Configuration Manual

MSc Research Project
Data Analytics

Florian Demir
Student ID: x20216301

School of Computing
National College of
Ireland

Supervisor: Mohammed Hasanuzzaman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Florian Demir
Student ID:	x20216301
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Mohammed Hasanuzzaman
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	2268
Page Count:	11

Signature:	Florian Demir
Date:	12th August 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 keepa copy on computer.	□

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

Configuration Manual

Florian Demir
x20216301

1 Abstract

The clothing industry appears as a sector that has existed for centuries, developed every year and will never end. The biggest losses in this sector are the cost of returned products to companies and the transportation and storage costs of products. Considering real-life conditions, less return means less waste, and less waste means more profit for companies. To best address these challenges and obtain objective, scientific results, this study introduces a new approach to predicting rates of return using Feed-Forward Neural Networks, a new type of machine learning technique. In this study, to achieve optimal results in unbalanced data sets, samples in the minority class were increased with the Synthetic Minority Oversampling Technique, hyperparameter optimization was performed with RandomSearchCV, and a Feed-forward Neural Networks model was established with an accuracy rate of 88.7% to estimate the return rates.

2 System Requirements

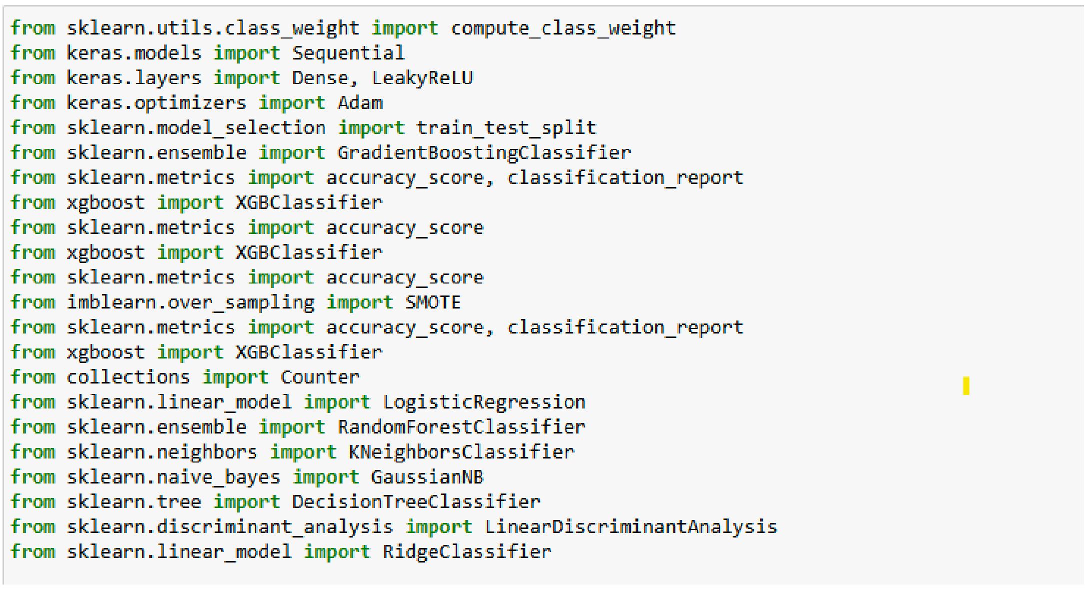
- Operating System: Windows, Mac, or Linux.
- Processor: Intel Core i5 8th Gen
- RAM: At least 8GB (16GB preferred).
- Disk Space: Free space that is retrievable and easily accessible, and this should be a minimum of 5GB.
- Internet Connection: During downloading of pre-trained models and datasets.

3 Software Requirements

- Python Version: Python 3.8 or higher.
- Jupyter Notebook

Python was used as the programming language in this research. Python libraries NumPy, Pandas, Seaborn, Tensorflow/Keras were used to run the models and analysis. The NumPy library allows us to quickly operate on mathematical data such as arrays and matrices. Pandas library has been a useful library in terms of fast and effective use of data frames during the data pre-processing phase, especially in time series data analysis. The Seaborn library, which has been used specifically to convert data into visuals, has been used to create statistical graphs. This library is based on the MatPotLib library. The reason why this library is used instead of other data visualization libraries is that it is compatible with the Pandas library. TensorFlow/Keras library is a library where the model for machine learning applications will be created and the neural networks it creates will be trained and run. Processes requiring GPU were performed on a system with RTX3080.

All related libraries can be installed using:



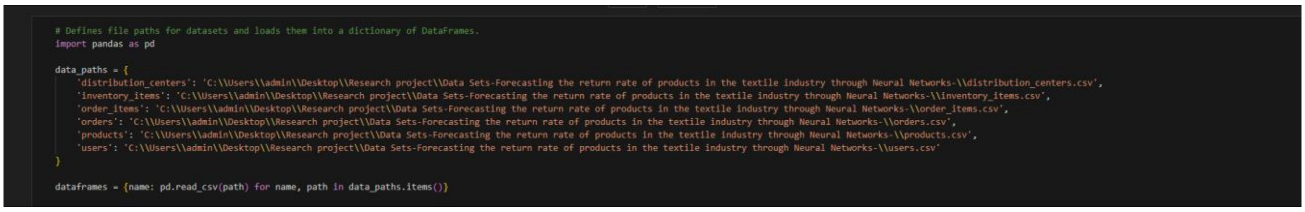
```
from sklearn.utils.class_weight import compute_class_weight
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
from collections import Counter
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import RidgeClassifier
```

Figure 1: Import Commands

4 Data Preparation

4.1 Datasets

A total of 6 datasets were used in this study. The data used was synthetically produced and made available. The data was combined by determining primary keys. A total of 490705 lines of data were used for this study. While associating the data sets, emphasis was given to the names of the products, user information of those who ordered and inventory information. In the first data analysis, it was seen that 10% of the purchased products were returned and 15% were cancelled after the order. Since the data set was created synthetically, the use of this data does not create any ethical problems. The order items dataset contains 181,759 rows and 11 columns of data. This data set shows the information and features of the ordered products. When using this data set, I focused on the date and time information required for the model and created a new dummy variable from this data set. The orders dataset consists of 125,226 rows and 12 columns in total and contains data showing the name of customers' orders, the time of the orders, the brand, category and prices of the products in the order. The products dataset shows the information of the products to be sold and contains a total of 29,120 rows and 9 columns of data. The distribution centres dataset contains 10 lines of data. This data set shows latitude and longitude data of 10 different distribution centres. The inventory items dataset covers the general features of the products, from when they were sold, to the cost of the product, to the category of the product, to the type and brand of the product. This dataset contains 490,705 rows and 12 rows of data. The users dataset contains a total of 100,000 rows and 11 columns of data showing the personal data of customers who purchased the products.

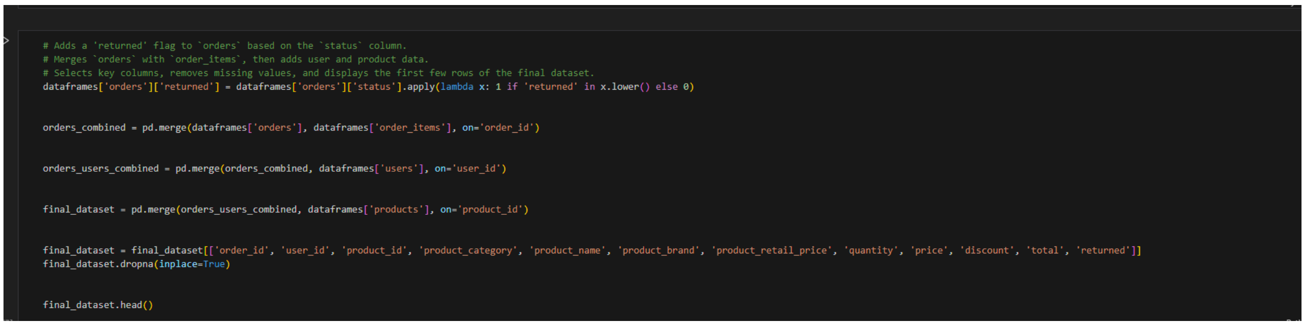


```
# Defines file paths for datasets and loads them into a dictionary of DataFrames.
import pandas as pd

data_paths = {
    'distribution_centers': 'C:\\Users\\admin\\Desktop\\Research project\\Data Sets-Forecasting the return rate of products in the textile industry through Neural Networks-\\distribution_centers.csv',
    'inventory_items': 'C:\\Users\\admin\\Desktop\\Research project\\Data Sets-Forecasting the return rate of products in the textile industry through Neural Networks-\\inventory_items.csv',
    'order_items': 'C:\\Users\\admin\\Desktop\\Research project\\Data Sets-Forecasting the return rate of products in the textile industry through Neural Networks-\\order_items.csv',
    'orders': 'C:\\Users\\admin\\Desktop\\Research project\\Data Sets-Forecasting the return rate of products in the textile industry through Neural Networks-\\orders.csv',
    'products': 'C:\\Users\\admin\\Desktop\\Research project\\Data Sets-Forecasting the return rate of products in the textile industry through Neural Networks-\\products.csv',
    'users': 'C:\\Users\\admin\\Desktop\\Research project\\Data Sets-Forecasting the return rate of products in the textile industry through Neural Networks-\\users.csv'
}

dataframes = {name: pd.read_csv(path) for name, path in data_paths.items()}
```

Figure 2: The process of loading data sets.

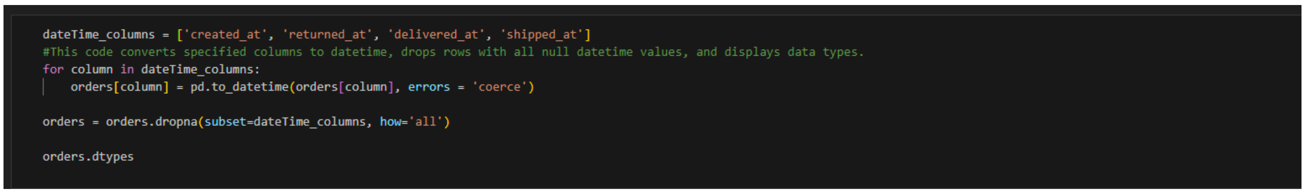
4.2 Data Pre-processing and Processing

I combined the Orders and Order items data sets and combined the information of the customers who bought the product and the product information in another data set. After examining the data after this combination, I cleared some columns that were not important for the model. After the review, I created new dummy variables. A very small number of products did not have either their brand information or their names in the data set. These lines were removed during the data preparation stage. After this stage, other missing data were identified. At this stage, missing numerical data was filled in using the median. Missing categorical data was removed from the data sets in rare cases, and in frequently found cases, the value was filled with the most recurring categorical values. NumPy and Pandas libraries were used to perform these operations. I also created two new features that include the categories of returned products and the brands of returned products. Then, I visualized the two features I created.



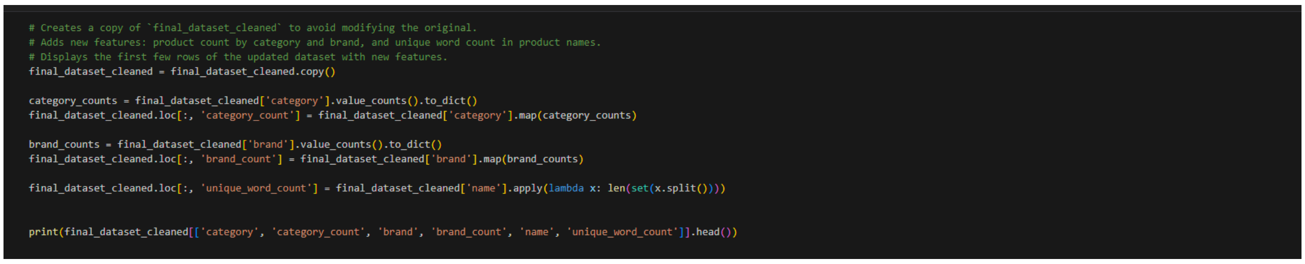
```python
# Adds a 'returned' flag to `orders` based on the `status` column.
# Merges `orders` with `order_items`, then adds user and product data.
# Selects key columns, removes missing values, and displays the first few rows of the final dataset.
dataframes['orders']['returned'] = dataframes['orders']['status'].apply(lambda x: 1 if 'returned' in x.lower() else 0)


orders_combined = pd.merge(dataframes['orders'], dataframes['order_items'], on='order_id')


orders_users_combined = pd.merge(orders_combined, dataframes['users'], on='user_id')


final_dataset = pd.merge(orders_users_combined, dataframes['products'], on='product_id')


final_dataset = final_dataset[['order_id', 'user_id', 'product_id', 'product_category', 'product_name', 'product_brand', 'product_retail_price', 'quantity', 'price', 'discount', 'total', 'returned']]
final_dataset.dropna(inplace=True)


final_dataset.head()
```

Figure 3: Combining data, removing missing values



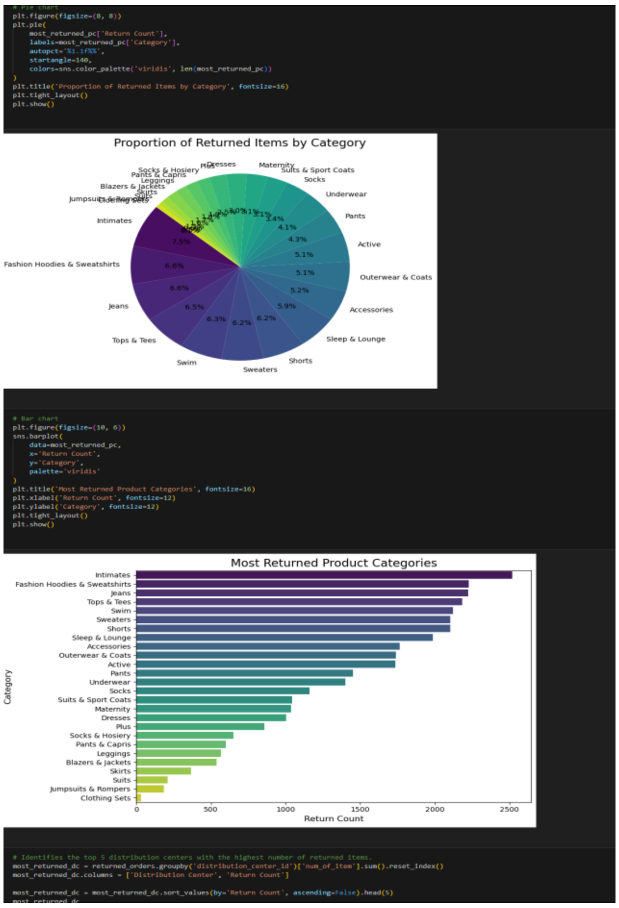
```python
dateTime_columns = ['created_at', 'returned_at', 'delivered_at', 'shipped_at']
#This code converts specified columns to datetime, drops rows with all null datetime values, and displays data types.
for column in dateTime_columns:
    orders[column] = pd.to_datetime(orders[column], errors = 'coerce')

orders = orders.dropna(subset=dateTime_columns, how='all')

orders.dtypes
```

Figure 4: The process of removing data with empty date and time columns from the dataset and changing the data type of the required columns.



```python
# Creates a copy of `final_dataset_cleaned` to avoid modifying the original.
# Adds new features: product count by category and brand, and unique word count in product names.
# Displays the first few rows of the updated dataset with new features.
final_dataset_cleaned = final_dataset_cleaned.copy()

category_counts = final_dataset_cleaned['category'].value_counts().to_dict()
final_dataset_cleaned.loc[:, 'category_count'] = final_dataset_cleaned['category'].map(category_counts)

brand_counts = final_dataset_cleaned['brand'].value_counts().to_dict()
final_dataset_cleaned.loc[:, 'brand_count'] = final_dataset_cleaned['brand'].map(brand_counts)

final_dataset_cleaned.loc[:, 'unique_word_count'] = final_dataset_cleaned['name'].apply(lambda x: len(set(x.split())))


print(final_dataset_cleaned[['category', 'category_count', 'brand', 'brand_count', 'name', 'unique_word_count']].head())
```

Figure 5: Adding new features


```
# Pie chart
plt.figure(figsize=(8, 8))
plt.pie(
    most_returned_pc['Return Count'],
    labels=most_returned_pc['Category'],
    autopct='%1.1f%%',
    startangle=140,
    colors=sns.color_palette('viridis', len(most_returned_pc))
)
plt.title('Proportion of Returned Items by Category', fontsize=16)
plt.tight_layout()
plt.show()
```

```
# Bar chart
plt.figure(figsize=(10, 6))
sns.barplot(
    data=most_returned_pc,
    x='Return Count',
    y='Category',
    palette='viridis'
)
plt.title('Most Returned Product Categories', fontsize=16)
plt.xlabel('Return Count', fontsize=12)
plt.ylabel('Category', fontsize=12)
plt.tight_layout()
plt.show()
```

```
# Identifies the top 5 distribution centers with the highest number of returned items.
most_returned_dc = returned_orders.groupby('distribution_center_id')['num_of_item'].sum().reset_index()
most_returned_dc.columns = ['Distribution Center', 'Return Count']

most_returned_dc = most_returned_dc.sort_values(by='Return Count', ascending=False).head(5)
most_returned_dc
```

Figure 6: Visualizing new features

4.2.1 Determination of the target variable required for the model

At this stage, missing data was identified, the data set was examined, and a new dummy variable was created in this data set. At this stage, dependent and independent variables are determined. While creating the dependent variable, the "returned at" column was used to assign the value 1 to another column if there was any date in this column, or 0 otherwise. A value of 0 indicates that the sold products were not returned for any reason, and a value of 1 indicates that the sold products were returned for any reason. The main purpose here is to determine the target variable for the Feed-forward Neural Networks model.

4.2.3 Quantification of categorical data for the model

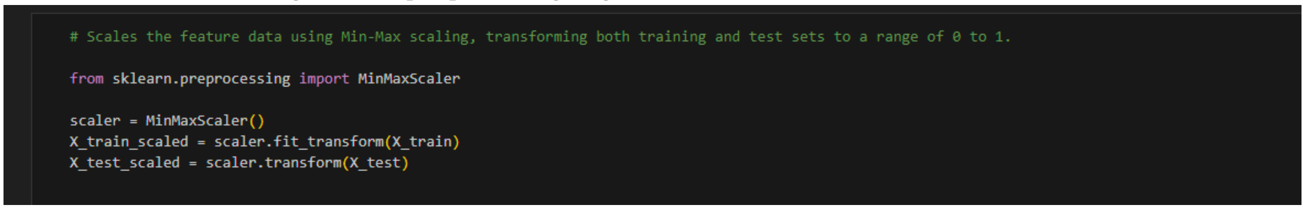
At this stage, the One-Hot Encoding method was used. Variables such as the distribution centre of the product, the category of the product, the platform from which the product was purchased, and the gender of the customers were converted into numerical data with the One-Hot Encoding method.

4.2.4 New feature selection and dimension reduction for the model

Until this stage, the data sets were first cleaned, then the target variable was determined, and then the categorical data was digitized. When the entire data set obtained was re-examined, unnecessary columns that reflected the characteristics of the data were removed, as some columns would slow down the operation of the model. In addition, those that aim to increase the accuracy of the model (for example, the category of the product purchased and returned by the customer) have been determined. Since the data set used contained approximately 500,000 pieces of data, multi-dimensional data were transformed into lower-dimensional data by using the Principal Component Analysis method, which is a dimension reduction technique. In the feature selection, some of the columns that were not directly related to the selected target variable were removed. In this way, the model showed a faster performance. In addition, the number of samples in the minority class was increased in order to make better predictions in the minority class of the model.

4.2.5 Standardization of data for the model

At this stage, the data to be used was standardized so that the Feed-forward Neural Networks model could learn faster and more robustly than other models. To do this, I used the Standard Scaler method in this study.
After all these data cleaning and data pre-processing stages, a new data set with a total of 158895 lines was obtained.

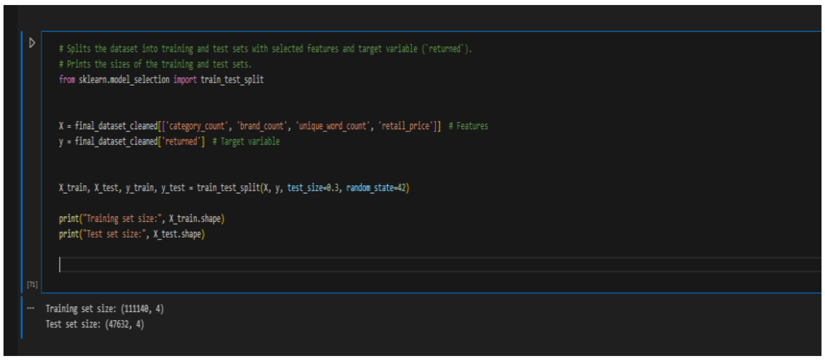


```
# Scales the feature data using Min-Max scaling, transforming both training and test sets to a range of 0 to 1.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 7: Standardization

4.2.6 Dividing the dataset into training and testing datasets for the model

Data set separation was made to ensure that the performance of the model under global conditions gives reliable results. The data set used in the study was divided into two as training and test data sets to be used in the training phase. The Education data set resulting from this separation contains 111140 data in total. The test data set has 47632 data. The training data set contains 70% of the data set that has gone through all pre-processing processes, and the test data set contains 30%. I trained the model using the training data set and tested the model with the test data set.



```
# Splits the dataset into training and test sets with selected features and target variable (`returned`).
# Prints the sizes of the training and test sets.
from sklearn.model_selection import train_test_split


X = final_dataset_cleaned[['category_count', 'brand_count', 'unique_word_count', 'retail_price']]  # Features
y = final_dataset_cleaned['returned']  # Target variable


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print("Training set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
Training set size: (111140, 4)
Test set size: (47632, 4)
```

Figure 8: Dividing the dataset into training and testing datasets for the model

4.2.7 Exploratory Data Analysis (EDA)

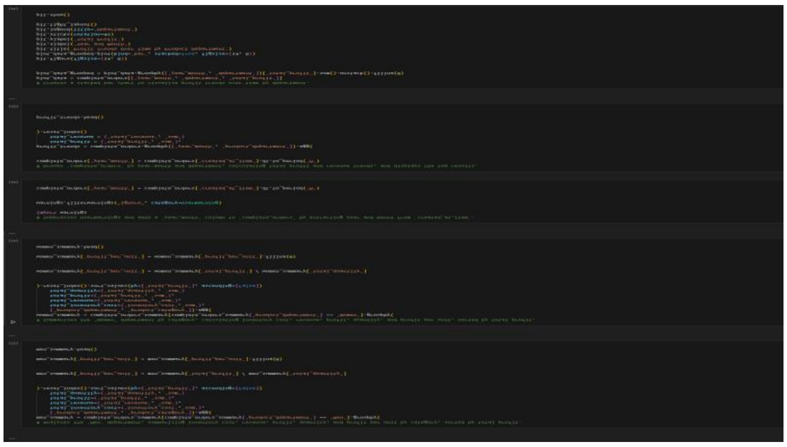
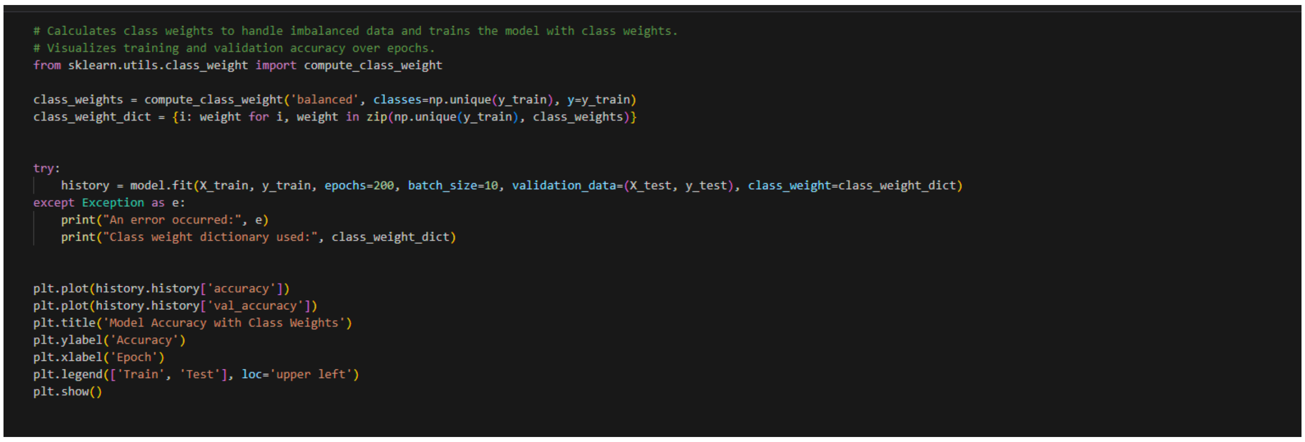


Figure 9: Exploratory Data Analysis

Synthetic Minority Over Sampling Technique and class weight parameters were used to eliminate the class imbalance problem during training phase.

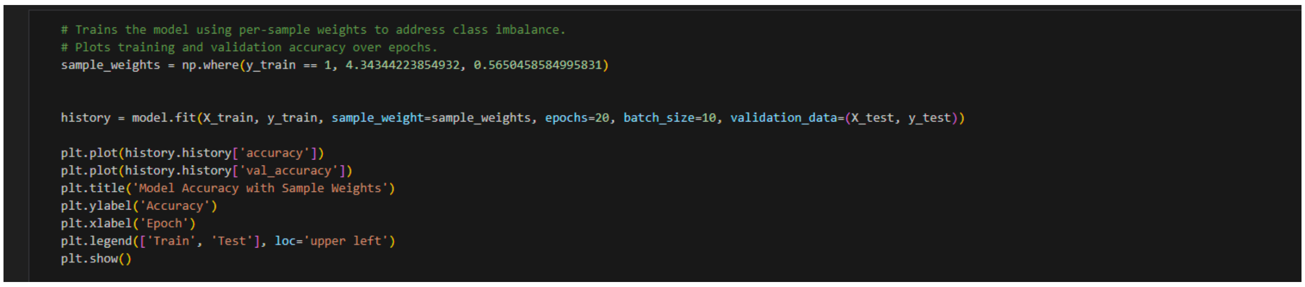
Class Weights



```python
# Calculates class weights to handle imbalanced data and trains the model with class weights.
# Visualizes training and validation accuracy over epochs.
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weight_dict = {i: weight for i, weight in zip(np.unique(y_train), class_weights)}


try:
    history = model.fit(X_train, y_train, epochs=200, batch_size=10, validation_data=(X_test, y_test), class_weight=class_weight_dict)
except Exception as e:
    print("An error occurred:", e)
    print("Class weight dictionary used:", class_weight_dict)


plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy with Class Weights')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 10: Class Weights

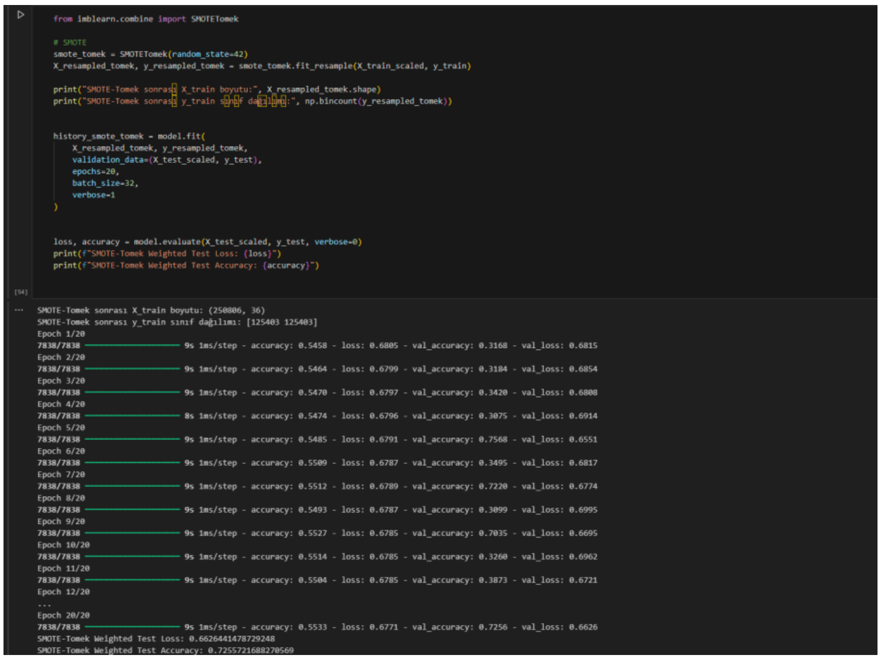
Sample Weights



```python
# Trains the model using per-sample weights to address class imbalance.
# Plots training and validation accuracy over epochs.
sample_weights = np.where(y_train == 1, 4.34344223854932, 0.5650458584995831)


history = model.fit(X_train, y_train, sample_weight=sample_weights, epochs=20, batch_size=10, validation_data=(X_test, y_test))

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy with Sample Weights')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Figure 11: Sample Weights

5 Model Implementation:

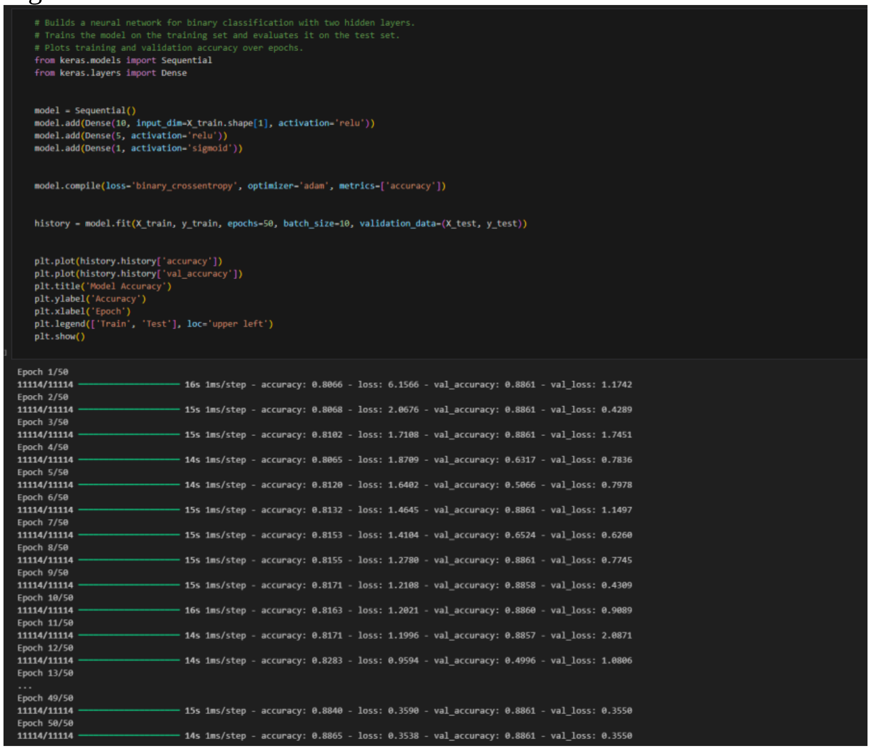


```python
from imblearn.combine import SMOTETomek

# SMOTE
smote_tomek = SMOTETomek(random_state=42)
X_resampled_tomek, y_resampled_tomek = smote_tomek.fit_resample(X_train_scaled, y_train)

print("SMOTE-Tomek sonras□ X_train boyutu:", X_resampled_tomek.shape)
print("SMOTE-Tomek sonras□ y_train s□n□f da□□l□m□:", np.bincount(y_resampled_tomek))


history_smote_tomek = model.fit(
    X_resampled_tomek, y_resampled_tomek,
    validation_data=(X_test_scaled, y_test),
    epochs=20,
    batch_size=32,
    verbose=1
)


loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"SMOTE-Tomek Weighted Test Loss: {loss}")
print(f"SMOTE-Tomek Weighted Test Accuracy: {accuracy}")
```

```
[54]
```

```
SMOTE-Tomek sonrası X_train boyutu: (250806, 36)
SMOTE-Tomek sonrası y_train sınıf dağılımı: [125403 125403]
Epoch 1/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5458 - loss: 0.6805 - val_accuracy: 0.3168 - val_loss: 0.6815
Epoch 2/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5464 - loss: 0.6799 - val_accuracy: 0.3184 - val_loss: 0.6854
Epoch 3/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5470 - loss: 0.6797 - val_accuracy: 0.3420 - val_loss: 0.6808
Epoch 4/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 8s 1ms/step - accuracy: 0.5474 - loss: 0.6796 - val_accuracy: 0.3075 - val_loss: 0.6914
Epoch 5/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5485 - loss: 0.6791 - val_accuracy: 0.7568 - val_loss: 0.6551
Epoch 6/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5509 - loss: 0.6787 - val_accuracy: 0.3495 - val_loss: 0.6817
Epoch 7/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5512 - loss: 0.6789 - val_accuracy: 0.7220 - val_loss: 0.6774
Epoch 8/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5493 - loss: 0.6787 - val_accuracy: 0.3099 - val_loss: 0.6995
Epoch 9/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5527 - loss: 0.6785 - val_accuracy: 0.7035 - val_loss: 0.6695
Epoch 10/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5514 - loss: 0.6785 - val_accuracy: 0.3260 - val_loss: 0.6962
Epoch 11/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5504 - loss: 0.6785 - val_accuracy: 0.3873 - val_loss: 0.6721
Epoch 12/20
...
Epoch 20/20
7838/7838 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.5533 - loss: 0.6771 - val_accuracy: 0.7256 - val_loss: 0.6626
SMOTE-Tomek Weighted Test Loss: 0.6626441478729248
SMOTE-Tomek Weighted Test Accuracy: 0.7255721688270569
```

Figure 12: SMOTE



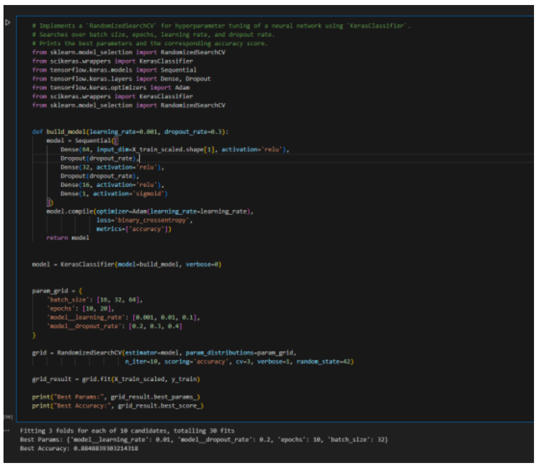
```
# Builds a neural network for binary classification with two hidden layers.
# Trains the model on the training set and evaluates it on the test set.
# Plots training and validation accuracy over epochs.
from keras.models import Sequential
from keras.layers import Dense


model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


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


history = model.fit(X_train, y_train, epochs=50, batch_size=10, validation_data=(X_test, y_test))


plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
Epoch 1/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 16s 1ms/step - accuracy: 0.8066 - loss: 6.1566 - val_accuracy: 0.8861 - val_loss: 1.1742
Epoch 2/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8068 - loss: 2.0676 - val_accuracy: 0.8861 - val_loss: 0.4289
Epoch 3/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8102 - loss: 1.7108 - val_accuracy: 0.8861 - val_loss: 1.7451
Epoch 4/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 14s 1ms/step - accuracy: 0.8065 - loss: 1.8709 - val_accuracy: 0.6317 - val_loss: 0.7836
Epoch 5/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 14s 1ms/step - accuracy: 0.8120 - loss: 1.6402 - val_accuracy: 0.5066 - val_loss: 0.7978
Epoch 6/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8132 - loss: 1.4645 - val_accuracy: 0.8861 - val_loss: 1.1497
Epoch 7/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8153 - loss: 1.4104 - val_accuracy: 0.6524 - val_loss: 0.6260
Epoch 8/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8155 - loss: 1.2780 - val_accuracy: 0.8861 - val_loss: 0.7745
Epoch 9/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8171 - loss: 1.2108 - val_accuracy: 0.8858 - val_loss: 0.4309
Epoch 10/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 16s 1ms/step - accuracy: 0.8163 - loss: 1.2021 - val_accuracy: 0.8860 - val_loss: 0.9089
Epoch 11/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 14s 1ms/step - accuracy: 0.8171 - loss: 1.1996 - val_accuracy: 0.8857 - val_loss: 2.0871
Epoch 12/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 14s 1ms/step - accuracy: 0.8283 - loss: 0.9594 - val_accuracy: 0.4996 - val_loss: 1.0806
Epoch 13/50
...
Epoch 49/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 15s 1ms/step - accuracy: 0.8840 - loss: 0.3590 - val_accuracy: 0.8861 - val_loss: 0.3550
Epoch 50/50
11114/11114 ━━━━━━━━━━━━━━━━━━━━ 14s 1ms/step - accuracy: 0.8865 - loss: 0.3538 - val_accuracy: 0.8861 - val_loss: 0.3550
```

Figure 13: First FNN Model

5.1 Hyperparameter Optimization

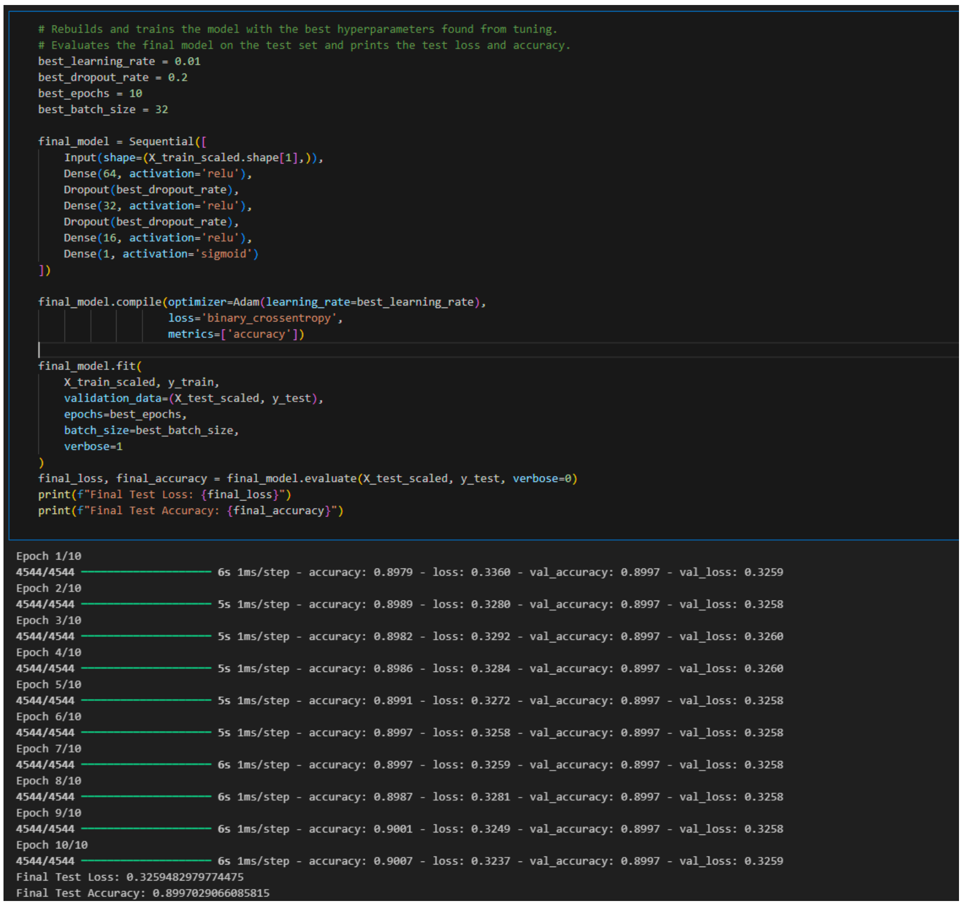
Hyperparameter optimization was performed on the model to reach the final model. For this, RandomSearchCV technique was used to provide an extra contribution to the accuracy of the model. As a result of hyper parameter optimization, the best hyper parameters were found to be 0.01, best dropout rate was found to be 0.2, best epochs were found to be 10, and best batch size was found to be 32, respectively. The accuracy measure was used as the measure of success of the created model. The main purpose of using this metric is to use the success metric as an accuracy metric in studies on this subject and to enable us to reach more accurate results when compared to other studies.



```
# Implements a `RandomizedSearchCV` for hyperparameter tuning of a neural network using `KerasClassifier`.
# Searches over batch size, epochs, learning rate, and dropout rate.
# Prints the best parameters and the corresponding accuracy score.
from sklearn.model_selection import RandomizedSearchCV
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import RandomizedSearchCV


def build_model(learning_rate=0.001, dropout_rate=0.3):
    model = Sequential([
        Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'),
        Dropout(dropout_rate),
        Dense(32, activation='relu'),
        Dropout(dropout_rate),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model


model = KerasClassifier(model=build_model, verbose=0)


param_grid = {
    'batch_size': [16, 32, 64],
    'epochs': [10, 20],
    'model__learning_rate': [0.001, 0.01, 0.1],
    'model__dropout_rate': [0.2, 0.3, 0.4]
}

grid = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
                          n_iter=10, scoring='accuracy', cv=3, verbose=1, random_state=42)

grid_result = grid.fit(X_train_scaled, y_train)

print("Best Params:", grid_result.best_params_)
print("Best Accuracy:", grid_result.best_score_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Params: {'model__learning_rate': 0.01, 'model__dropout_rate': 0.2, 'epochs': 10, 'batch_size': 32}
Best Accuracy: 0.8848839303214318
```

Figure 14: Hyperparameter Optimization



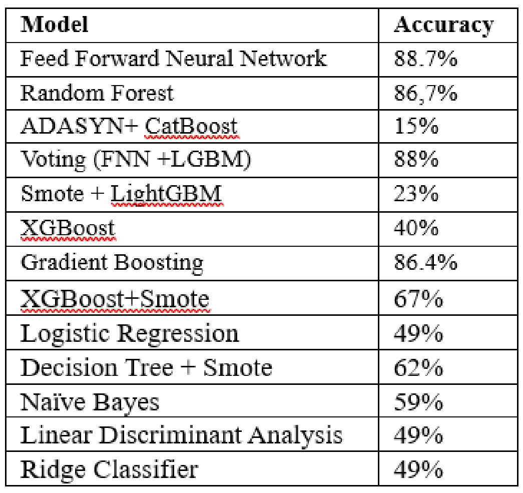
```
# Rebuilds and trains the model with the best hyperparameters found from tuning.
# Evaluates the final model on the test set and prints the test loss and accuracy.
best_learning_rate = 0.01
best_dropout_rate = 0.2
best_epochs = 10
best_batch_size = 32

final_model = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),
    Dense(64, activation='relu'),
    Dropout(best_dropout_rate),
    Dense(32, activation='relu'),
    Dropout(best_dropout_rate),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

final_model.compile(optimizer=Adam(learning_rate=best_learning_rate),
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

final_model.fit(
    X_train_scaled, y_train,
    validation_data=(X_test_scaled, y_test),
    epochs=best_epochs,
    batch_size=best_batch_size,
    verbose=1
)
final_loss, final_accuracy = final_model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"Final Test Loss: {final_loss}")
print(f"Final Test Accuracy: {final_accuracy}")
```

```
Epoch 1/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.8979 - loss: 0.3360 - val_accuracy: 0.8997 - val_loss: 0.3259
Epoch 2/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8989 - loss: 0.3280 - val_accuracy: 0.8997 - val_loss: 0.3258
Epoch 3/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8982 - loss: 0.3292 - val_accuracy: 0.8997 - val_loss: 0.3260
Epoch 4/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8986 - loss: 0.3284 - val_accuracy: 0.8997 - val_loss: 0.3260
Epoch 5/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8991 - loss: 0.3272 - val_accuracy: 0.8997 - val_loss: 0.3258
Epoch 6/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.8997 - loss: 0.3258 - val_accuracy: 0.8997 - val_loss: 0.3258
Epoch 7/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.8997 - loss: 0.3259 - val_accuracy: 0.8997 - val_loss: 0.3258
Epoch 8/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.8987 - loss: 0.3281 - val_accuracy: 0.8997 - val_loss: 0.3258
Epoch 9/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.9001 - loss: 0.3249 - val_accuracy: 0.8997 - val_loss: 0.3258
Epoch 10/10
4544/4544 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.9007 - loss: 0.3237 - val_accuracy: 0.8997 - val_loss: 0.3259
Final Test Loss: 0.3259482979774475
Final Test Accuracy: 0.8997029066085815
```

Figure 16: Final FNN Model with best hyperparameters

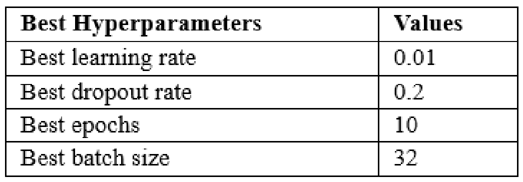
6 Model Evaluation

The accuracy rate obtained from the techniques and training and validation sets increased from 67% to 88.7%. As a result of hyper parameter optimization, the best hyper parameters were found to be 0.01, best dropout rate was found to be 0.2, best epochs were found to be 10, and best batch size was found to be 32, respectively. Additionally, the success of the result was checked and the Feed-forward Neural Networks model gave consistent accurate outputs with low losses (0.352) in all sets (training data set and test data set). I would especially like to point out that working with unbalanced data sets should not be considered as a problem in this academic study. Considering real life, no company would expect half of its products to be returned. For this reason, comparisons were made only with models that performed better in unbalanced data sets.



Model	Accuracy
Feed Forward Neural Network	88.7%
Random Forest	86,7%
ADASYN+ CatBoost	15%
Voting (FNN +LGBM)	88%
Smote + LightGBM	23%
XGBoost	40%
Gradient Boosting	86.4%
XGBoost+Smote	67%
Logistic Regression	49%
Decision Tree + Smote	62%
Naïve Bayes	59%
Linear Discriminant Analysis	49%
Ridge Classifier	49%

Table 1: Comparison of Accuracy Rates of Models.



Best Hyperparameters	Values
Best learning rate	0.01
Best dropout rate	0.2
Best epochs	10
Best batch size	32

Table 2: Best Hyperparameters

7 Execution of the Code

- Download the Dataset
- Unzip the Files into a Folder
- Open the Python File in Google Colab or Jupyter Notebook
- Change the Path to Refer to the Dataset Location
- Run the Code

8 Conclusion

As a result of the data analysis, it was observed that 10 percent of the orders in the data set were returned, and 15 percent were cancelled after they were ordered. Although we do not know why the products were returned or cancelled, the Feed Forward Neural Networks model was established with the help of the Synthetic Minority Over Sampling Technique, and the success rate obtained from the techniques and training and validation sets increased from 69% to 88.7 percent. Also, the success of the result was checked, and the Feed-forward Neural Networks model gave consistent accurate outputs with low losses in all data sets (training data set and validation data set).

After this study, it was revealed that Feed Forward Neural Networks together with Synthetic Minority Over Sampling Technique gave strong results to predict the return rates of products, especially in textile data. Being able to predict whether a product will be returned will significantly reduce costs such as transportation, production and stocking. Thanks to the established model, companies can reduce their costs and increase their profits.