E-Commerce Customer Retention Analysis
Configuration Manual

MSc Research Project
Data Analytics

Mohan Sugumaran
Student ID: x22183779

School of Computing
National College of Ireland

Supervisor: Arjun Chikkankod

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Mohan Sugumaran

Student ID: X22183779

Programme: Data Analytics **Year:** 2024

Module: Research Project

Lecturer: Arjun Chikkankod

Submission Due Date: 12/08/2024

Project Title: E-commerce Customer Retention Analysis

Word Count: 973 **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: Mohan Sugumaran

Date: 12/08/2024

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

Mohan Sugumaran
Student ID: x22183779

1 Introduction

The aim of this research project is to precisely forecast customer attrition in the context of e-commerce. This handbook provides a detailed explanation of the hardware and software prerequisites required for future researchers to successfully reproduce the experiment. The handbook provides a comprehensive overview of every step involved in the project execution, encompassing data gathering, preprocessing, model training, testing, and assessment. When relevant, footnotes are included with reference code repositories, allowing for the replication and validation of all phases.

2 System Requirements

2.1 Hardware Requirements

The system used for this project is equipped with an 11th Gen Intel Core i7-1165G7 processor running at 2.80 GHz, with 16 GB of installed RAM, and operates on a 64-bit Windows operating system. This setup provides sufficient computational power to handle the data processing, model training, and evaluation tasks required for the customer churn prediction project.

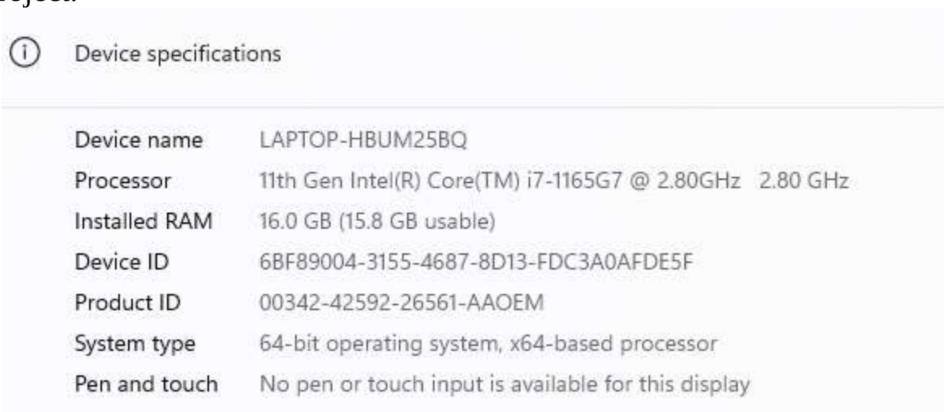


Figure 1 : Hardware Configuration

2.2 Software Requirements

The software configuration entails utilising Jupyter Notebook as the primary development environment for coding, data processing, and model training. The dataset was instantly uploaded and analysed within Jupyter Notebook, enabling a dynamic and streamlined workflow for the customer churn prediction project.

3 Importing required libraries

The essential Python libraries for this project are listed in the requirements.txt file, which guarantees the accurate installation of all dependencies. To recreate the environment, you may install the necessary libraries by using the command "pip install -r requirements.txt" in your terminal.

```
scikit-learn
numpy
pandas
matplotlib
seaborn
```

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc, accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from collections import Counter
from IPython.display import display, HTML
```

Figure 2: Required Python Libraries

4 Data Exploration

This figure represents one of the data exploration techniques used to gain insights and understand the data. Similarly, various exploratory methods were employed throughout this research.

Figure 3: Uni-Variate Analysis of Categorical Columns

---

https://www.kaggle.com/datasets/rakeshswaminathan/churn-data

5 Data Preparation

5.1 Data Collection

The dataset for the customer churn analysis was loaded into the Jupyter Notebook using the pd.read_excel function from the specified file path.

```
2]: # Loding data
churn  = pd.read_excel(r'C:/Users/mohan/Downloads/Research Project/dataset/Customer Churn Data.xlsx', sheet_name = 'Data for DSBA')
```

Figure 4: Loading Customer Churn Data

5.2 Data Preprocessing

Following the process of data cleaning, an exploratory data analysis (EDA) was conducted to examine the structure of the dataset and verify that all characteristics had consistent non-null values.

EDA after data cleaning

```
166]: # checking info of data
churn.info()

<class 'pandas.core.frame.DataFrame'>
Index: 11144 entries, 0 to 11259
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   AccountID                11144 non-null  int64
 1   Churn                    11144 non-null  int64
 2   Tenure                   11144 non-null  Int64
 3   City_Tier                11144 non-null  Int64
 4   CC_Contacted_LY          11144 non-null  Int64
 5   Payment                  11144 non-null  float64
 6   Gender                   11144 non-null  float64
 7   Service_Score            11144 non-null  float64
 8   Account_user_count       11144 non-null  float64
 9   account_segment          11144 non-null  float64
 10  CC_Agent_Score           11144 non-null  float64
 11  Marital_Status           11144 non-null  float64
 12  rev_per_month            11144 non-null  float64
 13  Complain_ly              11144 non-null  float64
 14  rev_growth_yoy           11144 non-null  float64
 15  coupon_used_for_payment  11144 non-null  float64
 16  Day_Since_CC_connect     11144 non-null  float64
 17  cashback                 11144 non-null  float64
 18  Login_device             11144 non-null  float64
dtypes: Int64(3), float64(14), int64(2)
```

Figure 5: EDA info

Preprocessed the 'Tenure' variable by handling missing and erroneous data, changing it to an integer data format, and imputing the missing values with the median value.

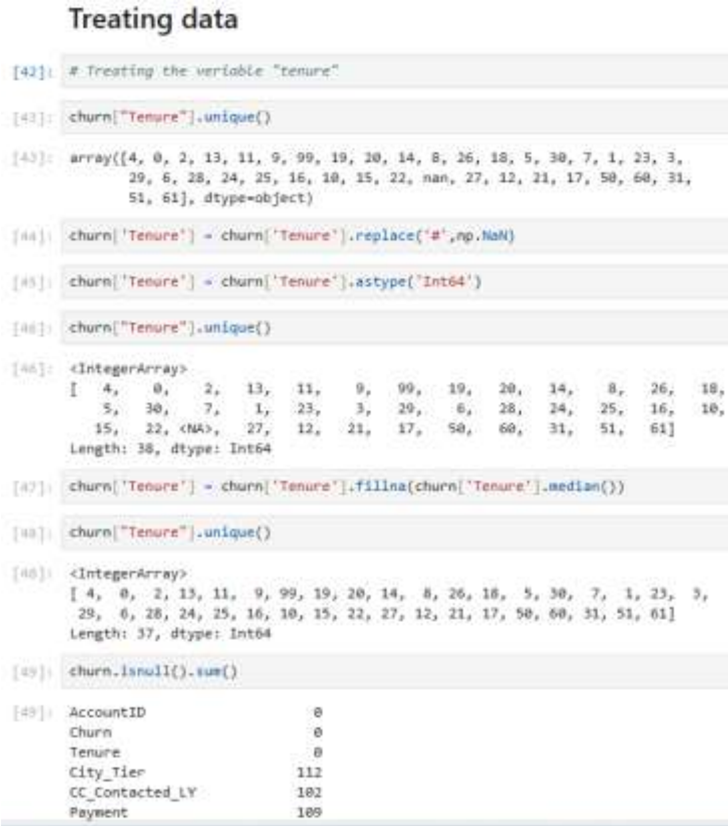
Scaling data

```
from sklearn.preprocessing import MinMaxScaler

churn['Scaled_Churn'] = MinMaxScaler().fit_transform(churn[['Churn']])
churn['Scaled_Tenure'] = MinMaxScaler().fit_transform(churn[['Tenure']])
churn['Scaled_City_Tier'] = MinMaxScaler().fit_transform(churn[['City_Tier']])
churn['Scaled_CC_Contacted_LY'] = MinMaxScaler().fit_transform(churn[['CC_Contacted_LY']])
churn['Scaled_Payment'] = MinMaxScaler().fit_transform(churn[['Payment']])
churn['Scaled_Gender'] = MinMaxScaler().fit_transform(churn[['Gender']])
churn['Scaled_Service_Score'] = MinMaxScaler().fit_transform(churn[['Service_Score']])
churn['Scaled_Account_user_count'] = MinMaxScaler().fit_transform(churn[['Account_user_count']])
churn['Scaled_account_segment'] = MinMaxScaler().fit_transform(churn[['account_segment']])
churn['Scaled_CC_Agent_Score'] = MinMaxScaler().fit_transform(churn[['CC_Agent_Score']])
churn['Scaled_Marital_Status'] = MinMaxScaler().fit_transform(churn[['Marital_Status']])
churn['Scaled_rev_per_month'] = MinMaxScaler().fit_transform(churn[['rev_per_month']])
churn['Scaled_Complain_ly'] = MinMaxScaler().fit_transform(churn[['Complain_ly']])
churn['Scaled_rev_growth_yoy'] = MinMaxScaler().fit_transform(churn[['rev_growth_yoy']])
churn['Scaled_coupon_used_for_payment'] = MinMaxScaler().fit_transform(churn[['coupon_used_for_payment']])
churn['Scaled_Day_Since_CC_connect'] = MinMaxScaler().fit_transform(churn[['Day_Since_CC_connect']])
churn['Scaled_cashback'] = MinMaxScaler().fit_transform(churn[['cashback']])
churn['Scaled_Login_device'] = MinMaxScaler().fit_transform(churn[['Login_device']])
```

Figure 6: Data Scaling with MinMaxScaler

The dataset underwent scaling using the MinMaxScaler technique to standardise features within the range of 0 and 1, guaranteeing consistency across all variables. The rescaled data was subsequently aggregated into a fresh DataFrame to facilitate additional analysis and model training.



```
Treating data

[42]: # Treating the variable "tenure"

[43]: churn["Tenure"].unique()

[43]: array([4, 0, 2, 13, 11, 9, 99, 19, 20, 14, 8, 26, 18, 5, 30, 7, 1, 23, 3,
       29, 6, 28, 24, 25, 16, 10, 15, 22, nan, 27, 12, 21, 17, 50, 60, 31,
       51, 61], dtype=object)

[44]: churn['Tenure'] = churn['Tenure'].replace('#',np.NaN)

[45]: churn['Tenure'] = churn['Tenure'].astype('Int64')

[46]: churn["Tenure"].unique()

[46]: <IntegerArray>
[   4,    0,    2,   13,   11,    9,   99,   19,   20,   14,    8,   26,   18,
    5,   30,    7,    1,   23,    3,   29,    6,   28,   24,   25,   16,   10,
   15,   22, <NA>,   27,   12,   21,   17,   50,   60,   31,   51,   61]
Length: 38, dtype: Int64

[47]: churn['Tenure'] = churn['Tenure'].fillna(churn['Tenure'].median())

[48]: churn["Tenure"].unique()

[48]: <IntegerArray>
[ 4,  0,  2, 13, 11,  9, 99, 19, 20, 14,  8, 26, 18,  5, 30,  7,  1, 23,  3,
 29,  6, 28, 24, 25, 16, 10, 15, 22, 27, 12, 21, 17, 50, 60, 31, 51, 61]
Length: 37, dtype: Int64

[49]: churn.isnull().sum()

[49]: AccountID            0
      Churn                0
      Tenure               0
      City_Tier          112
      CC_Contacted_LY    102
      Payment            109
```

Figure 7: Treating Data

5.3 Data Splitting

The dataset was split into training and testing sets, with 70% of the data used for training and 30% reserved for testing.

Figure 8: Splitting Data into Training and Testing Sets

6 Model Implementation

6.1 Overview of Models

The model was evaluated both with and without SMOTE, and the results were stored for comparison, this process was repeated for all other models as well.

```
[255]: # Store results for comparison
results_without_smote = []
results_with_smote = []

# Logistic Regression without SMOTE
logistic_regression_model = LogisticRegression(solver='liblinear')
acc, prec, rec, f1 = evaluate_model(logistic_regression_model, X_train, y_train, X_test, y_test, 'Logistic Regression', with_smote=False)
results_without_smote.append(['Logistic Regression', acc, prec, rec, f1])

# Logistic Regression with SMOTE
acc, prec, rec, f1 = evaluate_model(logistic_regression_model, X_train_res, y_train_res, X_test, y_test, 'Logistic Regression', with_smote=True)
results_with_smote.append(['Logistic Regression', acc, prec, rec, f1])
```

Figure 9: Comparing Model Performance with and without SMOTE

6.2 Hyperparameter Tuning

Hyperparameter grids were defined for several models, such as Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, AdaBoost, Bagging, and SVM, in order to enhance their performance throughout the process of model tuning.

Define Hyperparameter Grids ¶

```
param_grid_logistic = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga']
}

param_grid_decision_tree = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

param_grid_random_forest = {
    'n_estimators': [50, 100, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

param_grid_gradient_boosting = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9, 1.0]
}

param_grid_adaboost = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5, 1.0]
}

param_grid_bagging = {
    'n_estimators': [10, 50, 100],
    'max_samples': [0.5, 0.7, 1.0],
    'max_features': [0.5, 0.7, 1.0]
}

param_grid_svc = {
    'C': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto']
}
```

Figure 10 : Hyperparameter Grids for Model Tuning

6.3 Training Procedures

Below is the sample for the model training for the Random Forest model, with the parameter of 100 estimators to evaluate its performance without applying SMOTE. Similarly, this is performed for all the other models.

```
# Random Forest without SMOTE
random_forest_model = RandomForestClassifier(n_estimators=100)
```

Figure 11 : Random Forest Model Initialization without SMOTE

6.4 Model Evaluation:

An evaluation function was defined to calculate metrics, generate confusion matrices and ROC curves, and store results for models both with and without SMOTE.

```
# Evaluation function
def evaluate_and_store(model_name, model, X_train, y_train, X_test, y_test, with_smote=True):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Metrics calculation
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted No Churn', 'Predicted Churn'], yticklabels=['Actual No Churn', 'Actual
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix - {model_name} {"with SMOTE" if with_smote else "without SMOTE"}')
    plt.show()

    # Classification Report
    print(f"Classification Report - {model_name} {"with SMOTE" if with_smote else "without SMOTE"}:")
    print(classification_report(y_test, y_pred))

    # ROC AUC Curve
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f"ROC AUC Curve - {model_name} {"with SMOTE" if with_smote else "without SMOTE"}")
    plt.legend(loc="lower right")
    plt.show()

    # Append results
    results_with_smote.append([model_name, accuracy, precision, recall, f1])
    best_models[model_name] = (model, accuracy, recall)
```

Figure 12 : Evaluation Function for Model Performance Metrics

7 Results Analysis

The figure 8 below presents a comprehensive summary of the model comparison prior to and following the implementation of SMOTE. This comparison is essential for obtaining findings and analysing the optimal model for predicting customer retention.

```
[263]: # Convert results to DataFrames for better visualization
results_df_without_smote = pd.DataFrame(results_without_smote, columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
results_df_with_smote = pd.DataFrame(results_with_smote, columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

# Display final comparison tables
display(HTML("<h2>Model Comparison Table Without SMOTE</h2>"))
display(HTML(results_df_without_smote.to_html(index=False)))

display(HTML("<h2>Model Comparison Table With SMOTE</h2>"))
display(HTML(results_df_with_smote.to_html(index=False)))
```

Figure 13 : Converting and Displaying Model Comparison Results

.

Additionally, the ROC AUC curves for all models were generated to aid in selecting the best model

```
[276]: import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Function to plot combined ROC curves
def plot_combined_roc(models, X_test, y_test):
    plt.figure(figsize=(10, 8))

    for model_name, model in models.items():
        y_pred_proba = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{model_name} (area = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Combined ROC Curves for All Models')
    plt.legend(loc="lower right")
    plt.show()

# Storing best models after evaluation for combined ROC plot
best_models = {
    'Logistic Regression': best_logistic_model,
    'Decision Tree': best_decision_tree_model,
    'Random Forest': best_random_forest_model,
    'Gradient Boosting': best_gradient_boosting_model,
    'AdaBoost': best_adaboost_model,
    'Bagging': best_bagging_model,
    'Support Vector Machine': best_svc_model,
    'Naive Bayes': naive_bayes_model
}

# Plot combined ROC curves
plot_combined_roc(best_models, X_test, y_test)
```

Figure 14 : Plotting Combined ROC Curves for Best Models

8 Model Deployment

8.1 Model Export

The code identifies the best model based on accuracy and recall, then saves it to a pickle file for future use.

```
[273]: # Identify the best model based on accuracy and recall
best_model_name, (best_model, best_acc, best_rec) = max(best_models.items(), key=lambda item: (item[1][1], item[1][2]))

print(f"The best model is {best_model_name} with an accuracy of {best_acc} and recall of {best_rec}")

# Save the best model to a pickle file
with open(f'best_model_{best_model_name}.pkl', 'wb') as file:
    pickle.dump(best_model, file)
```

Figure 15 : Identifying and Saving the Best Model

8.2 Streamlit Application

Figure 16 : Streamlit App Setup for Customer Retention Analysis

This code snippet demonstrates the setup of a Streamlit application for predicting customer churn using a pre-trained model.

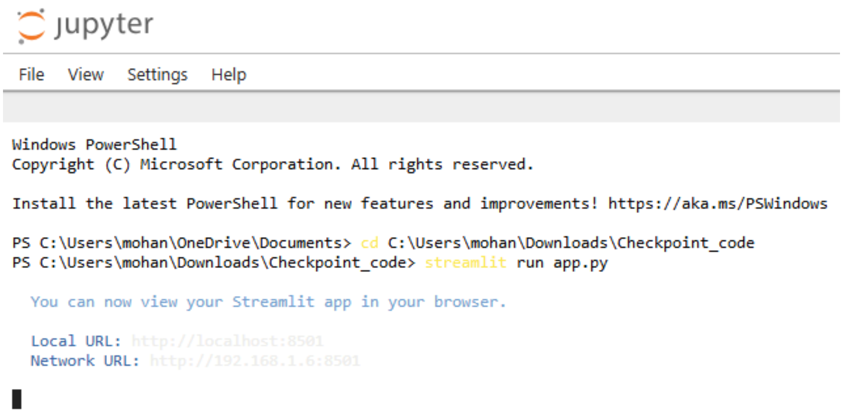


Figure 17 : Launching Streamlit App Locally via Jupyter Notebook

Executing the Streamlit application on a local machine by using the command `streamlit run app.py` through PowerShell within Jupyter Notebook. This enables real-time visualisation and analysis from within the web browser.

8.3 Deployment

This figure 12 shows the deployment process of a Streamlit app by connecting to a GitHub repository. After specifying the repository, branch, and main file path, we can deploy the app with a single click.

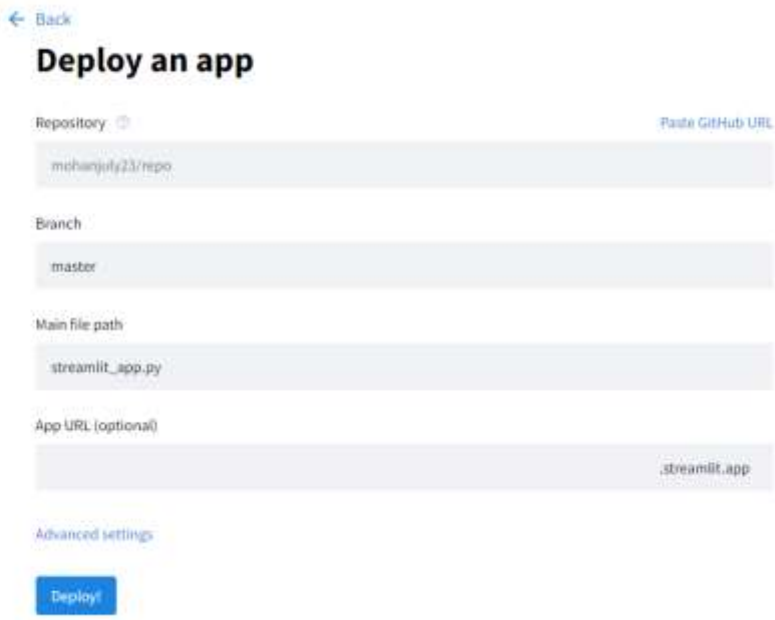


Figure 18 : Deploying Streamlit App via GitHub

Web Application:

This image shows a deployed Streamlit application for E-commerce Customer Retention Analysis, where users can input customer data to predict whether a customer will churn or not, along with the prediction probability.

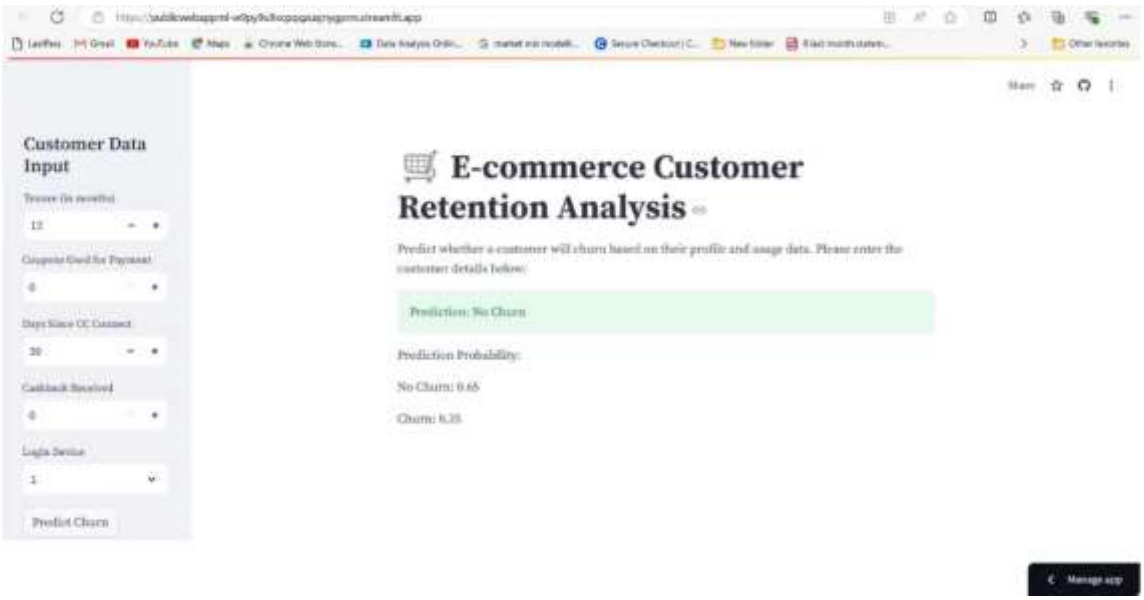


Figure 19 : Streamlit E-commerce Customer Retention Analysis Dashboard

---

https://publicwebappml-w9py9u9xcpqqzusjnygpnn.streamlit.app/

References

Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J., 2016. Data Mining: Practical Machine Learning Tools and Techniques. 4th ed. San Francisco: Morgan Kaufmann.

Géron, A., 2019. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed. Sebastopol: O'Reilly Media.

Kuhn, M. and Johnson, K., 2013. Applied Predictive Modeling. New York: Springer.