Configuration Manual

MSc Research Project
MSc. in Cyber Security

Laxmi Kumthe
Student ID: x23126001

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Laxmi Bhaskar Kumthe		
Student ID:	x23126001		
Programme:	MSc. in Cyber Security	**Year:**	2023-24
Module:	H9PRAC – Research in Computing		
Supervisor:	Mr. Vikas Sahni		
Submission Due Date:	12th August, 2024		
Project Title:	Research Configuration Manual		
Word Count:	1108	**Page Count:**	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Laxmi

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

Laxmi Kumthe
Student ID: x23126001

1 Introduction

The technologies and techniques utilized during the study implementation are described in depth in the configuration handbook. The following are the contents of the various configuration manual sections:

Section 2 thoroughly describes the experimental setup. In Section 3, the various technologies and software tools employed are discussed. Section 4 provides a comprehensive, step-by-step guide for executing the machine learning workflow.

It begins with collecting a dataset of SSRF (Server-Side Request Forgery) attack data from the CIC-Bell dataset[1] provided by CIC, ensuring correct annotations and pre-processing by removing null rows and columns, duplicate values, and unnecessary columns. Data balancing is achieved using SMOTE (Synthetic Minority Over-sampling Technique) to ensure an even distribution of classes. Feature selection is performed using the Chi-square (Chi2) test, and feature scaling is done using a standard scaler. The dataset is split into training and testing sets, maintaining a balanced distribution. For training, two models are used: RandomForestClassifier and CNN-LSTM. The RandomForestClassifier model is initialized, trained with the training dataset, and saved. Its performance is evaluated using accuracy, F1-score, and other evaluation metrics. The CNN-LSTM model is created by combining Convolutional Neural Networks (CNN) for feature extraction and Long Short-Term Memory (LSTM) networks for sequential data processing. It is compiled, trained with the training dataset, and saved, with performance evaluated similarly. Both models are assessed using classification reports to identify potential biases, and the model with higher performance is selected.

2 Experimental Setup

The experiment was conducted on a personal computer configured specifically for this purpose.

- **Hardware Specifications**: The system runs on an AMD Ryzen 5 3550H processor, has 40 GB of RAM, and includes a 118 GB SSD.
- **Operating System**: Windows 11.
- **Experimental Setup**: The setup includes Windows 11, Anaconda3, Python 3.6, and Visual Studio Code.

[1] https://www.unb.ca/cic/datasets/dns-2021.html

3 Technologies and Software Used for Implementation

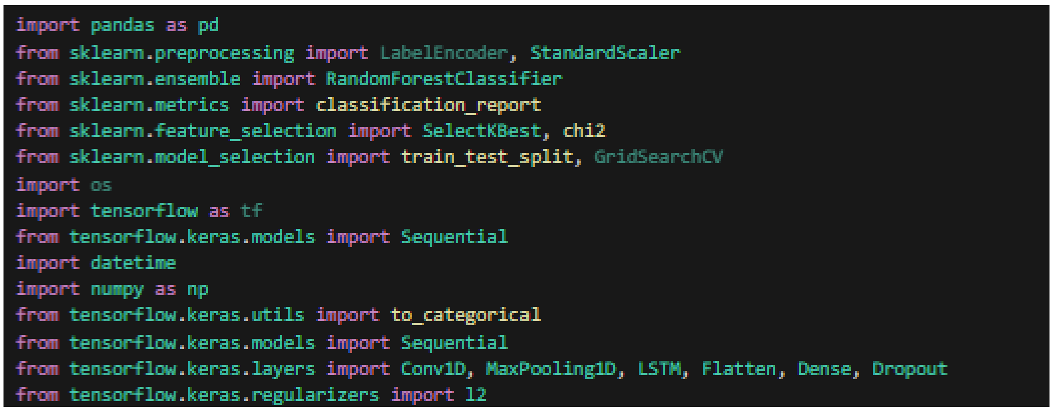
- **Software Used:** VS Code, Anaconda3, Python 3.6.

An open-source distribution for Python and R called Anaconda makes package management and data science deployment easier. It makes use of the conda package management system to control package versions. It ensures that installations don't conflict with already-existing frameworks and packages by first assessing the current environment. More than 250 pre-installed packages are included with Anaconda, and via PyPI, the conda package and virtual environment manager, and other resources, users can get approximately 7,500 additional open-source packages[2]. Python is a high-level, interpreted, object-oriented language with dynamic semantics that is frequently used for system scripting, server-side web development, software development, and mathematics. It is quite popular because to its dynamic typing, dynamic binding, and high-level built-in data structures. Because of these characteristics, it's ideal for Rapid Application Development and as a scripting language for assembling pre-made parts. Python is easier to use than other programming languages because of its English-like syntax, which enables programmers to write fewer lines of code. It is compatible with multiple operating systems, including Windows, Mac, Linux, and Raspberry Pi, and operates on an interpreter system. This makes it possible for code to run instantly and for quick prototyping. Python supports procedural, object-oriented, and functional programming approaches[3]. The free open-source code editor Visual Studio Code (VS Code) was created by Microsoft. Because it supports a large number of programming and scripting languages, it is a flexible tool for developing, debugging, and version management. With its extensive extension library and flexible interface, VS Code lets users customize the editor to suit their own requirements. It has code refactoring, snippets, intelligent code completion, syntax highlighting, and built-in support for Git. Moreover, the comprehensive plugin ecosystem and integrated terminal improve compatibility with other development tools, making VS Code a well-liked option for developers because of its effectiveness and versatility.

[2] https://domino.ai/data-science-dictionary/anaconda
[3] https://www.teradata.com/glossary/what-is-python

4 Implementation

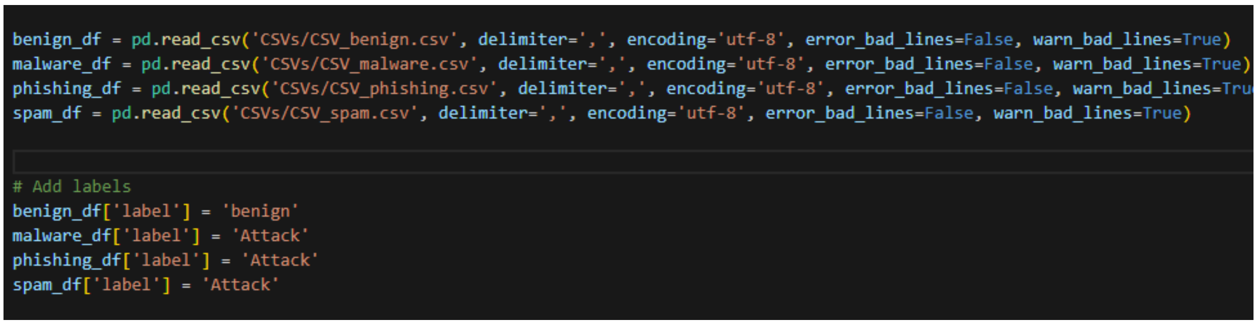
Step 1: Importing necessary library



```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split, GridSearchCV
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
import datetime
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Flatten, Dense, Dropout
from tensorflow.keras.regularizers import l2
```

Figure (1): Importing of necessary libraries

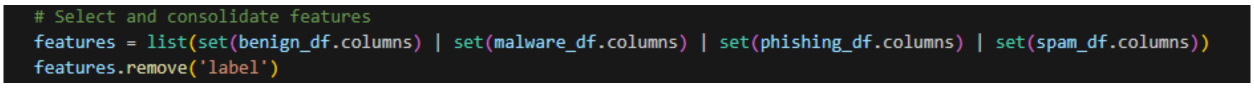
Step 2: Loading datasets



```
benign_df = pd.read_csv('CSVs/CSV_benign.csv', delimiter=',', encoding='utf-8', error_bad_lines=False, warn_bad_lines=True)
malware_df = pd.read_csv('CSVs/CSV_malware.csv', delimiter=',', encoding='utf-8', error_bad_lines=False, warn_bad_lines=True)
phishing_df = pd.read_csv('CSVs/CSV_phishing.csv', delimiter=',', encoding='utf-8', error_bad_lines=False, warn_bad_lines=Tru
spam_df = pd.read_csv('CSVs/CSV_spam.csv', delimiter=',', encoding='utf-8', error_bad_lines=False, warn_bad_lines=True)


# Add labels
benign_df['label'] = 'benign'
malware_df['label'] = 'Attack'
phishing_df['label'] = 'Attack'
spam_df['label'] = 'Attack'
```

Figure (2): Reading of datasets and adding a new column named label to all datasets

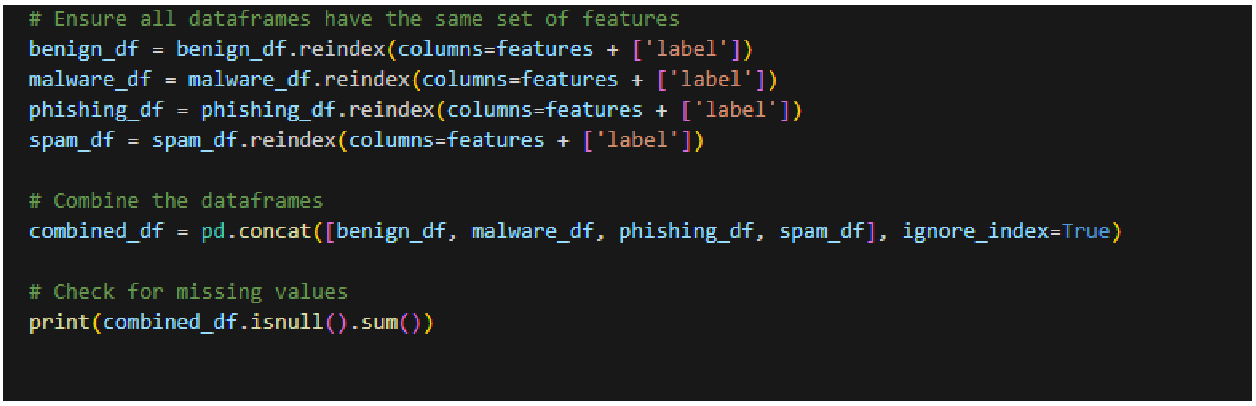
Step 3: Selecting common features in all datasets loaded



```
# Select and consolidate features
features = list(set(benign_df.columns) | set(malware_df.columns) | set(phishing_df.columns) | set(spam_df.columns))
features.remove('label')
```

Figure (3): Selections of common features from the datasets loaded

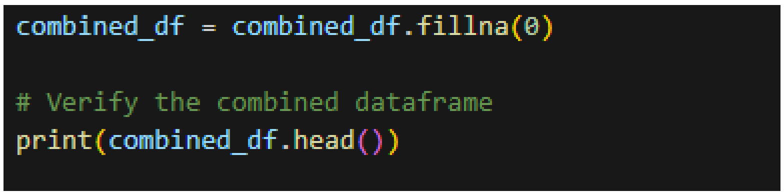
Step 4: Creation of combined dataset with common features and the label column



```python
# Ensure all dataframes have the same set of features
benign_df = benign_df.reindex(columns=features + ['label'])
malware_df = malware_df.reindex(columns=features + ['label'])
phishing_df = phishing_df.reindex(columns=features + ['label'])
spam_df = spam_df.reindex(columns=features + ['label'])

# Combine the dataframes
combined_df = pd.concat([benign_df, malware_df, phishing_df, spam_df], ignore_index=True)

# Check for missing values
print(combined_df.isnull().sum())
```

Figure (4): Creation of combined dataset with checking of null value counts if present in the dataset

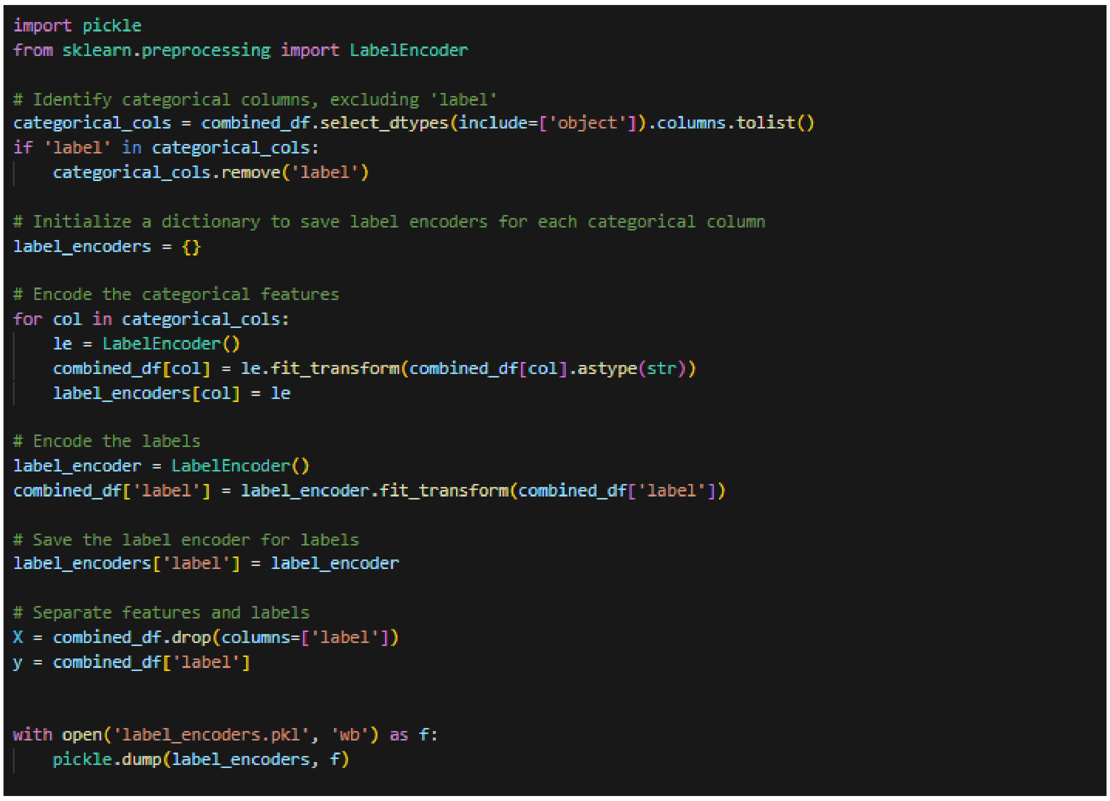
Step 5: Handling the null values present in the dataset



```python
combined_df = combined_df.fillna(0)

# Verify the combined dataframe
print(combined_df.head())
```

Figure (5): Handling of null values present in the dataset

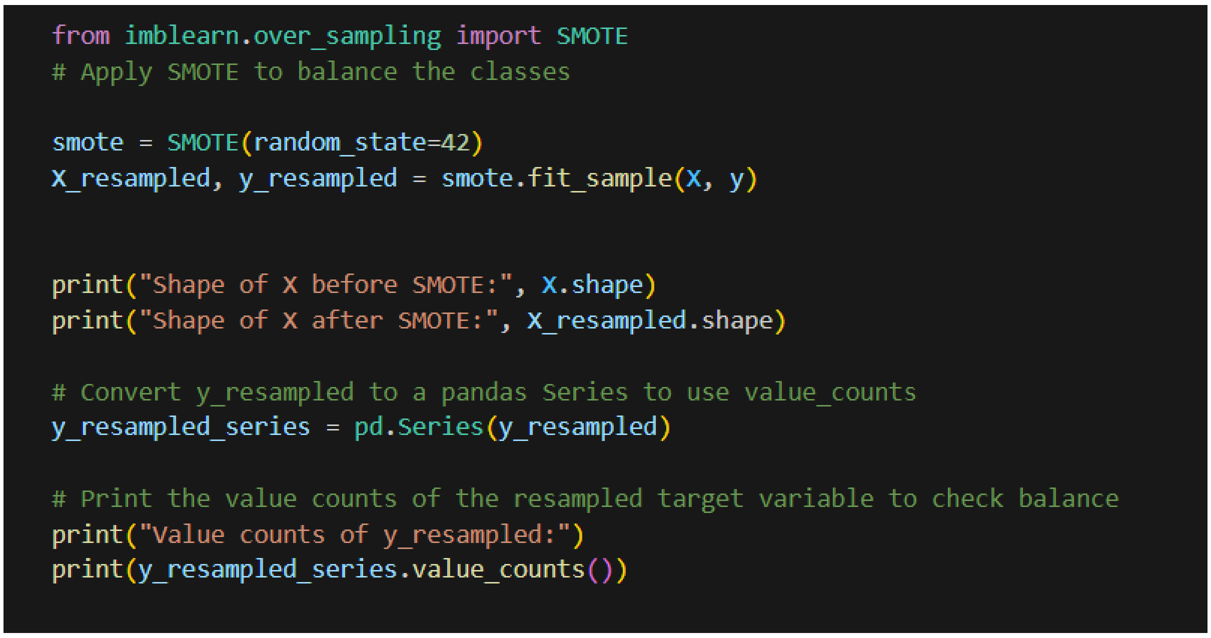
Step 6: Performing label encoder for converting categorical values to numerical for better model performance



```python
import pickle
from sklearn.preprocessing import LabelEncoder

# Identify categorical columns, excluding 'label'
categorical_cols = combined_df.select_dtypes(include=['object']).columns.tolist()
if 'label' in categorical_cols:
    categorical_cols.remove('label')

# Initialize a dictionary to save label encoders for each categorical column
label_encoders = {}

# Encode the categorical features
for col in categorical_cols:
    le = LabelEncoder()
    combined_df[col] = le.fit_transform(combined_df[col].astype(str))
    label_encoders[col] = le

# Encode the labels
label_encoder = LabelEncoder()
combined_df['label'] = label_encoder.fit_transform(combined_df['label'])

# Save the label encoder for labels
label_encoders['label'] = label_encoder

# Separate features and labels
X = combined_df.drop(columns=['label'])
y = combined_df['label']


with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)
```

Figure (6): Performing of label encoder for converting categorical values to numerical and separating features and label

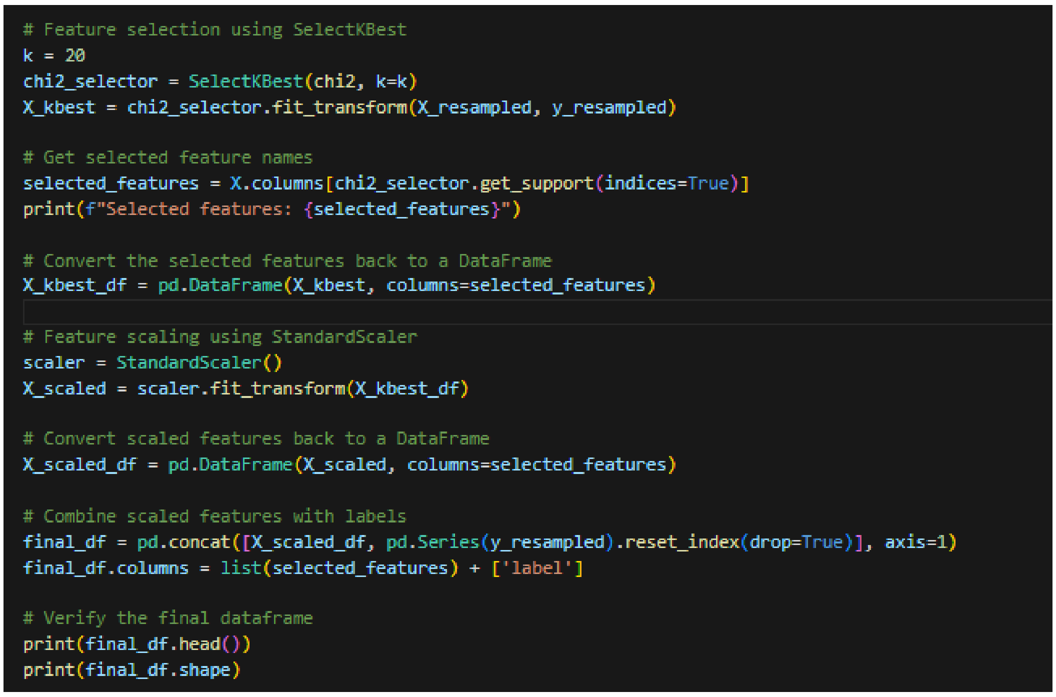
Step 7: Performing data balancing using SMOTE.



```python
from imblearn.over_sampling import SMOTE
# Apply SMOTE to balance the classes

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_sample(X, y)


print("Shape of X before SMOTE:", X.shape)
print("Shape of X after SMOTE:", X_resampled.shape)

# Convert y_resampled to a pandas Series to use value_counts
y_resampled_series = pd.Series(y_resampled)

# Print the value counts of the resampled target variable to check balance
print("Value counts of y_resampled:")
print(y_resampled_series.value_counts())
```

Figure (7): Performing of Data balancing using SMOTE

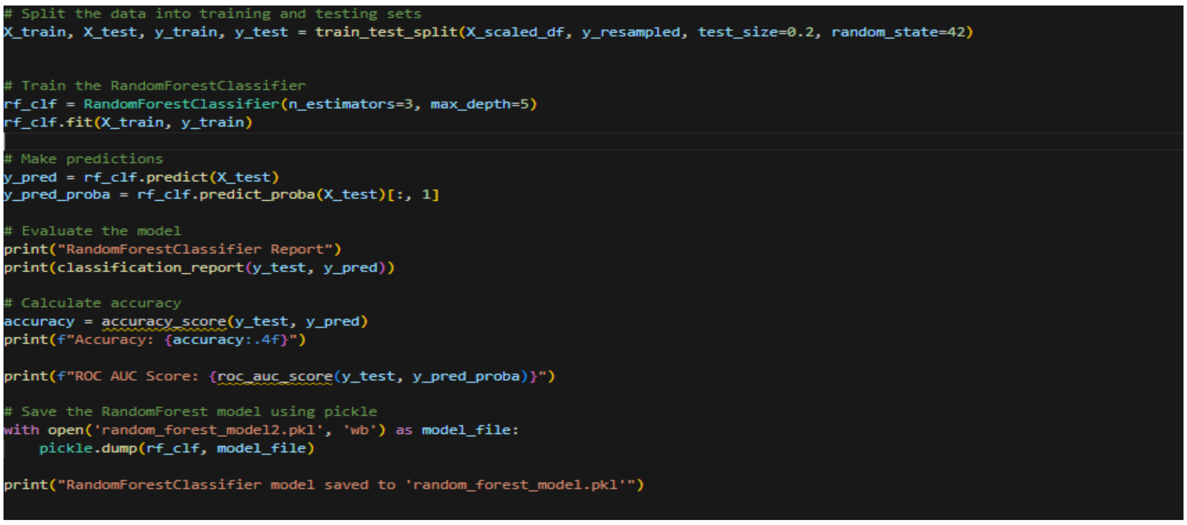
Step 8: Performing feature selection method chi2 for selecting best features in the dataset and feature scaling using standard scalar



```python
# Feature selection using SelectKBest
k = 20
chi2_selector = SelectKBest(chi2, k=k)
X_kbest = chi2_selector.fit_transform(X_resampled, y_resampled)

# Get selected feature names
selected_features = X.columns[chi2_selector.get_support(indices=True)]
print(f"Selected features: {selected_features}")

# Convert the selected features back to a DataFrame
X_kbest_df = pd.DataFrame(X_kbest, columns=selected_features)

# Feature scaling using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_kbest_df)

# Convert scaled features back to a DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=selected_features)

# Combine scaled features with labels
final_df = pd.concat([X_scaled_df, pd.Series(y_resampled).reset_index(drop=True)], axis=1)
final_df.columns = list(selected_features) + ['label']

# Verify the final dataframe
print(final_df.head())
print(final_df.shape)
```

Figure (8): Performing of feature selection method chi2 for selecting best features in the dataset and feature scaling using standard scalar

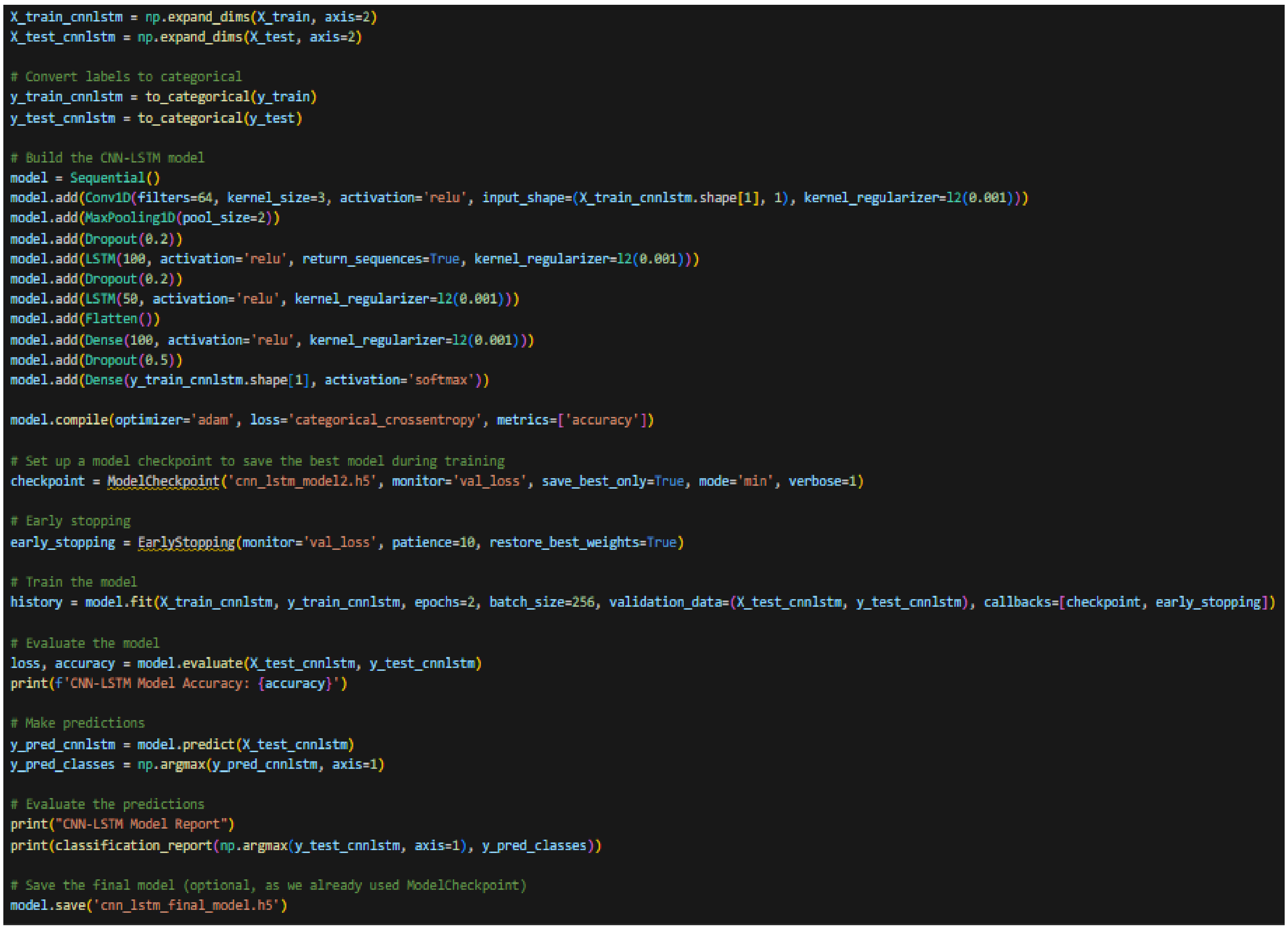
Step 9: Training of random forest classifier with model evaluation



```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y_resampled, test_size=0.2, random_state=42)


# Train the RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=3, max_depth=5)
rf_clf.fit(X_train, y_train)

# Make predictions
y_pred = rf_clf.predict(X_test)
y_pred_proba = rf_clf.predict_proba(X_test)[:, 1]

# Evaluate the model
print("RandomForestClassifier Report")
print(classification_report(y_test, y_pred))

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

print(f"ROC AUC Score: {roc_auc_score(y_test, y_pred_proba)}")

# Save the RandomForest model using pickle
with open('random_forest_model2.pkl', 'wb') as model_file:
    pickle.dump(rf_clf, model_file)

print("RandomForestClassifier model saved to 'random_forest_model.pkl'")
```

Figure (9): Training of random forest classifier with model evaluation

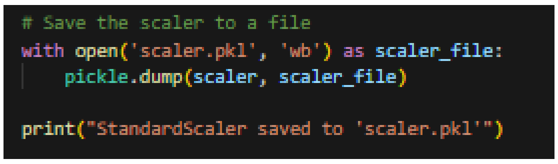
Step 10: Training of CNN-LSTM model with model evaluation



```python
X_train_cnnlstm = np.expand_dims(X_train, axis=2)
X_test_cnnlstm = np.expand_dims(X_test, axis=2)

# Convert labels to categorical
y_train_cnnlstm = to_categorical(y_train)
y_test_cnnlstm = to_categorical(y_test)

# Build the CNN-LSTM model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_cnnlstm.shape[1], 1), kernel_regularizer=l2(0.001)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))
model.add(LSTM(100, activation='relu', return_sequences=True, kernel_regularizer=l2(0.001)))
model.add(Dropout(0.2))
model.add(LSTM(50, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Flatten())
model.add(Dense(100, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(y_train_cnnlstm.shape[1], activation='softmax'))

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

# Set up a model checkpoint to save the best model during training
checkpoint = ModelCheckpoint('cnn_lstm_model2.h5', monitor='val_loss', save_best_only=True, mode='min', verbose=1)

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model
history = model.fit(X_train_cnnlstm, y_train_cnnlstm, epochs=2, batch_size=256, validation_data=(X_test_cnnlstm, y_test_cnnlstm), callbacks=[checkpoint, early_stopping])

# Evaluate the model
loss, accuracy = model.evaluate(X_test_cnnlstm, y_test_cnnlstm)
print(f'CNN-LSTM Model Accuracy: {accuracy}')

# Make predictions
y_pred_cnnlstm = model.predict(X_test_cnnlstm)
y_pred_classes = np.argmax(y_pred_cnnlstm, axis=1)

# Evaluate the predictions
print("CNN-LSTM Model Report")
print(classification_report(np.argmax(y_test_cnnlstm, axis=1), y_pred_classes))

# Save the final model (optional, as we already used ModelCheckpoint)
model.save('cnn_lstm_final_model.h5')
```

Figure (10): Shows training of CNN-LSTM model

Step 11: Saving the standard scaler



```python
# Save the scaler to a file
with open('scaler.pkl', 'wb') as scaler_file:
    pickle.dump(scaler, scaler_file)

print("StandardScaler saved to 'scaler.pkl'")
```

Figure (11): Shows saving of standard scaler