Early stopping
)
```

```python
# Fit and Predict
t_treated = optimized_lgbm.fit(features_treated, y_treated, eval_set=[(features, df4['is_clicked'])])
t_control = optimized_lgbm.fit(features_control, y_control, eval_set=[(features, df4['is_clicked'])])
```

Figure 16: T-Learner with LGBM

In Figure 16, the code outlines the implementation of T-learner with LGBM on both treatment and control groups. The model is fine-tuned with various methods to overcome over-fitting.

7 Evaluation

In this research, three evaluation metrics are used to evaluate the uplift models. Uplift Score at 30%, AUUC and AUQC scores.

```python
# Evaluation

uplift_score = uplift_at_k(y_test, uplift_two_model, t_test, strategy='overall', k=0.3)
auuc = uplift_auc_score(y_test, uplift_two_model, t_test)
auqc = qini_auc_score(y_test, uplift_two_model, t_test)

print(f'Two-Model Approach - Uplift score at 30%: {uplift_score}')
print(f'Two-Model Approach - Area Under Uplift Curve (AUUC): {auuc}')
print(f'Two-Model Approach - Area Under Qini Curve (AUQC): {auqc}')


# Plotting Uplift Score at 30%
plt.figure(figsize=(6, 4))
plt.bar('Uplift Score at 30%', uplift_score, color='skyblue')
plt.xlabel('Evaluation Metric')
plt.ylabel('Score')
plt.title('Two Models: Uplift Score at 30%')
plt.ylim(0, uplift_score + 0.05)  # Adjust y-axis limit
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

```

Figure 17: Evaluation Metrics

Figure 17 shows the code for calculating the uplift score at 30%, AUUC, and AUQC scores. Additionally, the uplift score is plotted using the matplotlib library.

```
# Plot the Qini curve

plt.figure(figsize=(8, 6))
plot_qini_curve(y_val, uplift_tm, trmnt_val, perfect=True)
plt.title('Qini Curve')
plt.xlabel('Number of Targets (Percentile)')
plt.ylabel('Cumulative Gain')
plt.grid(True)
plt.show()
```

Figure 18: Qini Curve

Figure 18 shows the code to plot the Qini curve using 'sklift.viz' library and from 'plot_qini_curve' package.

References

Zhao, Z. and Harinen, T. (2019). Uplift modeling for multiple treatments with cost optimization, *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 422–431.