Configuration Manual

MSc Research Project

UNDERSTANDING CUSTOMER BEHAVIOUR AND FINTECH SERVICES

DHRUVI SONI

Student ID: x22235914

School of Computing

National College of Ireland

Supervisor: Dr Brian Byrne

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Dhruvi Soni	
Student ID:	X22235914	
Programme:	MSc in Fintech	**Year 2024**
Module:	Thesis	
Lecturer:	………………………………………………………………………………………….………	
Submission Due Date:	12/08/2024	
Project Title:	Understanding customer behaviour in Fintech in India	
Word Count:	-	**Page Count:**

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: Dhruvi Soni

Date: 12/08/2024

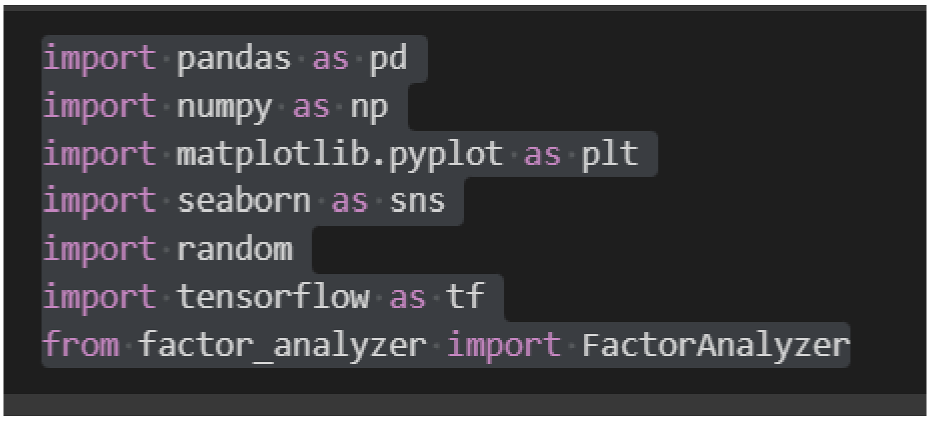
PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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.

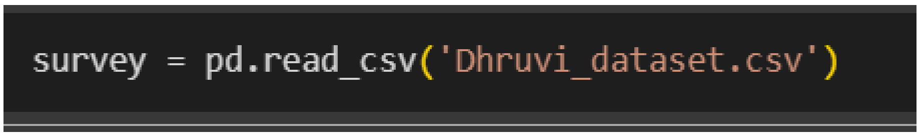
Office Use Only	
Signature:	Dhruvi Soni

Date:	12/08/2024
Penalty Applied (if applicable):	



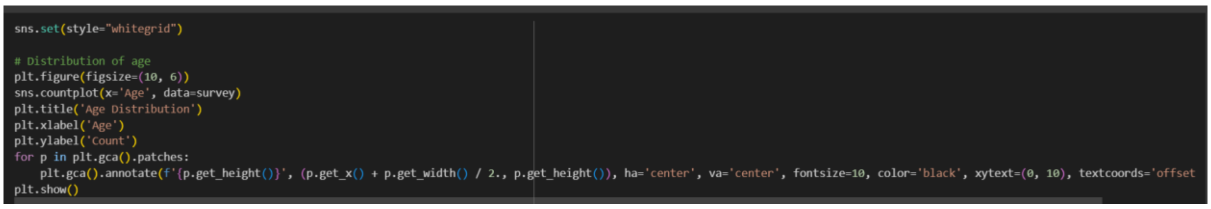
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import tensorflow as tf
from factor_analyzer import FactorAnalyzer
```

Pandas and NumPy are tools for data organization and computation. Patterns are shown in data visualization via Matplotlib and Seaborn. TensorFlow creates machine learning models, while Factor Analyzer offers a comprehensive toolkit for data analysis by revealing latent correlations between variables.



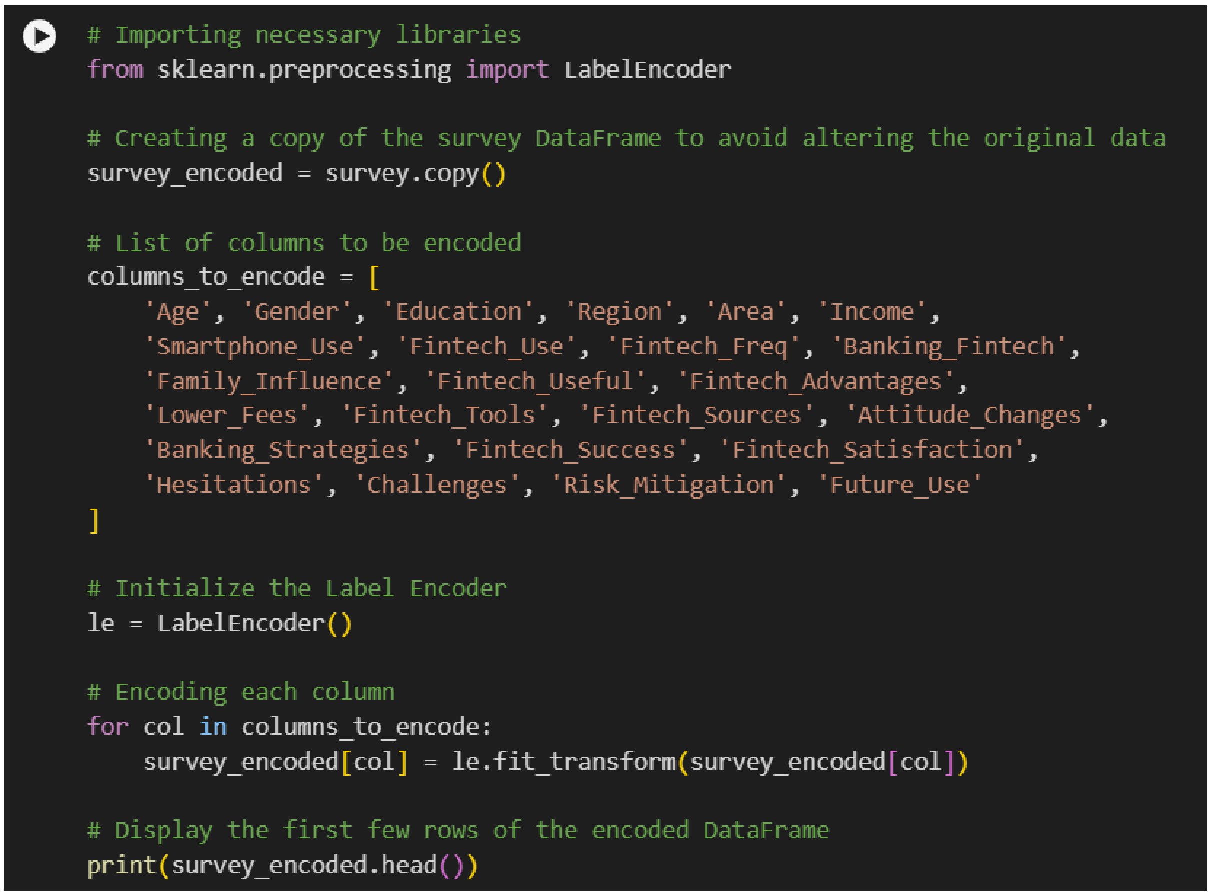
```
survey = pd.read_csv('Dhruvi_dataset.csv')
```

This is my dataset.



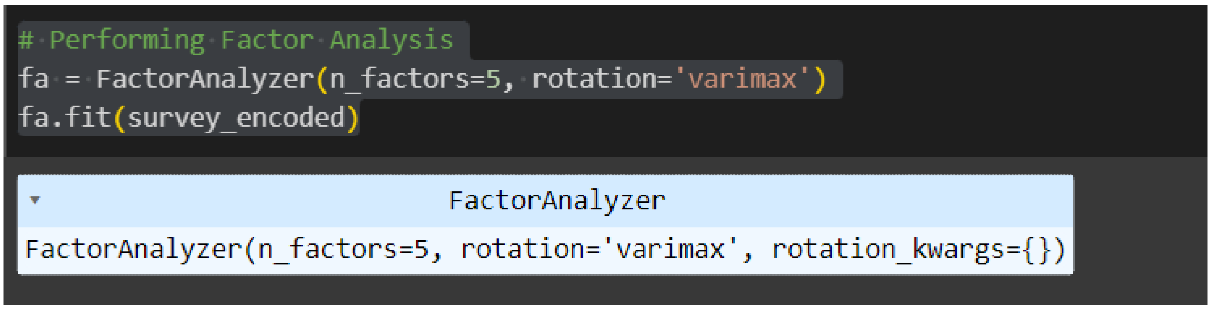
```
sns.set(style="whitegrid")

# Distribution of age
plt.figure(figsize=(10, 6))
sns.countplot(x='Age', data=survey)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='center', fontsize=10, color='black', xytext=(0, 10), textcoords='offset
plt.show()
```

This code uses a Seaborn count plot with a white grid layout to display the age distribution in the survey dataset. It highlights the data distribution and improves readability by annotating each bar with its count.



```python
# Importing necessary libraries
from sklearn.preprocessing import LabelEncoder

# Creating a copy of the survey DataFrame to avoid altering the original data
survey_encoded = survey.copy()

# List of columns to be encoded
columns_to_encode = [
    'Age', 'Gender', 'Education', 'Region', 'Area', 'Income',
    'Smartphone_Use', 'Fintech_Use', 'Fintech_Freq', 'Banking_Fintech',
    'Family_Influence', 'Fintech_Useful', 'Fintech_Advantages',
    'Lower_Fees', 'Fintech_Tools', 'Fintech_Sources', 'Attitude_Changes',
    'Banking_Strategies', 'Fintech_Success', 'Fintech_Satisfaction',
    'Hesitations', 'Challenges', 'Risk_Mitigation', 'Future_Use'
]

# Initialize the Label Encoder
le = LabelEncoder()

# Encoding each column
for col in columns_to_encode:
    survey_encoded[col] = le.fit_transform(survey_encoded[col])

# Display the first few rows of the encoded DataFrame
print(survey_encoded.head())
```

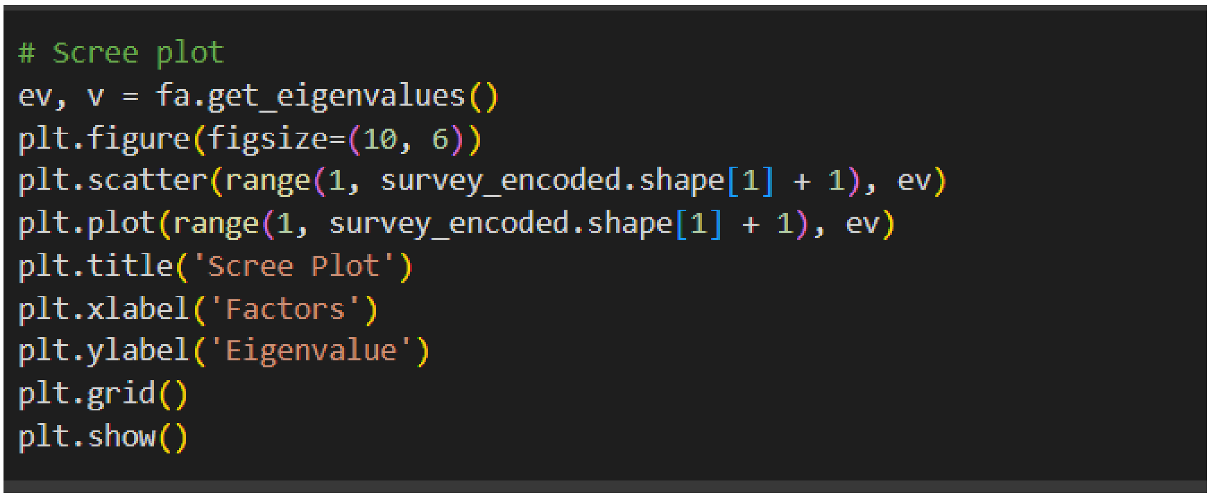
Using Label Encoder, this code converts the survey Data Frame's categorical columns into numerical values. It prepares the data for machine learning models, guaranteeing compatibility, and enhancing performance by converting text data into numerical values.



```python
# Performing Factor Analysis
fa = FactorAnalyzer(n_factors=5, rotation='varimax')
fa.fit(survey_encoded)
```

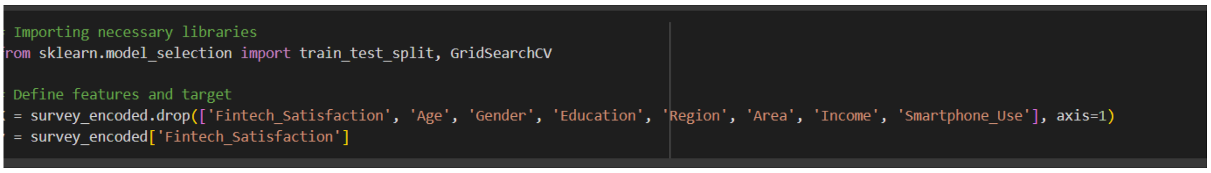
```
FactorAnalyzer
FactorAnalyzer(n_factors=5, rotation='varimax', rotation_kwargs={})
```

The survey data is analysed using this algorithm to look for hidden trends. It makes the data easier to grasp by identifying five primary elements through a process known as factor analysis.



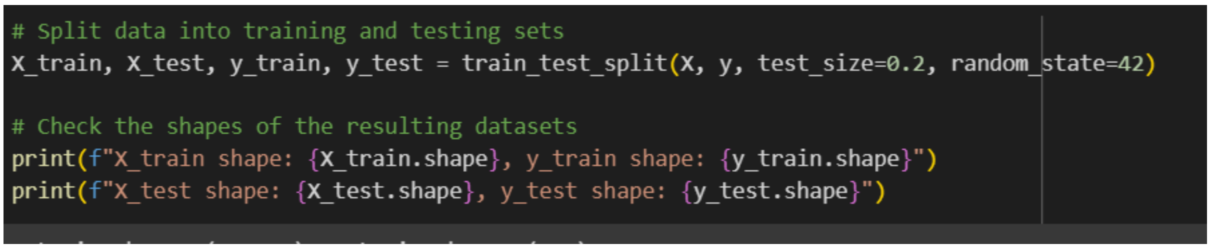
```
# Scree plot
ev, v = fa.get_eigenvalues()
plt.figure(figsize=(10, 6))
plt.scatter(range(1, survey_encoded.shape[1] + 1), ev)
plt.plot(range(1, survey_encoded.shape[1] + 1), ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```

This code creates a graph that indicates the relative importance of each factor in the analysis. Examining the points at which a factor's importance decreases can assist determine how many to retain.



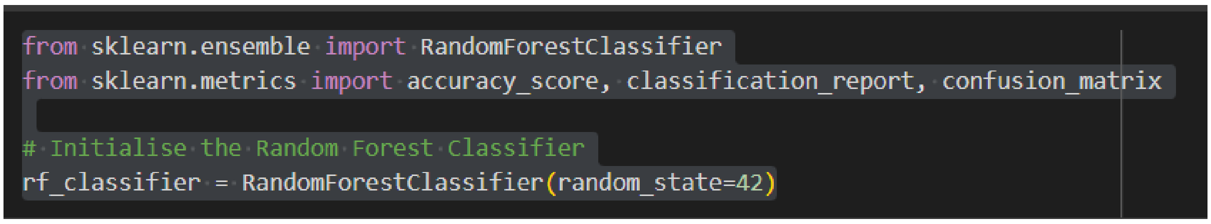
```
 Importing necessary libraries
rom sklearn.model_selection import train_test_split, GridSearchCV

 Define features and target
 = survey_encoded.drop(['Fintech_Satisfaction', 'Age', 'Gender', 'Education', 'Region', 'Area', 'Income', 'Smartphone_Use'], axis=1)
 = survey_encoded['Fintech_Satisfaction']
```

To partition data and modify models, this code imports tools. Next, it selects which columns to utilize as predictors and which column to predict to prepare the data. It ignores age and income columns in Favor of making predictions based just on the 'Fintech_Satisfaction' column.



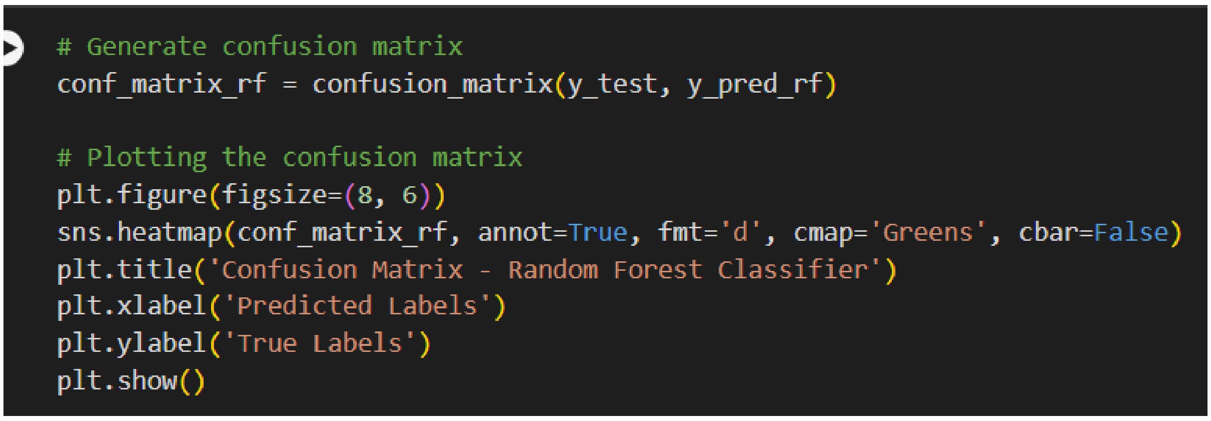
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shapes of the resulting datasets
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
```

The data is divided into training and testing sets by this code, with 80% of the data being utilized for training and 20% for testing. This aids in assessing how well the model performs with unknown data. To confirm that the split was performed correctly, the shapes of the generated datasets are printed.



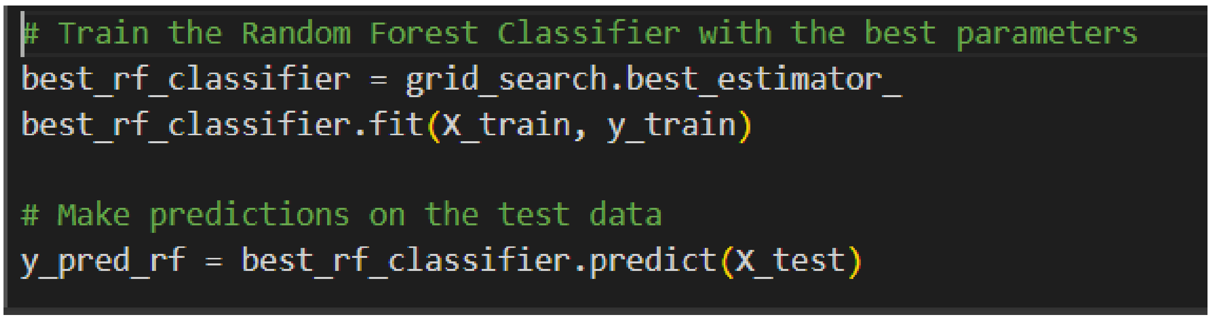
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialise the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
```

The code shown here uses a Random Forest Classifier to set up a machine learning model. To achieve consistent, repeatable results, it initializes the classifier with a fixed random state and imports evaluation metrics to evaluate model performance. Data is predicted and classified by the model.



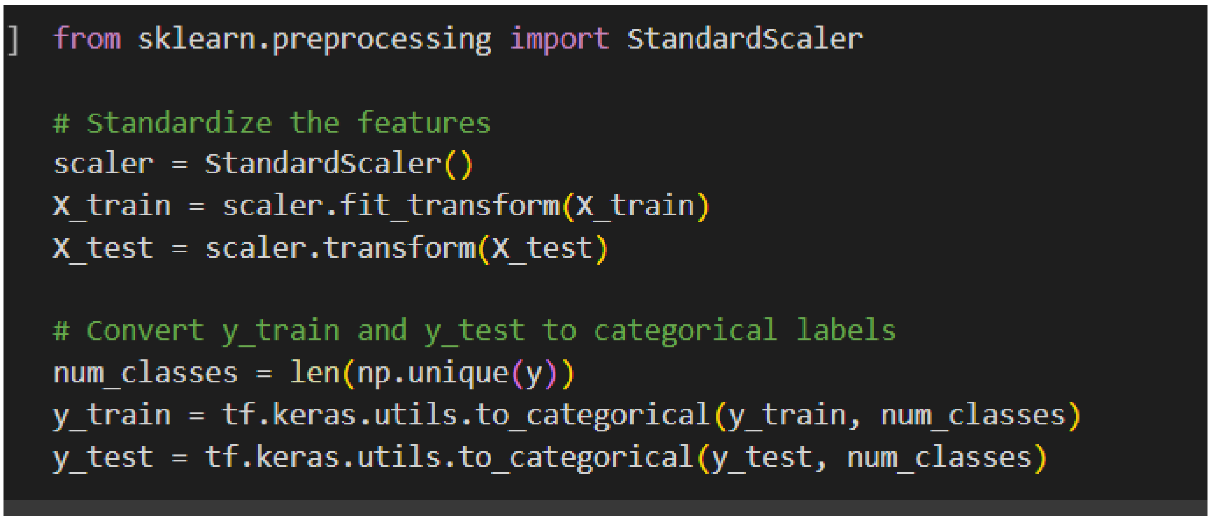
```python
# Generate confusion matrix
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Greens', cbar=False)
plt.title('Confusion Matrix - Random Forest Classifier')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

To compare the actual and predicted labels for a Random Forest Classifier, this method generates a confusion matrix. The matrix is then displayed as a green heatmap with numbers labelled in each cell. Plotting illustrates where predictions coincide with or diverge from actual labels, which aids in evaluating the model's accuracy.



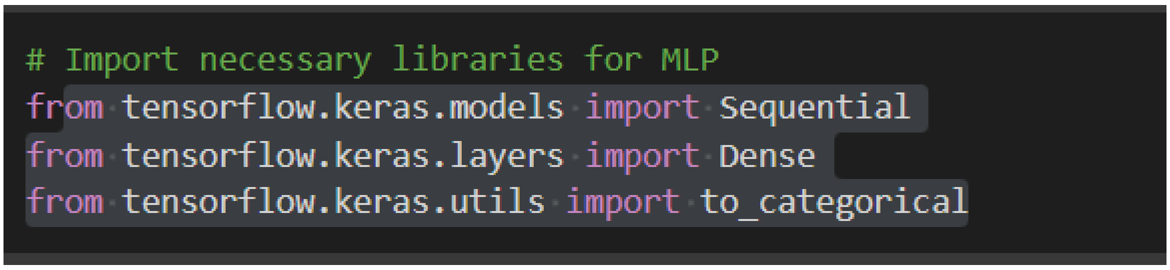
```python
# Train the Random Forest Classifier with the best parameters
best_rf_classifier = grid_search.best_estimator_
best_rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rf = best_rf_classifier.predict(X_test)
```

The Random Forest Classifier is trained with the best grid search parameters (grid_search.best_estimator_) in this code. This improved model is fitted to the X_train and y_train training data. Next, it uses the test data (X_test) to generate predictions (y_pred_rf) to assess the model's performance.



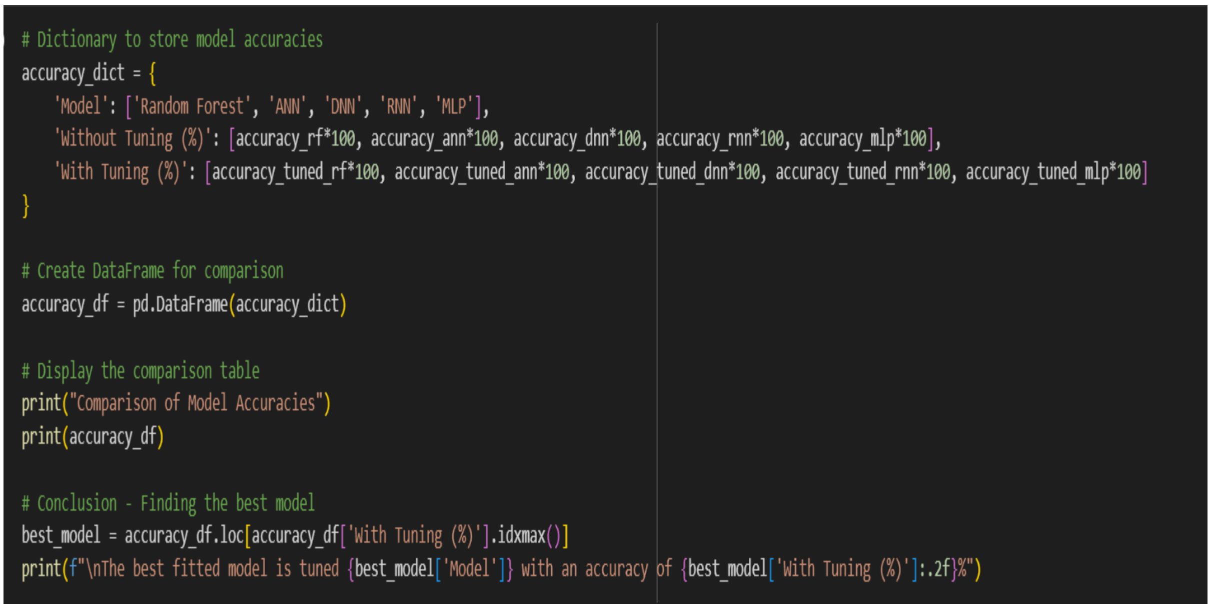
```python
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert y_train and y_test to categorical labels
num_classes = len(np.unique(y))
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

By scaling features to a mean of 0 and a standard deviation of 1, this algorithm standardizes features and guarantees consistent data ranges. Additionally, it transforms target labels into a format known as one-hot encoding, which makes them appropriate for use with neural network models that need categorical input to classify.



```python
# Import necessary libraries for MLP
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
```

Using the Keras API provided by TensorFlow, this code imports libraries to construct a Multi-Layer Perceptron (MLP). Dense adds completely linked layers, Sequential builds a linear stack of layers, and to_categorical transforms labels into a one-hot encoded format for classification applications.



```python
# Dictionary to store model accuracies
accuracy_dict = {
    'Model': ['Random Forest', 'ANN', 'DNN', 'RNN', 'MLP'],
    'Without Tuning (%)': [accuracy_rf*100, accuracy_ann*100, accuracy_dnn*100, accuracy_rnn*100, accuracy_mlp*100],
    'With Tuning (%)': [accuracy_tuned_rf*100, accuracy_tuned_ann*100, accuracy_tuned_dnn*100, accuracy_tuned_rnn*100, accuracy_tuned_mlp*100]
}

# Create DataFrame for comparison
accuracy_df = pd.DataFrame(accuracy_dict)

# Display the comparison table
print("Comparison of Model Accuracies")
print(accuracy_df)

# Conclusion - Finding the best model
best_model = accuracy_df.loc[accuracy_df['With Tuning (%)'].idxmax()]
print(f"\nThe best fitted model is tuned {best_model['Model']} with an accuracy of {best_model['With Tuning (%)']:.2f}%")
```

To store and compare model accuracies with and without tuning, this code generates a dictionary. This dictionary is transformed into a DataFrame for convenient comparison, the table is printed, and the model with the highest accuracy after tuning is identified as the best-performing model, along with its name and accuracy.