Configuration Manual

MSc Research Project

Msc Data Analytics

Rishika Poojari

Student ID: X20214201

School of Computing

National College of Ireland

Supervisor: Rejwanul Haque & John Kelly

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	RISHIKA RAMESH POOJARI
Student ID:	X20214201
Programme:	Msc Data Analytics **Year:** 2022-2023
Module:	Msc Research Project
Lecturer:	Prof. Rejwanul Haque & Prof. John Kelly
Submission Due Date:	14.08.2023
Project Title:	Enhancing Retail Strategies: An Integrated Framework for Market Basket Analysis using Apriori and MLP in Consumer Behavior Modeling
Word Count:	3062 **Page Count:** 15

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:	Rishika Poojari
Date:	12.08.2023

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

Rishika Poojari
Student ID: X20214201

1 Introduction

The "Market Basket Analysis" notebook offers a comprehensive exploration and analysis of consumer purchase patterns, aiming to unveil associations and co-occurrence relationships among different products in a retail setting. Employing sophisticated data processing techniques and modelling algorithms, this analysis provides invaluable insights into product recommendations and strategic store placements.

With a focus on merging and analysing datasets related to product orders, aisles, and departments, the notebook delves deep into the intricacies of data, revealing trends, and behaviours that are often overlooked in traditional retail analytics. The results, derived from this in-depth investigation, have the potential to shape effective marketing strategies, optimise store layouts, and enhance the overall shopping experience for customers.

The journey from raw data collection to insightful visualisations and model evaluations is meticulously detailed, ensuring clarity and reproducibility. This configuration manual is designed to assist users in seamlessly navigating and executing the notebook, ensuring they harness the full potential of the Market Basket Analysis.

2 Required Specifications

Hardware Specification

- **Actual Specification:**
    - Processor: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz 2.71 GHz
    - System Type: x64-based processor, 64-bit operating system
    - Memory: 32.0 GB RAM (31.9 GB usable)
    - Display: Normal LED Laptop Display
- **Expected/Recommended Specification:**
    - Processor: Multi-core CPU (e.g., Intel Core i7 or equivalent) for efficient data processing.
    - Memory: Minimum 16GB RAM. 32GB or higher is recommended for handling large datasets seamlessly.
    - Storage: Sufficient storage space (preferably SSD) to store datasets, intermediate files, and results.
    - System Type: 64-bit operating system with a x64-based processor for compatibility with modern software tools.

Software Specification

- **Actual Specification:**
    - Operating System: Windows 10 Pro (Edition) - Version 22H2, OS build 19045.3208, Installed on 8/10/2021
- **Python Installation:**
    - Windows/MacOS/Linux:

- Navigate to the official Python website's download page: https://www.python.org/downloads/
    - Download the latest version of Python for your operating system.
    - Run the downloaded installer.
    - For Windows: Ensure the "Add Python to PATH" option is checked before proceeding with the installation.
    - Follow the installation prompts.
    - To verify the installation, open a command prompt or terminal and type python --version. It should display the installed Python version.
- **Jupyter Notebook Installation:**
    - Installing via pip:
        - Open a command prompt or terminal.
        - Run the command pip install notebook.
        - After the installation, launch Jupyter Notebook by entering the command jupyter notebook in the terminal or command prompt. This will open a new tab in your default web browser with the Jupyter Notebook interface.
    - Installing via Anaconda (recommended for new users):
        - Anaconda is a free distribution of Python and R for scientific computing and data science. It includes many popular libraries out-of-the-box and simplifies package management.
        - Navigate to the Anaconda distribution download page: https://www.anaconda.com/products/distribution
        - Download the appropriate version for your operating system.
        - Run the downloaded installer and follow the installation prompts.
        - Once installed, you can launch Jupyter Notebook using the Anaconda Navigator GUI or by typing jupyter notebook in a terminal or Anaconda command prompt.
- Expected/Recommended Specification:
    - Operating System: Windows, macOS, or Linux - Ensuring compatibility with Python and related tools.
    - Python Environment: Python 3.x. Ensure it matches the specific version used in the notebook for maximum compatibility.
    - Notebook Environment: Jupyter Notebook or Jupyter Lab for interactive code execution and visualization.

Library Dependencies

To successfully run the "Market Basket Analysis" notebook, ensure you have the following libraries installed:

- **warnings:** Built-in module for warning control.
- **pandas:** Essential for data manipulation and analysis.
- **numpy:** For numerical operations and matrix computations.
- **seaborn:** Advanced data visualisation library based on matplotlib.
- **matplotlib:** Extensive library to create interactive, static or dynamic visualisations.
- **mlxtend.frequent_patterns:** For association rule mining, including apriori and association_rules.
- **tensorflow:** Open-source platform for machine learning.
- **tensorflow.keras:** An advanced neural networks API that runs on top of TensorFlow.
- **sklearn.model_selection:** Module for splitting datasets and cross-validation.

- **sklearn.preprocessing:** Module for data preprocessing techniques like LabelEncoder, OneHotEncoder, and StandardScaler.
- **tensorflow.keras.callbacks:** For callbacks like EarlyStopping during training.
- **sklearn.metrics:** For evaluating models using metrics like roc_curve, auc, confusion_matrix, and classification_report.

Ensure all these libraries are installed and up-to-date before executing the notebook.
You can install these libraries using pip, like:

```
pip install pandas numpy seaborn matplotlib mlxtend tensorflow
scikit-learn
```

This command includes the main libraries, but some submodules might be automatically fetched with the primary library.

3 Data Collection

- If not already a member, create a free account on Kaggle.
- Access the Instacart Market Basket Analysis competition page.
- Click on the "Data" tab on the competition page.
- Press the "Download All" button to retrieve the entire dataset as a zip file.
- Unzip the downloaded file to access the individual CSV datasets, including data on orders, products, aisles, and more.
- The data is provided by Instacart for competition purposes. Ensure you adhere to competition rules or terms of use when using the data outside of personal projects.

Data Loading and Cleaning

The code reads multiple CSV files containing order, product, aisle, and department data into individual dataframes. It then merges these dataframes into a comprehensive dataset. After merging, the code performs initial data exploration, checks for duplicates and null values, fills missing values in the 'days_since_prior_order' column with zeros, and finally saves the cleaned and combined dataset as "Combined_Instacart_Data.csv".

```
import pandas as pd
# Load the data
orders = pd.read_csv('orders.csv')
order_products = pd.read_csv('order_products__prior.csv')
products = pd.read_csv('products.csv')
aisles = pd.read_csv('aisles.csv')
departments = pd.read_csv('departments.csv')
# Merge the dataframes
merged_data = pd.merge(order_products, products, on='product_id', how='left')
merged_data = pd.merge(merged_data, aisles, on='aisle_id', how='left')
merged_data = pd.merge(merged_data, departments, on='department_id', how='left')
merged_data = pd.merge(merged_data, orders, on='order_id', how='left')
# Display the first few rows of the merged dataframe
merged_data.head()
# Printing the shape of the data
merged_data.shape
# Printing the information of the data
merged_data.info()
# Printing the statistical description of the data
merged_data.describe()
# Printing the sum of the duplicate values
```

4 Exploratory Data Analysis

The code visualises various aspects of the dataset, such as product frequencies, department distributions, reorder ratios, and order timings, using bar plots, count plots, line plots, heatmaps, and histograms to offer insights into shopping behaviours, product popularities, and order patterns.

```
import warnings
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
warnings.filterwarnings("ignore")
sns.set(style="whitegrid")

# Create the bar plot
plt.figure(figsize=(12, 6))
sns.barplot(data = top_10_product_frequency_count, x='product_name', y='frequency_count')
plt.xticks(rotation=90)
plt.xlabel('Product Name')
plt.ylabel('Frequency Count')
plt.title('Top 10 Products by Frequency Count')
# Display the plot
plt.show()

plt.figure(figsize=(12, 6))
sns.countplot(data = merged_data,x = 'department',order =
merged_data['department'].value_counts().index, palette='viridis')
plt.title("Departments Distribution", fontsize=15)
plt.xlabel("Department", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

top_products =
merged_data[merged_data['reordered']==1]['product_name'].value_counts().sort_values(ascending=False)
[:10]
plt.figure(figsize=(10,5))
sns.barplot(x=top_products.index, y=top_products.values, color='skyblue')
plt.title('Top 10 Products with Highest Reorder Ratio')
plt.xlabel('Products')
plt.ylabel('Reorder Count')
plt.xticks(rotation='vertical')
plt.show()

grouped_df = merged_data.groupby(["department"])["reordered"].mean().reset_index()
plt.figure(figsize=(12, 8))
sns.pointplot(x='department', y='reordered', data=grouped_df, color='b')
plt.ylabel('Reorder ratio', fontsize=12)
plt.xlabel('Department', fontsize=12)
plt.title("Department-wise Reorder Ratio", fontsize=15)
plt.xticks(rotation='vertical')
# Adjust transparency of the plot elements
plt.plot(grouped_df['department'], grouped_df['reordered'], 'bo-', alpha=0.8)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,8))
merged_data['department'].value_counts().plot(kind='bar', color='skyblue')
plt.title('Product Sales by Department')
plt.xlabel('Department')
plt.ylabel('Number of Products Sold')
```

```
plt.xticks(rotation=45)
plt.show()

"""
Bars: The bars represent the mean reorder ratio for each department. The height of each bar
indicates the average
proportion of items reordered in that particular department. The bars provide a visual comparison
between departments
in terms of their reorder ratio.
Error bars: The error bars are represented as vertical lines above and below each bar. They indicate
the variability or
uncertainty associated with the mean reorder ratio for each department. The length of the error bars
represents the
magnitude of the variability. The error bars provide insights into the confidence or reliability of
the mean reorder
ratio estimate."""

grouped_df = merged_data.groupby(["department"])["reordered"].mean().reset_index()
grouped_df['reordered_std'] =
merged_data.groupby(["department"])["reordered"].std().reset_index()["reordered"]
plt.figure(figsize=(12, 8))
sns.barplot(x='department', y='reordered', data=grouped_df, color='b')
plt.errorbar(x=grouped_df['department'], y=grouped_df['reordered'],
yerr=grouped_df['reordered_std'], fmt='none', color='k')
plt.ylabel('Reorder Ratio', fontsize=12)
plt.xlabel('Department', fontsize=12)
plt.title('Department-wise Reorder Ratio with Error Bars', fontsize=15)
plt.xticks(rotation='vertical')
plt.tight_layout()
plt.show()

weekday_names = {0: 'Sunday',1: 'Monday',2: 'Tuesday',3: 'Wednesday',4: 'Thursday',5: 'Friday',6:
'Saturday'}
# Calculate the number of unique orders for each day of the week
orders_per_day = merged_data.groupby('order_dow')['order_id'].apply(lambda x: len(x.unique()))
# Map numerical day of the week codes to week names
weekdays = [weekday_names[day] for day in orders_per_day.index]
# Visualization
plt.figure(figsize=(12, 6))
plt.bar(weekdays, orders_per_day)
plt.xticks(rotation='vertical')
plt.ylabel('Order Count')
plt.xlabel('Day of Week')
plt.title('Number of Unique Orders by Day of Week')
plt.show()

weekday_map = {0:'Sunday', 1:'Monday', 2:'Tuesday', 3:'Wednesday', 4:'Thursday', 5:'Friday',
6:'Saturday'}
busiest_days = merged_data['order_dow'].map(weekday_map).value_counts().loc[weekday_map.values()]
# Visualization
plt.figure(figsize=(10,5))
sns.lineplot(x=busiest_days.index, y=busiest_days.values)
plt.title('Busiest Days of The Week')
plt.ylabel('Number of Orders', fontsize=12)
plt.xlabel('Day of The Week', fontsize=12)
plt.xticks(rotation='vertical')  # Add this line if the weekday labels are overlapping
plt.show()

plt.figure(figsize=(10,5))
sns.countplot(x='order_hour_of_day', data=merged_data, color='skyblue')
plt.title('Order Distribution Across the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Orders')
```

```
plt.show()

plt.figure(figsize=(10, 5))
sns.histplot(data=merged_data, x='add_to_cart_order', bins=20, kde=True)
plt.title('Distribution of Add-to-Cart Order')
plt.xlabel('Add-to-Cart Order')
plt.ylabel('Count')
plt.show()

"""
The graph displayed is a set of bar plots, where each plot represents the distribution of products
across different aisles
within each department.
The graph provides insights into the volume of products within each department and how they are
distributed across various
aisles.
The graph allows to compare the distribution of products across aisles within each department. By
examining the heights of
the bars, we can identify the dominant aisles within a department based on the product count.
"""
colors = sns.color_palette("Set2")  # Choose a different color palette
# Get the unique departments
unique_departments = merged_data['department'].unique()
num_rows = len(unique_departments)
# Plot departments volume, split by aisles
fig, axes = plt.subplots(num_rows, 1, figsize=(12, num_rows*4))
for i, department in enumerate(unique_departments):
    ax = axes[i]
    department_df = merged_data[merged_data['department'] == department]
    aisle_counts = department_df['aisle'].value_counts().sort_values(ascending=False)
    sns.barplot(x=aisle_counts.index, y=aisle_counts.values, ax=ax, palette=colors)
    ax.set_title(f'Department: {department}')
    ax.set_xlabel('Aisle')
    ax.set_ylabel('Product Count')
    ax.set_xticklabels(aisle_counts.index, rotation=45)
plt.tight_layout()
# Display the plots
plt.show()

grouped_df = merged_data.groupby(['order_dow',
'order_hour_of_day'])['reordered'].aggregate("mean").reset_index()
grouped_df = grouped_df.pivot('order_dow', 'order_hour_of_day', 'reordered')
plt.figure(figsize=(12, 6))
sns.heatmap(grouped_df, annot=True)
plt.title("Reorder ratio of Day of week vs Hour of day")
plt.show()
```

5 Apriori Algorithm

The code identifies the top 100 frequently purchased products, transforms the data for association rule mining, and then applies the Apriori algorithm to find frequent itemsets and derive association rules based on the lift metric.

```
product_counts = merged_data.groupby('product_id')['order_id'].count().reset_index().rename(columns
= {'order_id':'frequency'})
product_counts = product_counts.sort_values('frequency',
ascending=False)[0:100].reset_index(drop=True)
product_counts.head(10)

freq_products = list(product_counts.product_id)
freq_products[1:10]
```

```
order_products = merged_data[merged_data.product_id.isin(freq_products)]
order_products.shape

basket = order_products.pivot_table(columns='product_name', values='reordered',
                                     index='order_id').reset_index().fillna(0).set_index('order_id
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket = basket.applymap(encode_units)
basket.head()

from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
basket_subset = basket[:100000]
frequent_items = apriori(basket_subset, min_support=0.01, use_colnames=True)
frequent_items.head()

rules = association_rules(frequent_items, metric='lift', min_threshold=1)
rules.sort_values('lift', ascending=False)[:5]
```

6 MLP without Association Rule

The code loads a preprocessed dataset, samples and balances it, encodes categorical features, splits the data for training, validation, and testing, then defines, compiles, and trains a multi-layer perceptron (MLP) model using TensorFlow/Keras, and finally saves the trained model.

```
import warnings
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, Flatten
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from mlxtend.frequent_patterns import apriori, association_rules
warnings.filterwarnings("ignore")
print("Starting...")
# Load the preprocessed data
data = pd.read_csv('Combined_Instacart_Data.csv')
data = data.drop('Unnamed: 0', axis=1)
print("Data loaded.")
# Sample a fraction of your data to reduce complexity
data_sample = data.sample(frac=0.1, random_state=1)
print("Sampled data.")
# Separate majority and minority classes
data_majority = data_sample[data_sample.reordered==1]
data_minority = data_sample[data_sample.reordered==0]
# Downsample majority class
data_majority_downsampled = data_majority.sample(n=len(data_minority), random_state=123)
# Combine minority class with downsampled majority class
data_balanced = pd.concat([data_majority_downsampled, data_minority])
print("Data balanced.")
# Drop the specified features
feature_set = data_balanced.drop(['order_id', 'eval_set', 'add_to_cart_order'], axis=1)
# Label encode the categorical features
le = LabelEncoder()
```

```
feature_set['product_name'] = le.fit_transform(feature_set['product_name'])
feature_set['aisle'] = le.fit_transform(feature_set['aisle'])
feature_set['department'] = le.fit_transform(feature_set['department'])
print("Categorical features label encoded.")
# Split the data into features and target
X = feature_set.drop('reordered', axis=1)
y = feature_set['reordered']
print("Data split into features and target.")
# Split the data into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
print("Data splitted into train, val and test sets.")
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import BatchNormalization
# Define the MLP model
input_shape = (X_train.shape[1],)  # Input shape based on the number of features in X_train
mlp_model = Sequential([
    Dense(256, activation='relu',input_shape=input_shape),
    Dropout(0.3),
    BatchNormalization(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    BatchNormalization(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    BatchNormalization(),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid'),
])
print("Model defined.")
# Compile the model
mlp_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
print("Model compiled.")
# Summary of model
mlp_model.summary()
# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
# Train the model
mlp_history = mlp_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50,
batch_size=64,  callbacks=[early_stopping])
print("Model trained. Done.")
# Save the trained model
mlp_model.save('mlp_model.h5')
```

Plotting the Accuracy and Loss Curve

The code visualises the accuracy and loss curves of the trained model over epochs and then computes and plots the Receiver Operating Characteristic (ROC) curve along with its Area Under the Curve (AUC) value.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
# Plot the accuracy and loss curves
history = mlp_history.history
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
# Accuracy curves
ax[0].plot(history['accuracy'], label='Train Accuracy')
ax[0].plot(history['val_accuracy'], label='Validation Accuracy')
ax[0].set_title('Accuracy Curves')
```

```
feature_set['product_name'] = le.fit_transform(feature_set['product_name'])
feature_set['aisle'] = le.fit_transform(feature_set['aisle'])
feature_set['department'] = le.fit_transform(feature_set['department'])
print("Categorical features label encoded.")
# Split the data into features and target
X = feature_set.drop('reordered', axis=1)
y = feature_set['reordered']
print("Data split into features and target.")
# Split the data into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
print("Data splitted into train, val and test sets.")
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import BatchNormalization
# Define the MLP model
input_shape = (X_train.shape[1],)  # Input shape based on the number of features in X_train
mlp_model = Sequential([
    Dense(256, activation='relu',input_shape=input_shape),
    Dropout(0.3),
    BatchNormalization(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    BatchNormalization(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    BatchNormalization(),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid'),
])
print("Model defined.")
# Compile the model
mlp_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
print("Model compiled.")
# Summary of model
mlp_model.summary()
# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
# Train the model
mlp_history = mlp_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50,
batch_size=64,  callbacks=[early_stopping])
print("Model trained. Done.")
# Save the trained model
mlp_model.save('mlp_model.h5')
```

Evaluation of the model

The code loads a saved neural network model, makes predictions on a test set, displays a classification report, and visualises the results with a confusion matrix heatmap.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report
# Load the saved model
model = load_model('mlp_model.h5')
# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = np.round(y_pred)
```

```python
# Print the classification report
print(classification_report(y_test, y_pred))
# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

7 MLP with Association Rule

The code loads and preprocesses the dataset to balance it and filter for top products, applies the Apriori algorithm to derive association rules, generates new features based on these rules, prepares the data for machine learning, defines and trains a neural network model using the TensorFlow/Keras framework, and then saves the trained model.

```python
import warnings
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, Flatten
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from mlxtend.frequent_patterns import apriori, association_rules
warnings.filterwarnings("ignore")
print("Starting...")
# Load the preprocessed data
data = pd.read_csv('Combined_Instacart_Data.csv')
data = data.drop('Unnamed: 0', axis=1)
print("Data loaded.")
# Sample a fraction of your data to reduce complexity
data_sample = data.sample(frac=0.1, random_state=1)
print("Sampled data.")
# Separate majority and minority classes
data_majority = data_sample[data_sample.reordered==1]
data_minority = data_sample[data_sample.reordered==0]
# Downsample majority class
data_majority_downsampled = data_majority.sample(n=len(data_minority), random_state=123)
# Combine minority class with downsampled majority class
data_balanced = pd.concat([data_majority_downsampled, data_minority])
print("Data balanced.")
# Determine the top N products
N = 1000  # Adjust this value as needed
top_N_products = data_balanced['product_name'].value_counts().index[:N]
# Filter your data to only include the top N products
data_sample_top_N = data_balanced[data_balanced['product_name'].isin(top_N_products)]
# Convert your data into a format suitable for association rules
basket = data_sample_top_N.pivot_table(index='order_id', columns='product_name', values='reordered',
fill_value=0)
# The encoding function
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
```

```
# Apply the encoding function
basket_sets = basket.applymap(encode_units)
print("Encoding applied.")
# Use the Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket_sets, min_support=0.01, use_colnames=True)
print("Frequent itemsets found.")
# Generate the association rules from the frequent itemsets
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
print("Association rules generated.")
# Create a new feature for each rule
for i, rule in rules.iterrows():
    data_balanced[str(rule['antecedents']) + " -> " + str(rule['consequents'])] =
data_sample['product_name'].apply(lambda x: 1 if x in rule['antecedents'] else 0)
print("New features created from rules.")
# Drop the specified features
feature_set = data_balanced.drop(['order_id', 'eval_set', 'add_to_cart_order'], axis=1)
# Label encode the categorical features
le = LabelEncoder()
feature_set['product_name'] = le.fit_transform(feature_set['product_name'])
feature_set['aisle'] = le.fit_transform(feature_set['aisle'])
feature_set['department'] = le.fit_transform(feature_set['department'])
print("Categorical features label encoded.")
# Split the data into features and target
X = feature_set.drop('reordered', axis=1)
y = feature_set['reordered']
print("Data split into features and target.")
# Split the data into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
print("Data splitted into train, val and test sets.")
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import BatchNormalization
# Define the MLP model
input_shape = (X_train.shape[1],)  # Input shape based on the number of features in X_train
ar_mlp_model = Sequential([
    Dense(256, activation='relu',input_shape=input_shape),
    Dropout(0.3),
    BatchNormalization(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    BatchNormalization(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    BatchNormalization(),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid'),
])
print("Model defined.")
```

```
# Compile the model
ar_mlp_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
print("Model compiled.")
# Summary of model
ar_mlp_model.summary()
# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
# Train the model
ar_mlp_history = ar_mlp_model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50,
batch_size=64, callbacks=[early_stopping])
print("Model trained. Done.")
# Save the model
ar_mlp_model.save('ar_mlp_model.h5')
```

Plotting the Accuracy and Loss Curve

The code visualizes the accuracy and loss progression of the trained neural network over epochs and computes and plots the Receiver Operating Characteristic (ROC) curve along with its Area Under the Curve (AUC) value for model evaluation.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
# Plot the accuracy and loss curves
history = ar_mlp_history.history
fig, ax = plt.subplots(1, 2, figsize=(15, 5))

# Accuracy curves
ax[0].plot(history['accuracy'], label='Train Accuracy')
ax[0].plot(history['val_accuracy'], label='Validation Accuracy')
ax[0].set_title('Accuracy Curves')
ax[0].set_xlabel('Epoch')
ax[0].set_ylabel('Accuracy')
ax[0].legend()
# Loss curves
ax[1].plot(history['loss'], label='Train Loss')
ax[1].plot(history['val_loss'], label='Validation Loss')
ax[1].set_title('Loss Curves')
ax[1].set_xlabel('Epoch')
ax[1].set_ylabel('Loss')
ax[1].legend()
plt.show()
# Compute the AUC curve
y_val_pred = ar_mlp_model.predict(X_val)
fpr, tpr, _ = roc_curve(y_val, y_val_pred)
roc_auc = auc(fpr, tpr)
# Plot the AUC curve
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

Evaluation of the model

The code loads a previously saved neural network model, makes predictions on the test set, outputs a classification report, and visualises the results using a confusion matrix heatmap.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report
# Load the saved model
model = load_model('ar_mlp_model.h5')
# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = np.round(y_pred)
# Print the classification report
print(classification_report(y_test, y_pred))
# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Ghemawat, S. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved from https://www.tensorflow.org/
2. Chollet, F. (2015). Keras. Retrieved from https://keras.io/
3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from https://scikit-learn.org/stable/
4. Waskom, M., Botvinnik, O., Ostblom, J., Lukauskas, S., Hobson, P., Gelbart, M., ... & de Ruiter, J. (2021). Seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021. Retrieved from https://seaborn.pydata.org/
5. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. Retrieved from https://matplotlib.org/
6. McKinney, W. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56). Retrieved from https://pandas.pydata.org/
7. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. Retrieved from https://numpy.org/