# Configuration Manual

MSc Research Project
MSc Data Analytics

Lakshmiraj Natarajan
Student ID: x22173391

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Lakshmiraj Natarajan |
| **Student ID:** | x22173391 |
| **Programme:** | MSc Data Analytics |
| **Year:** | 2023-2024 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Jorge Basilio |
| **Submission Due Date:** | 14/12/2023 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 557 |
| **Page Count:** | 12 |

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:** | Lakshmiraj Natarajan |
| **Date:** | 13th December 2023 |

## 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

Lakshmiraj Natarajan
x22173391

## 1 Introduction

This configuration manual contains the requirements in the aspect of hardware and software for this research project. The steps taken to build the model to assist in predicting customers for premium subscriptions in the fin-tech company mobile application were documented in this manual.

## 2 System Configuration

### 2.1 Hardware Configuration

Table 1: Hardware Configuration Details

| **Hardware Configuration** | |
|---|---|
| **Component** | **Details** |
| Operating System | MacOS 13 |
| Disk Space | 27.0 GB utilized out of 107.7 GB |
| RAM | 12.7 GB |
| Environment Model Name | Intel(R) Xeon(R) CPU @ 2.20GHz |

### 2.2 Software Configuration

Table 2: Software Configuration Details

| **Software Configuration** | |
|---|---|
| **Component** | **Details** |
| Programming Language | Python Version 3.10.12 |
| IDE | Google Colab |
| Browser | Google Chrome |
| DBMS | Google Drive |

## 3 Importing Library Tools

Essential libraries needed for this project are shown in Fig. 1.

```
[1] from google.colab import drive
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from datetime import datetime
    import os
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from keras.models import Sequential
    from keras.layers import Dense
    from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
    from keras.regularizers import l2
    from sklearn.model_selection import StratifiedKFold
    from sklearn.model_selection import cross_val_score
```

Figure 1: Importing necessary library

```
[2] drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[3]

    data_folder = '/content/drive/MyDrive/Colab Notebooks/thesisData'
    file_path = data_folder + '/FineTech_appData.csv'

    # Read the CSV file into a Pandas DataFrame
    data = pd.read_csv(file_path)
```

Figure 2: Data acquisition and EDA - Data mounting to the environment

# 4 Data acquisition and EDA

Data acquisition and Exploratory Data Analysis process code snippets were attached in this section. In Fig. 2. Data mounting to the environment i.e., Colab. In Fig. 3. Distribution with the age and Fig. 4. Distribution with hour. In Fig. 5. Distribution with day of the week and with Fig. 6. Correlation ratio with the heat map.

# 5 Data Transformation and Feature Engineering

# 6 Data Preparation

# 7 Model Implementation

```
[5] # Histogram for 'age'
    sns.histplot(data['age'], kde=True)
    plt.title('Distribution of Age')
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.show()
```

Figure 3: Data acquisition and EDA - Distribution with the age

```
[6]  sns.histplot(data['hour'], kde=True)
     plt.title('Distribution of hour')
     plt.xlabel('hour')
     plt.ylabel('Frequency')

     plt.show()
```

Figure 4: Data acquisition and EDA - Distribution with hour

```
[7]  sns.histplot(data['dayofweek'], kde=True)
     plt.title('Distribution of Day of the Week')
     plt.xlabel('dayofweek')
     plt.ylabel('Frequency')
     plt.show()
```

Figure 5: Data acquisition and EDA - Distribution with day of the week

```
[12]

    ## Correlation Matrix
    sns.set(style="white", font_scale=2)

    # Compute the correlation matrix
    corr = dataset2.corr()

    # Generate a mask for the upper triangle
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=(18, 15))
    f.suptitle("Correlation Matrix", fontsize = 40)

    # Generate a custom diverging colormap
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Figure 6: Data acquisition and EDA - Correlation matrix

```
[13] # Assuming screens in 'screen_list' are comma-separated
     distinct_screens = set()

     # Loop through each row and extract distinct screens
     for screens in data['screen_list']:
         for screen in screens.split(','):
             distinct_screens.add(screen.strip())  # Remove any leading or trailing whitespaces

     # Display the distinct screens
     print(distinct_screens)
```

Figure 7: Data Transformation - formating screenlist

```
[20]
# Assuming 'data' is a DataFrame with a column named 'first_open'
data["first_open"] = [datetime.strptime(row_date, "%Y-%m-%d %H:%M:%S.%f") for row_date in data["first_open"]]
```

Figure 8: Data Transformation - Calculating first open time

```
[22] # Assuming 'data' is a DataFrame with a column named 'enrolled_date'
data["enrolled_date"] = [datetime.strptime(row_date, "%Y-%m-%d %H:%M:%S.%f") if isinstance(row_date, str) else row_date for row_date in data["enrolled_date"]]
```

Figure 9: Data Transformation - Calculating enrolled time

```
data["difference"] = (data.enrolled_date-data.first_open).astype('timedelta64[h]')
response_hist = plt.hist(data["difference"].dropna())
plt.title('Distribution of Time-Since-Screen-Reached')
plt.show()
```

Figure 10: Data Transformation - Calculating difference of time between first open and enrolled time

```
[26] data.loc[data.difference > 50, 'enrolled'] = 0
     data = data.drop(columns=['enrolled_date', 'difference', 'first_open'])
```

Figure 11: Data Transformation - Defining time frame to make optimal model

```
[27] # Load Top Screens

     topScreen_file_path = data_folder + '/top_screens.csv'

     # Read the CSV file into a Pandas DataFrame
     top_screens = pd.read_csv(topScreen_file_path).top_screens.values

     top_screens
```

Figure 12: Data Transformation - Accessing top screen data source file

```
[28] # Mapping Screens to Fields
     data["screen_list"] = data.screen_list.astype(str) + ','

     for sc in top_screens:
         data[sc] = data.screen_list.str.contains(sc).astype(int)
         data['screen_list'] = data.screen_list.str.replace(sc+",", "")
```

Figure 13: Data Transformation - Mapping top screen to reduce feature

```
[30] data['Other_screenlist'] = data.screen_list.str.count(",")
     data = data.drop(columns=['screen_list'])
```

Figure 14: Data Transformation - Grouping the unmatched screen list features into Other screen list feature

```
[32] savings_screens_funnel = ["Saving1",
                              "Saving2",
                              "Saving2Amount",
                              "Saving4",
                              "Saving5",
                              "Saving6",
                              "Saving7",
                              "Saving8",
                              "Saving9",
                              "Saving10"]
     data["SavingCount_col"] = data[savings_screens_funnel].sum(axis=1)
     data = data.drop(columns=savings_screens_funnel)
```

Figure 15: Data Transformation - Creating funnel for same sequence saving features

```
[33] #credit monetering funnel

     cm_screens_funnel = ["Credit1",
                          "Credit2",
                          "Credit3",
                          "Credit3Container",
                          "Credit3Dashboard"]
     data["CMCount_col"] = data[cm_screens_funnel].sum(axis=1)
     data = data.drop(columns=cm_screens_funnel)
```

Figure 16: Data Transformation - Creating funnel for same sequence credit monitoring features

```
[34] #credit card funnel

     cc_screens_funnel = ["CC1",
                          "CC1Category",
                          "CC3"]
     data["CCCount_col"] = data[cc_screens_funnel].sum(axis=1)
     data = data.drop(columns=cc_screens_funnel)
```

Figure 17: Data Transformation - Creating funnel for same sequence credit card features

```
[35] # loan screen funnel

     loan_screens_funnel = ["Loan",
                            "Loan2",
                            "Loan3",
                            "Loan4"]
     data["LoansCount_col"] = data[loan_screens_funnel].sum(axis=1)
     data = data.drop(columns=loan_screens_funnel)
```

Figure 18: Data Transformation - Creating funnel for same sequence loan features

```
[37] data.to_csv('feature_engineered_data.csv', index = False)

     file_name = 'feature_engineered_data.csv'

     # Save the DataFrame to the drive
     file_path = os.path.join(data_folder, file_name)
     data.to_csv(file_path, index=False)
```

Figure 19: Data Transformation - Saving the modified dataset as a separate file

```
[39] # Splitting Independent and Response Variables
     response = modellingData["enrolled"]
     modellingData = modellingData.drop(columns="enrolled")
```

```
[40] # Splitting the dataset into the Training set and Test set

     X_train, X_test, y_train, y_test = train_test_split(modellingData, response,
                                                         test_size = 0.2,
                                                         random_state = 0)
```

Figure 20: Data Preparation - Identifying the target and response variable and test train split of the dataset

```
[45] sc_X = StandardScaler()
     X_train2 = pd.DataFrame(sc_X.fit_transform(X_train))
     X_test2 = pd.DataFrame(sc_X.transform(X_test))
     X_train2.columns = X_train.columns.values
     X_test2.columns = X_test.columns.values
     X_train2.index = X_train.index.values
     X_test2.index = X_test.index.values
     X_train = X_train2
     X_test = X_test2
```

Figure 21: Data Preparation - Feature Scaling before modeling

```
[46] # Create a Sequential model
     model = Sequential()

     # Add input layer
     model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))

     # Add hidden layers (you can customize the architecture)
     model.add(Dense(units=32, activation='relu'))
     model.add(Dense(units=16, activation='relu'))

     # Add output layer with a sigmoid activation function for binary classification
     model.add(Dense(units=1, activation='sigmoid'))

     # Compile the model
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
[47] model.fit(X_train, y_train, batch_size=64, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
500/500 [==============================] - 4s 5ms/step - loss: 0.5175 - accuracy: 0.7478 - val_loss: 0.4899 - val_accuracy: 0.7711
Epoch 2/10
500/500 [==============================] - 2s 5ms/step - loss: 0.4727 - accuracy: 0.7768 - val_loss: 0.4744 - val_accuracy: 0.7771
Epoch 3/10
500/500 [==============================] - 2s 4ms/step - loss: 0.4626 - accuracy: 0.7822 - val_loss: 0.4723 - val_accuracy: 0.7778
Epoch 4/10
500/500 [==============================] - 1s 3ms/step - loss: 0.4556 - accuracy: 0.7863 - val_loss: 0.4736 - val_accuracy: 0.7755
Epoch 5/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4509 - accuracy: 0.7874 - val_loss: 0.4688 - val_accuracy: 0.7778
Epoch 6/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4463 - accuracy: 0.7905 - val_loss: 0.4665 - val_accuracy: 0.7804
Epoch 7/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4414 - accuracy: 0.7909 - val_loss: 0.4712 - val_accuracy: 0.7766
Epoch 8/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4379 - accuracy: 0.7942 - val_loss: 0.4653 - val_accuracy: 0.7822
Epoch 9/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4343 - accuracy: 0.7977 - val_loss: 0.4700 - val_accuracy: 0.7796
Epoch 10/10
500/500 [==============================] - 2s 4ms/step - loss: 0.4304 - accuracy: 0.7984 - val_loss: 0.4683 - val_accuracy: 0.7779
<keras.src.callbacks.History at 0x7ef946abeb90>
```

Figure 22: Model Implementation - Neural Network model

```
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.7759
F1 Score: 0.7676034429119567
Precision: 0.8043903499239295
Recall: 0.7340341134470448
```

```
# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=[0, 1], columns=[0, 1])
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm, annot=True, fmt='g')
plt.title("Confusion Matrix for Neural Network model")
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

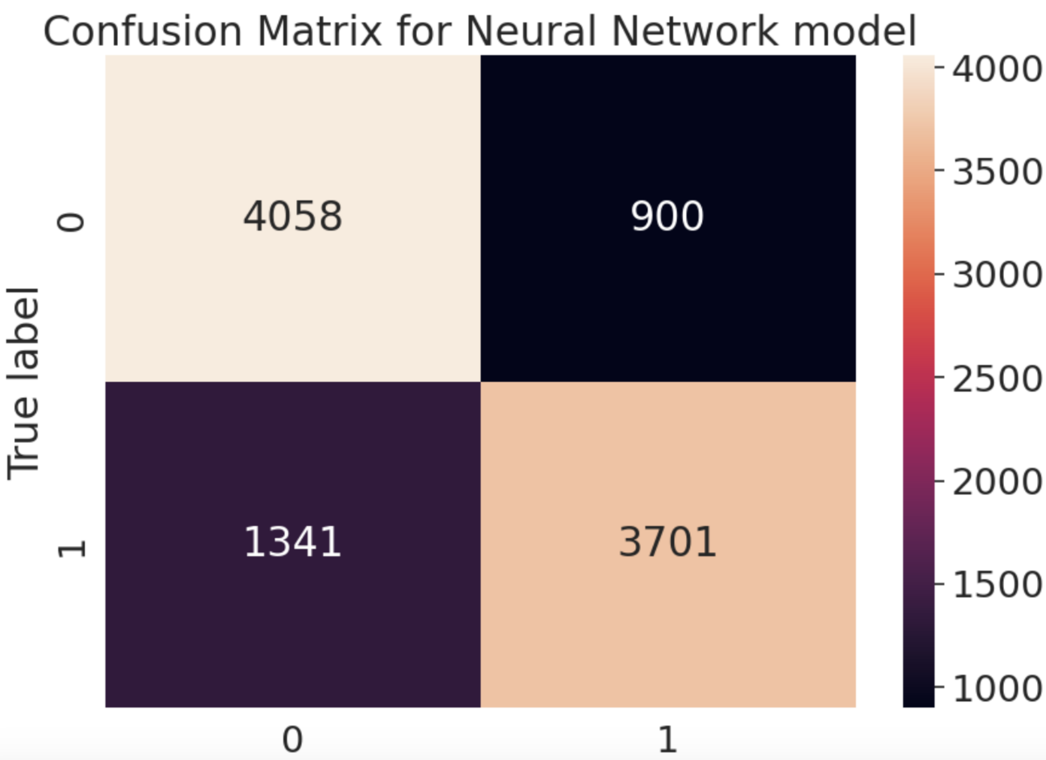



Figure 23: Model Implementation - Neural Network model evaluation metrics and confusion matrix

```
[51]
# Create a Sequential model
model = Sequential()

# Add input layer with L2 regularization
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1], kernel_regularizer=l2(0.01)))

# Add hidden layers with L2 regularization
model.add(Dense(units=32, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dense(units=16, activation='relu', kernel_regularizer=l2(0.01)))

# Add output layer with L2 regularization for binary classification
model.add(Dense(units=1, activation='sigmoid', kernel_regularizer=l2(0.01)))
```

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, batch_size=64, epochs=10, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
Epoch 1/10
500/500 [==============================] - 4s 4ms/step - loss: 0.8885 - accuracy: 0.7441 - val_loss: 0.5935 - val_accuracy: 0.7703
Epoch 2/10
500/500 [==============================] - 2s 3ms/step - loss: 0.5700 - accuracy: 0.7686 - val_loss: 0.5582 - val_accuracy: 0.7694
Epoch 3/10
500/500 [==============================] - 2s 3ms/step - loss: 0.5557 - accuracy: 0.7694 - val_loss: 0.5518 - val_accuracy: 0.7665
Epoch 4/10
500/500 [==============================] - 2s 3ms/step - loss: 0.5524 - accuracy: 0.7692 - val_loss: 0.5499 - val_accuracy: 0.7695
Epoch 5/10
500/500 [==============================] - 2s 3ms/step - loss: 0.5509 - accuracy: 0.7702 - val_loss: 0.5505 - val_accuracy: 0.7660
Epoch 6/10
500/500 [==============================] - 2s 3ms/step - loss: 0.5491 - accuracy: 0.7691 - val_loss: 0.5485 - val_accuracy: 0.7705
Epoch 7/10
500/500 [==============================] - 3s 5ms/step - loss: 0.5478 - accuracy: 0.7710 - val_loss: 0.5465 - val_accuracy: 0.7684
Epoch 8/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5475 - accuracy: 0.7703 - val_loss: 0.5470 - val_accuracy: 0.7675
Epoch 9/10
500/500 [==============================] - 2s 5ms/step - loss: 0.5469 - accuracy: 0.7696 - val_loss: 0.5486 - val_accuracy: 0.7689
Epoch 10/10
500/500 [==============================] - 2s 3ms/step - loss: 0.5461 - accuracy: 0.7713 - val_loss: 0.5460 - val_accuracy: 0.7679
313/313 [==============================] - 1s 2ms/step - loss: 0.5475 - accuracy: 0.7699
```

Figure 24: Model Implementation - Neural Network model with Regularisation technique

```
[53] y_pred = model.predict(X_test)
     y_pred_classes = (y_pred > 0.5).astype("int32")

     precision = precision_score(y_test, y_pred_classes)
     recall = recall_score(y_test, y_pred_classes)
     f1 = f1_score(y_test, y_pred_classes)
     cm = confusion_matrix(y_test, y_pred_classes)

     print("Confusion Matrix:\n", cm)
     print("Accuracy:", test_accuracy)
     print("F1 Score:", f1)
     print("Precision:", precision)
     print("Recall:", recall)
```

```
313/313 [==============================] - 1s 2ms/step
Confusion Matrix:
 [[3840 1220]
 [1081 3859]]
Accuracy: 0.7699000239372253
F1 Score: 0.7703363609142629
Precision: 0.7597952352825359
Recall: 0.7811740890688259
```

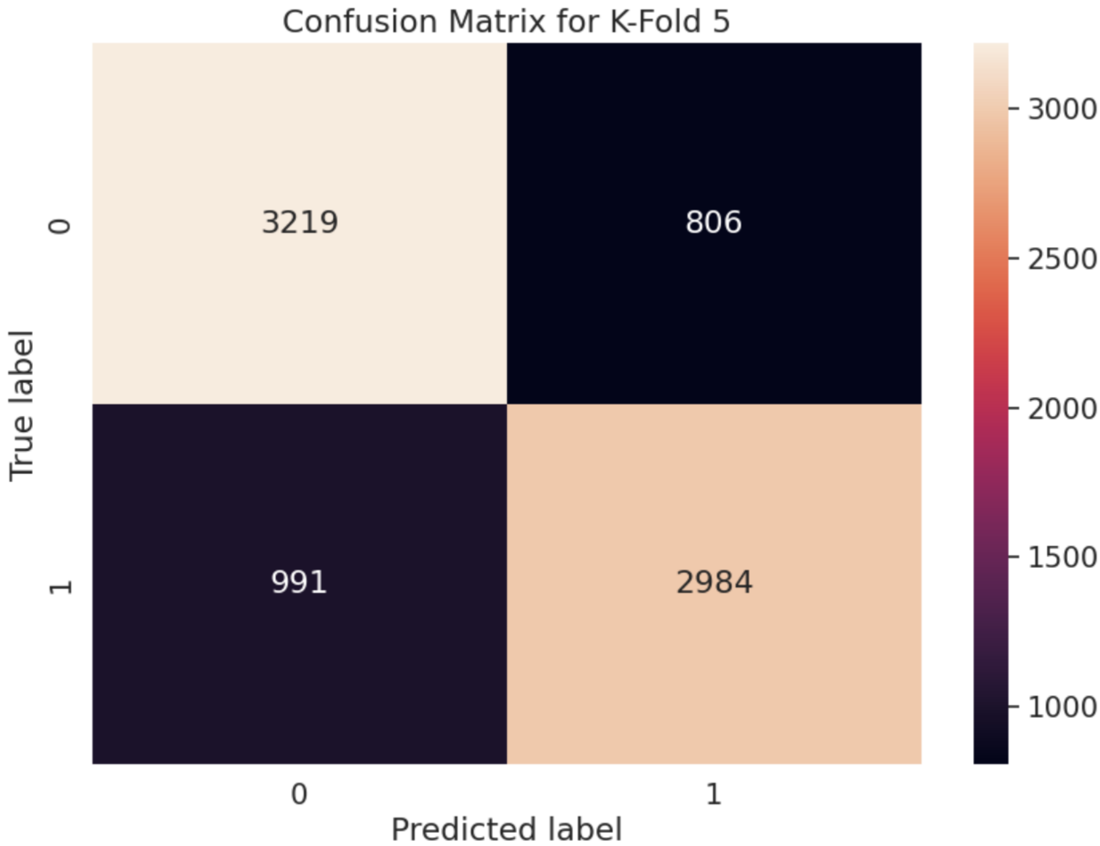
Figure 25: Model Implementation - Neural Network model with Regularisation technique evaluation metrics and confusion matrix

```python
# Define a function to create your neural network model
def create_model():
    model = Sequential()
    model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dense(units=32, activation='relu'))
    model.add(Dense(units=16, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Use StratifiedKFold for cross-validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)

# Initialize an empty list to store accuracy scores, f1 score, precision score, recall score
accuracy_scores = []
f1_scores = []
precision_scores = []
recall_scores = []


# Apply K-fold cross-validation
for i, (train, test) in enumerate(kfold.split(X_train, y_train)):
    model = create_model()  # Create a new model for each fold
    X_train_fold, X_test_fold = X_train.iloc[train], X_train.iloc[test]
    y_train_fold, y_test_fold = y_train.iloc[train], y_train.iloc[test]

    model.fit(X_train_fold, y_train_fold, epochs=10, batch_size=64, verbose=0)
    y_pred_fold = (model.predict(X_test_fold) > 0.5).astype(int)

    # Calculate metrics for this fold
    accuracy_fold = accuracy_score(y_test_fold, y_pred_fold)
    f1_fold = f1_score(y_test_fold, y_pred_fold)
    precision_fold = precision_score(y_test_fold, y_pred_fold)
    recall_fold = recall_score(y_test_fold, y_pred_fold)

    accuracy_scores.append(accuracy_fold)
    f1_scores.append(f1_fold)
    precision_scores.append(precision_fold)
    recall_scores.append(recall_fold)
```

```
250/250 [==============================] - 1s 3ms/step
250/250 [==============================] - 0s 2ms/step
250/250 [==============================] - 1s 3ms/step
250/250 [==============================] - 0s 2ms/step
250/250 [==============================] - 1s 2ms/step
```

Figure 26: Model Implementation - Neural Network model with K-Fold cross-validation technique

```
    # Calculating  and ploting confusion matrix
    cm = confusion_matrix(y_test_fold, y_pred_fold)
    df_cm = pd.DataFrame(cm, index=[0, 1], columns=[0, 1])
    plt.figure(figsize=(10, 7))
    sns.heatmap(df_cm, annot=True, fmt='g')
    plt.title(f"Confusion Matrix for K-Fold {i + 1}")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```




```
print("Cross-Validation Results (Accuracy):", accuracy_scores)
print("Mean Accuracy:", np.mean(accuracy_scores))
print("Standard Deviation of Accuracy:", np.std(accuracy_scores))
print("Average F1 Score:", np.mean(f1_scores))
print("Average Precision:", np.mean(precision_scores))
print("Average Recall:", np.mean(recall_scores))
```

```
Cross-Validation Results (Accuracy): [0.77425, 0.780125, 0.777875, 0.7815, 0.775375]
Mean Accuracy: 0.7778249999999999
Standard Deviation of Accuracy: 0.0027415780127510313
Average F1 Score: 0.769360630628922
Average Precision: 0.794448773178319
Average Recall: 0.7459622641509435
```

Figure 27: Model Implementation - Neural Network model with K-Fold cross-validation technique evaluation metrics

## creating model by using using ensemble method

```
[60] # Train multiple models (for the ensemble)
     models = [create_model() for _ in range(3)]
     for model in models:
         model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

     # Get predictions from all models
     predictions = [model.predict(X_test) for model in models]
```

```
Epoch 1/10
500/500 [==============================] - 3s 4ms/step - loss: 0.5167 - accuracy: 0.7487 - val_loss: 0.4866 - val_accuracy: 0.7671
Epoch 2/10
500/500 [==============================] - 2s 4ms/step - loss: 0.4734 - accuracy: 0.7776 - val_loss: 0.4737 - val_accuracy: 0.7751
Epoch 3/10
500/500 [==============================] - 2s 4ms/step - loss: 0.4634 - accuracy: 0.7819 - val_loss: 0.4762 - val_accuracy: 0.7700
Epoch 4/10
500/500 [==============================] - 2s 5ms/step - loss: 0.4566 - accuracy: 0.7851 - val_loss: 0.4689 - val_accuracy: 0.7771
Epoch 5/10
500/500 [==============================] - 2s 5ms/step - loss: 0.4514 - accuracy: 0.7899 - val_loss: 0.4660 - val_accuracy: 0.7784
Epoch 6/10
500/500 [==============================] - 2s 5ms/step - loss: 0.4468 - accuracy: 0.7907 - val_loss: 0.4714 - val_accuracy: 0.7772
Epoch 7/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4437 - accuracy: 0.7917 - val_loss: 0.4647 - val_accuracy: 0.7804
Epoch 8/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4397 - accuracy: 0.7948 - val_loss: 0.4679 - val_accuracy: 0.7779
Epoch 9/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4368 - accuracy: 0.7964 - val_loss: 0.4670 - val_accuracy: 0.7782
Epoch 10/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4329 - accuracy: 0.7997 - val_loss: 0.4694 - val_accuracy: 0.7782
Epoch 1/10
500/500 [==============================] - 3s 4ms/step - loss: 0.5160 - accuracy: 0.7470 - val_loss: 0.4872 - val_accuracy: 0.7630
Epoch 2/10
500/500 [==============================] - 2s 5ms/step - loss: 0.4752 - accuracy: 0.7767 - val_loss: 0.4781 - val_accuracy: 0.7705
Epoch 3/10
500/500 [==============================] - 3s 5ms/step - loss: 0.4652 - accuracy: 0.7817 - val_loss: 0.4760 - val_accuracy: 0.7706
Epoch 4/10
500/500 [==============================] - 2s 5ms/step - loss: 0.4579 - accuracy: 0.7863 - val_loss: 0.4699 - val_accuracy: 0.7766
Epoch 5/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4517 - accuracy: 0.7894 - val_loss: 0.4696 - val_accuracy: 0.7764
Epoch 6/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4479 - accuracy: 0.7889 - val_loss: 0.4675 - val_accuracy: 0.7778
Epoch 7/10
500/500 [==============================] - 1s 3ms/step - loss: 0.4436 - accuracy: 0.7932 - val_loss: 0.4725 - val_accuracy: 0.7725
Epoch 8/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4393 - accuracy: 0.7956 - val_loss: 0.4663 - val_accuracy: 0.7810
Epoch 9/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4366 - accuracy: 0.7974 - val_loss: 0.4681 - val_accuracy: 0.7768
Epoch 10/10
500/500 [==============================] - 2s 3ms/step - loss: 0.4332 - accuracy: 0.7987 - val_loss: 0.4719 - val_accuracy: 0.7789
Epoch 1/10
500/500 [==============================] - 5s 8ms/step - loss: 0.5167 - accuracy: 0.7469 - val_loss: 0.4884 - val_accuracy: 0.7667
Epoch 2/10
500/500 [==============================] - 1s 3ms/step - loss: 0.4733 - accuracy: 0.7772 - val_loss: 0.4765 - val_accuracy: 0.7717
Epoch 3/10
```

Figure 28: Model Implementation - Neural Network model with Ensemble Method

```python
# Average the predictions for ensemble
average_predictions = np.mean(predictions, axis=0)
average_predictions = (average_predictions > 0.5).astype(int)  # Thresholding to get binary predictions

# Compute the evaluation metrics
accuracy = accuracy_score(y_test, average_predictions)
f1 = f1_score(y_test, average_predictions)
precision = precision_score(y_test, average_predictions)
recall = recall_score(y_test, average_predictions)

# Print the evaluation metrics
print(f'Accuracy: {accuracy:}')
print(f'F1 Score: {f1:}')
print(f'Precision: {precision:}')
print(f'Recall: {recall:}')
```

```
Accuracy: 0.7822
F1 Score: 0.777391659852821
Precision: 0.8019822859552931
Recall: 0.754264180880603
```

```python
# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, average_predictions)
df_cm = pd.DataFrame(cm, index=[0, 1], columns=[0, 1])
plt.figure(figsize=(10, 7))
sns.heatmap(df_cm, annot=True, fmt='g')
plt.title("Confusion Matrix for Ensemble Model")
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

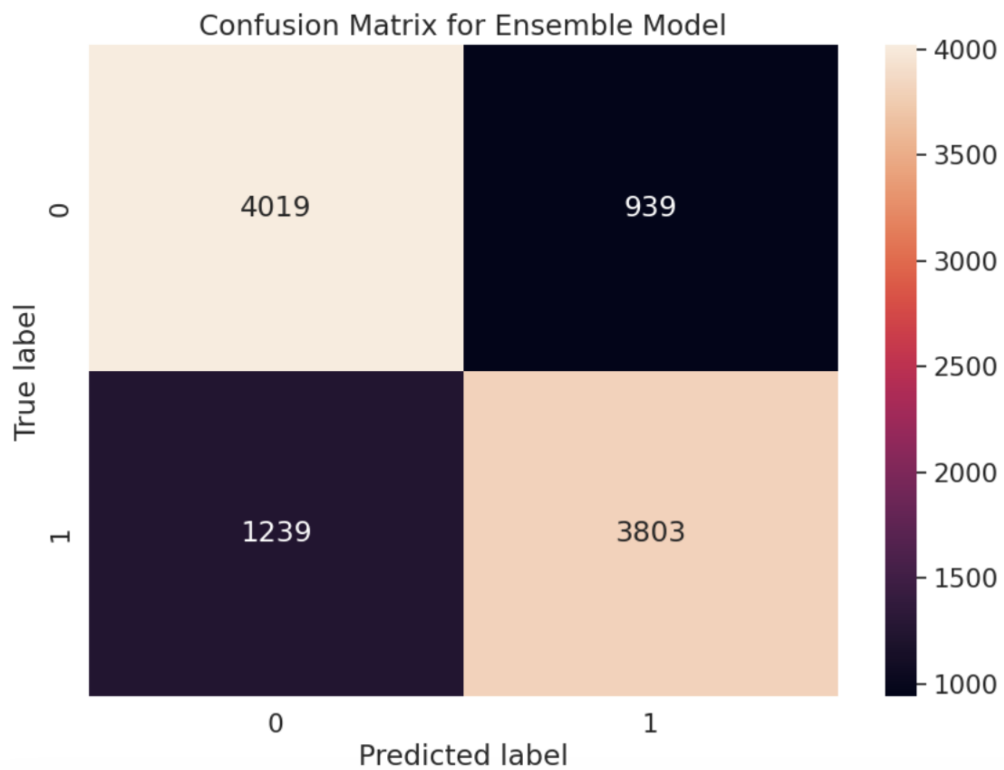



Figure 29: Model Implementation - Neural Network model with Ensemble Method evaluation metrics