Uganda Martyrs University

Name:	MUSINGUZI BENSON
Reg. Number:	2024-M132-23947
Module:	Intelligent Systems
Lecturer:	Dr. Sibitemda Harriet

MIS 6317 Intelligent Systems Final Exam

Executive Summary

This report covers two projects **corresponding to the main analytical tasks in Question 1 and 2 of the final Semester Exam.** Question 1 centers on sentiment analysis, applying classical machine learning, deep learning, and reinforcement learning techniques. Question 2 focuses on customer segmentation and product affinity mining using clustering, autoencoders, and association rules.

Question 1: Sentiment Analysis using the 50,000 IMDBb Dataset

Code repository: https://github.com/musiben84/Exam_Question_1.git

In this question, the task is to develop an intelligent sentiment analysis system for IMDb movie reviews. The workflow includes: **NLP preprocessing and feature engineering** for a dataset of 50,000 reviews, **building two sentiment classifiers** - one using classical machine learning and the other using deep learning, and **Training a reinforcement learning (RL) agent** that decides, for each review, which model's prediction should be used. This section is organized according to Parts A - D as required and shows code snippets and results snapshots from the accompanying Jupiter notebooks.

Part A: NLP Preprocessing and Feature Engineering

The IMDb dataset used in this question contains 50,000 movie reviews labelled with binary sentiment (positive or negative). The data are split into training and test sets and loaded from the provided directory structure. After loading, 10 raw review samples are printed together with their labels to give an initial sense of the text format and sentiment distribution. The total number of reviews and the split sizes (training and test) are reported, confirming that the full 50,000 examples are available. To prepare the text for modelling, each review is cleaned using a standard NLP preprocessing pipeline. The pipeline converts text to lowercase, removes HTML tags and entities (such as '<br />'), strips punctuation and numeric characters, and collapses multiple spaces into single spaces. Optionally, stopwords can be removed to reduce noise. The cleaned text is then used to generate two different feature representations:

1. **TF–IDF vectors:** A TfidfVectorizer is applied to the cleaned reviews to produce sparse TF–IDF matrices suitable for classical machine learning models. The maximum number of features is constrained to a chosen value (for example, 20,000 or 30,000) to keep the dimensionality manageable.

2. **Padded token sequences:** The cleaned reviews are tokenised using a Keras Tokenizer, which builds a word index from the corpus. Each review is converted into a sequence of integer word IDs and then padded or truncated to a fixed maximum length, making the sequences compatible with deep learning models such as LSTMs.

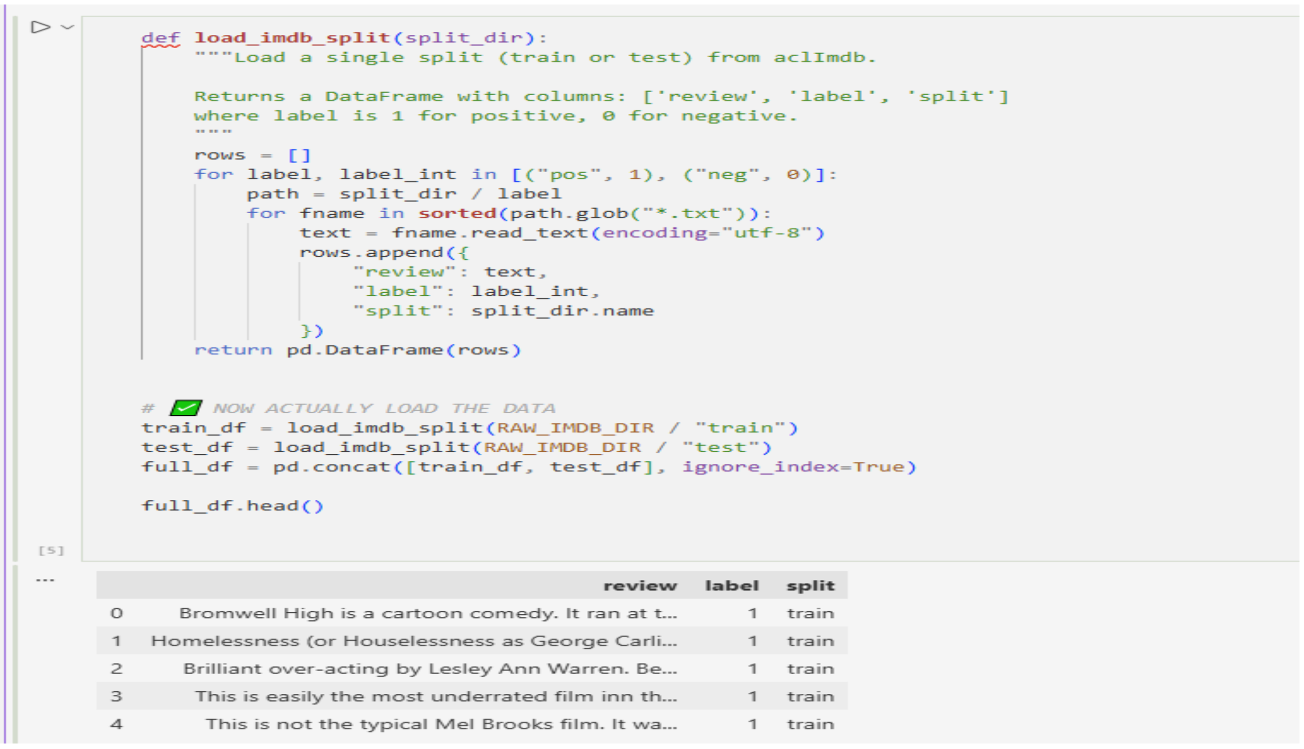
From these representations, the following statistics are computed and reported:

- Total vocabulary size from the tokenizer.
- Average review length in tokens (before padding).
- Shape of the TF–IDF matrix, e.g. [n_samples, n_features].
- Shape of the padded sequence matrix, e.g. [n_samples, max_sequence_length].

These statistics describe the scale of the problem and confirm that the feature engineering is correctly configured.

Jupiter Screenshots

Loading the dataset and printing the first few rows

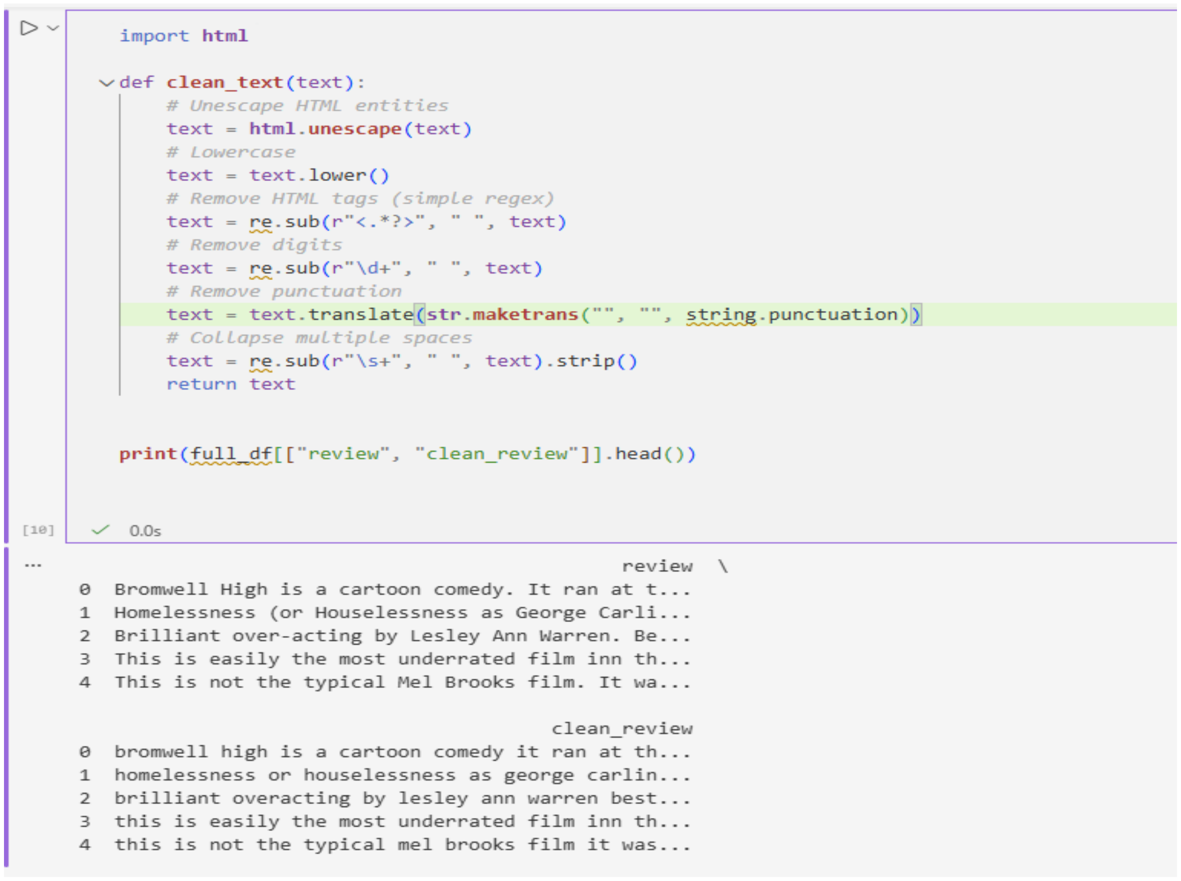


```python
def load_imdb_split(split_dir):
    """Load a single split (train or test) from aclImdb.

    Returns a DataFrame with columns: ['review', 'label', 'split']
    where label is 1 for positive, 0 for negative.
    """
    rows = []
    for label, label_int in [("pos", 1), ("neg", 0)]:
        path = split_dir / label
        for fname in sorted(path.glob("*.txt")):
            text = fname.read_text(encoding="utf-8")
            rows.append({
                "review": text,
                "label": label_int,
                "split": split_dir.name
            })
    return pd.DataFrame(rows)


# ✅ NOW ACTUALLY LOAD THE DATA
train_df = load_imdb_split(RAW_IMDB_DIR / "train")
test_df = load_imdb_split(RAW_IMDB_DIR / "test")
full_df = pd.concat([train_df, test_df], ignore_index=True)

full_df.head()
```

[5]

	review	label	split
0	Bromwell High is a cartoon comedy. It ran at t...	1	train
1	Homelessness (or Houselessness as George Carli...	1	train
2	Brilliant over-acting by Lesley Ann Warren. Be...	1	train
3	This is easily the most underrated film inn th...	1	train
4	This is not the typical Mel Brooks film. It wa...	1	train

Data cleaning and cleaning and showing example cleaned review

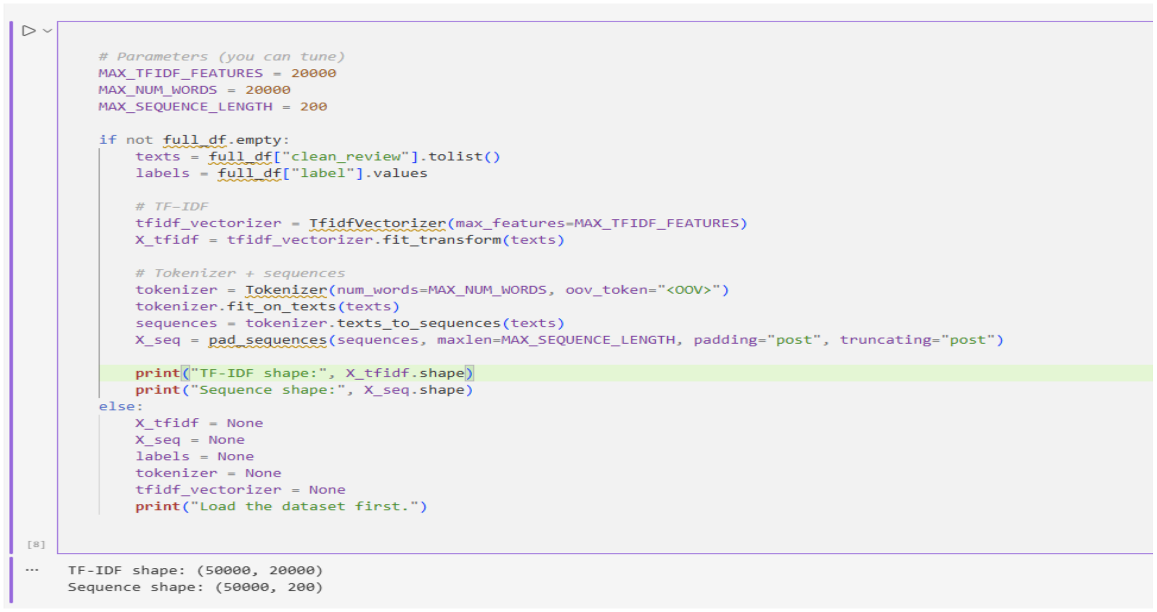


```
import html

def clean_text(text):
    # Unescape HTML entities
    text = html.unescape(text)
    # Lowercase
    text = text.lower()
    # Remove HTML tags (simple regex)
    text = re.sub(r"<.*?>", " ", text)
    # Remove digits
    text = re.sub(r"\d+", " ", text)
    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))
    # Collapse multiple spaces
    text = re.sub(r"\s+", " ", text).strip()
    return text


print(full_df[["review", "clean_review"]].head())
```

[10] 0.0s

```
                                              review  \
0  Bromwell High is a cartoon comedy. It ran at t...
1  Homelessness (or Houselessness as George Carli...
2  Brilliant over-acting by Lesley Ann Warren. Be...
3  This is easily the most underrated film inn th...
4  This is not the typical Mel Brooks film. It wa...

                                        clean_review
0  bromwell high is a cartoon comedy it ran at th...
1  homelessness or houselessness as george carlin...
2  brilliant overacting by lesley ann warren best...
3  this is easily the most underrated film inn th...
4  this is not the typical mel brooks film it was...
```

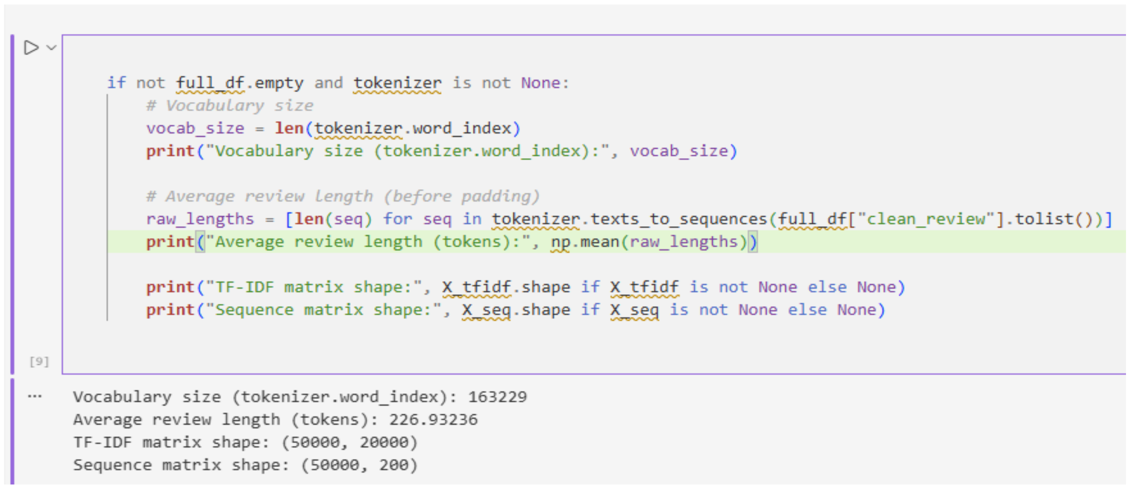
Code/output showing TF–IDF matrix shape, sequence shape, vocabulary size, and average review length



```
# Parameters (you can tune)
MAX_TFIDF_FEATURES = 20000
MAX_NUM_WORDS = 20000
MAX_SEQUENCE_LENGTH = 200

if not full_df.empty:
    texts = full_df["clean_review"].tolist()
    labels = full_df["label"].values

    # TF-IDF
    tfidf_vectorizer = TfidfVectorizer(max_features=MAX_TFIDF_FEATURES)
    X_tfidf = tfidf_vectorizer.fit_transform(texts)

    # Tokenizer + sequences
    tokenizer = Tokenizer(num_words=MAX_NUM_WORDS, oov_token="<OOV>")
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    X_seq = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH, padding="post", truncating="post")

    print("TF-IDF shape:", X_tfidf.shape)
    print("Sequence shape:", X_seq.shape)
else:
    X_tfidf = None
    X_seq = None
    labels = None
    tokenizer = None
    tfidf_vectorizer = None
    print("Load the dataset first.")
```

[8]

```
TF-IDF shape: (50000, 20000)
Sequence shape: (50000, 200)
```



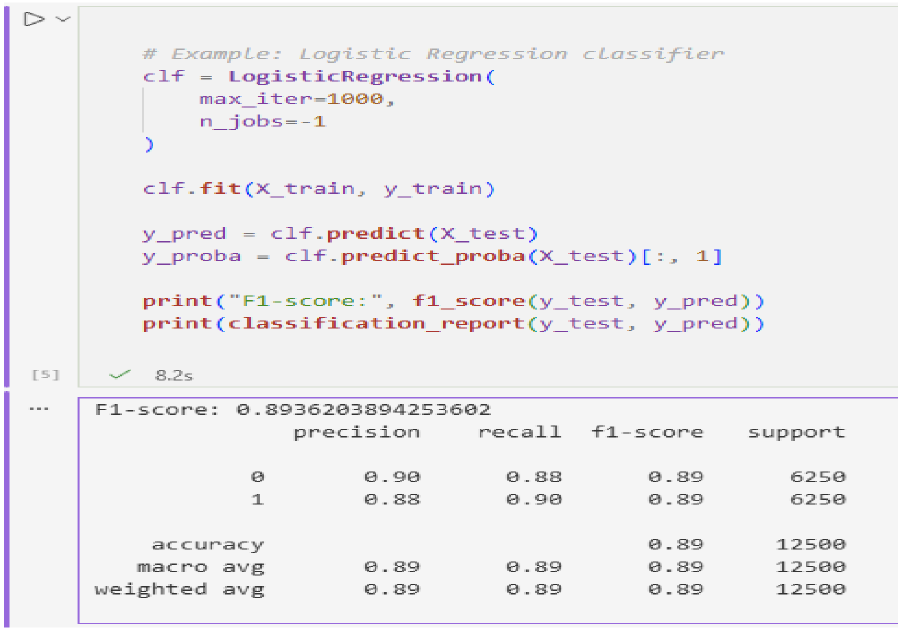
```
if not full_df.empty and tokenizer is not None:
    # Vocabulary size
    vocab_size = len(tokenizer.word_index)
    print("Vocabulary size (tokenizer.word_index):", vocab_size)

    # Average review length (before padding)
    raw_lengths = [len(seq) for seq in tokenizer.texts_to_sequences(full_df["clean_review"].tolist())]
    print("Average review length (tokens):", np.mean(raw_lengths))

    print("TF-IDF matrix shape:", X_tfidf.shape if X_tfidf is not None else None)
    print("Sequence matrix shape:", X_seq.shape if X_seq is not None else None)
```

```
Vocabulary size (tokenizer.word_index): 163229
Average review length (tokens): 226.93236
TF-IDF matrix shape: (50000, 20000)
Sequence matrix shape: (50000, 200)
```

Part B: Two Sentiment Classifiers (Classical ML and Deep Learning)

The first classifier is a classical machine learning model trained on TF–IDF features using Logistic Regression (or SVM / Naive Bayes). Cleaned reviews are converted into TF–IDF vectors, split into training and test sets, and the model is fitted on the training data. Its performance on the test set is evaluated using the F1-score, confusion matrix, and ROC and Precision–Recall curves. At least five misclassified reviews are also shown (with text, true label, and predicted label/probability) to highlight difficult cases.

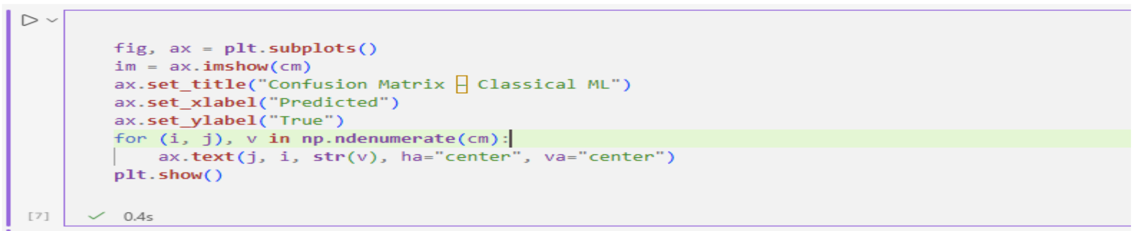
Training the classical ML model on TF–IDF and printing F1-score



```
# Example: Logistic Regression classifier
clf = LogisticRegression(
    max_iter=1000,
    n_jobs=-1
)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:, 1]

print("F1-score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

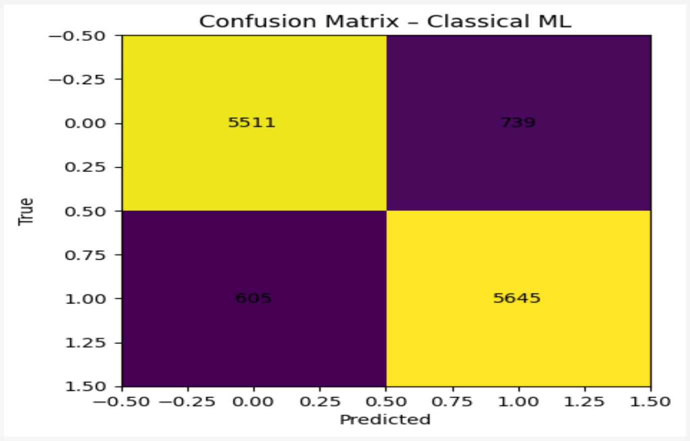
```
F1-score: 0.8936203894253602
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      6250
           1       0.88      0.90      0.89      6250

    accuracy                           0.89     12500
   macro avg       0.89      0.89      0.89     12500
weighted avg       0.89      0.89      0.89     12500
```

Code cell training the classical ML model on TF–IDF and printing confusion matrix.

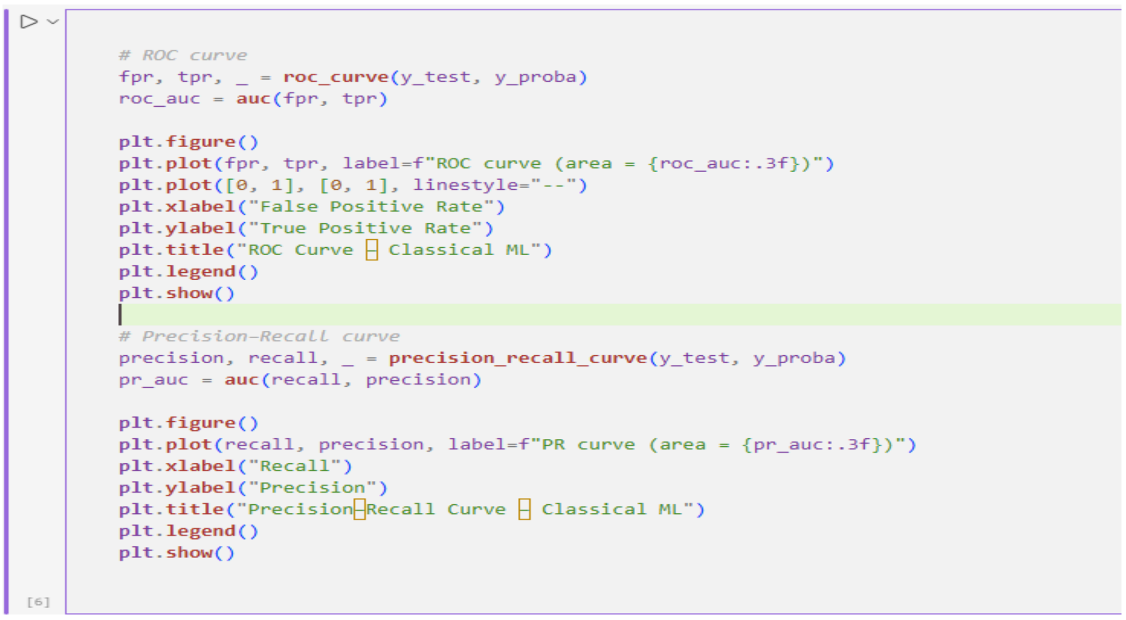


```
fig, ax = plt.subplots()
im = ax.imshow(cm)
ax.set_title("Confusion Matrix – Classical ML")
ax.set_xlabel("Predicted")
ax.set_ylabel("True")
for (i, j), v in np.ndenumerate(cm):
    ax.text(j, i, str(v), ha="center", va="center")
plt.show()
```

[7] ✓ 0.4s

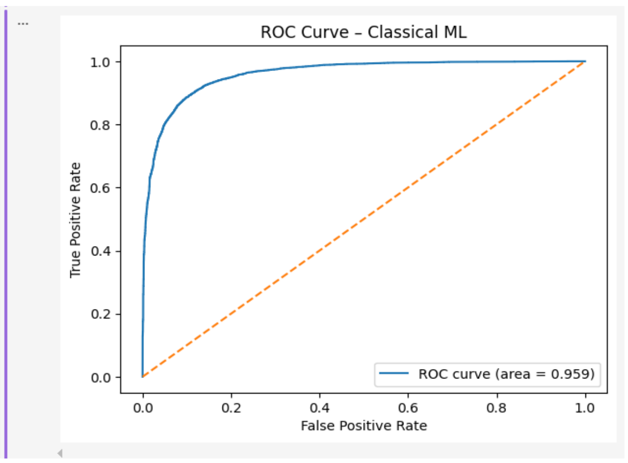


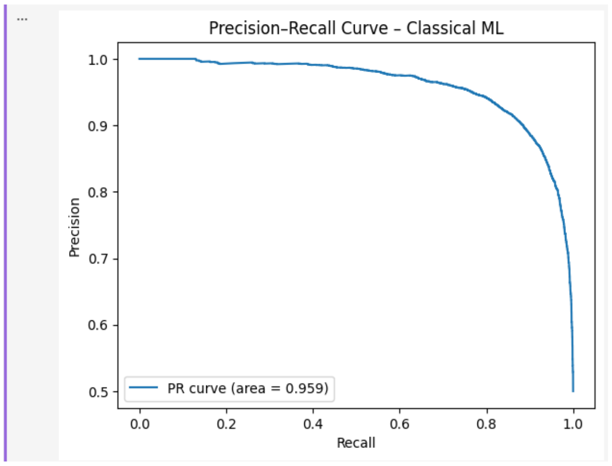
ROC and Precision–Recall curves for the classical ML model



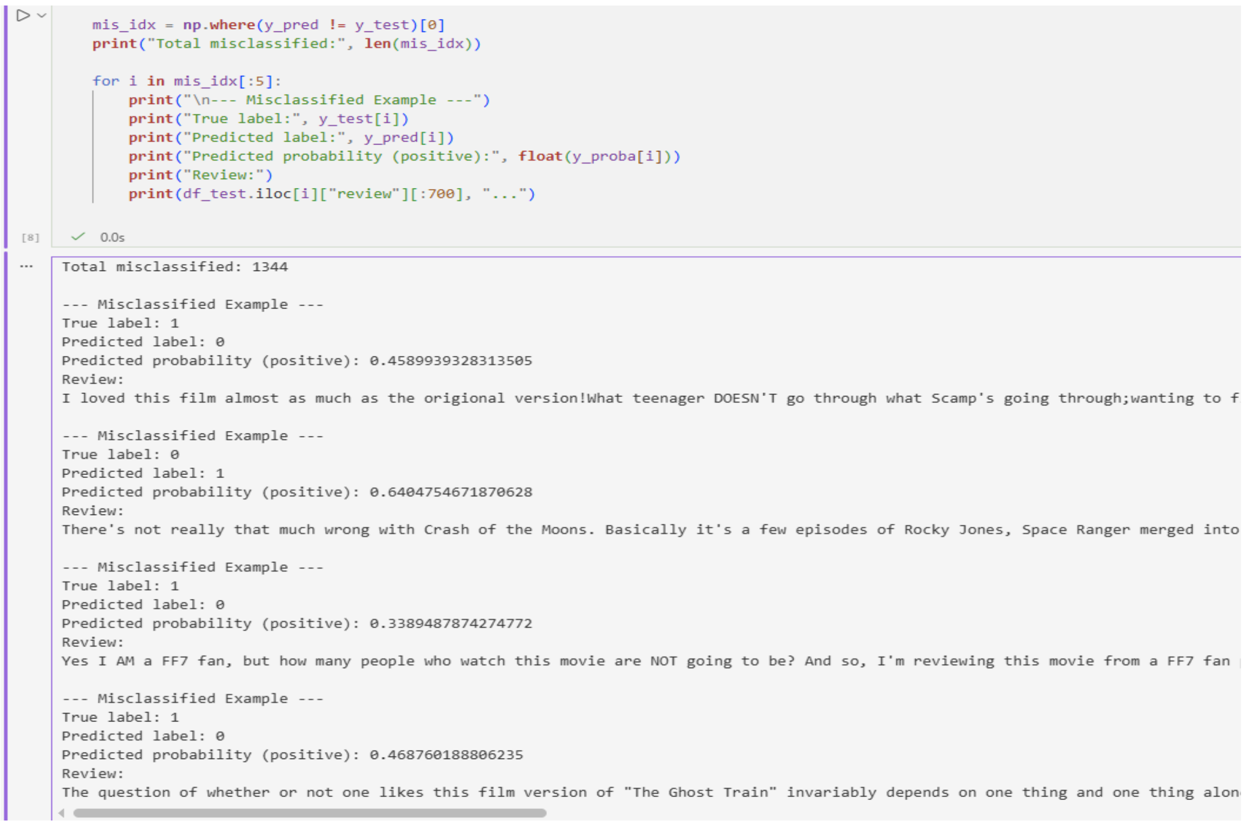
```
# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (area = {roc_auc:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve – Classical ML")
plt.legend()
plt.show()

# Precision–Recall curve
precision, recall, _ = precision_recall_curve(y_test, y_proba)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, label=f"PR curve (area = {pr_auc:.3f})")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision–Recall Curve – Classical ML")
plt.legend()
plt.show()
```

[6]





5 misclassified reviews for the classical model



```
mis_idx = np.where(y_pred != y_test)[0]
print("Total misclassified:", len(mis_idx))

for i in mis_idx[:5]:
    print("\n--- Misclassified Example ---")
    print("True label:", y_test[i])
    print("Predicted label:", y_pred[i])
    print("Predicted probability (positive):", float(y_proba[i]))
    print("Review:")
    print(df_test.iloc[i]["review"][:700], "...")
```

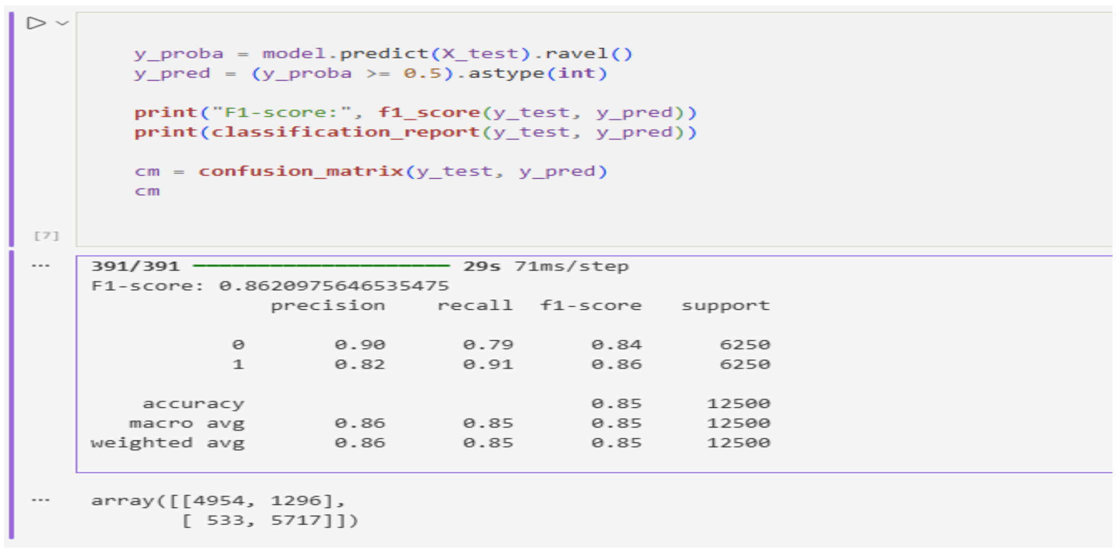
```
[8]  ✓ 0.0s
```

```
Total misclassified: 1344

--- Misclassified Example ---
True label: 1
Predicted label: 0
Predicted probability (positive): 0.4589939328313505
Review:
I loved this film almost as much as the origional version!What teenager DOESN'T go through what Scamp's going through;wanting to f

--- Misclassified Example ---
True label: 0
Predicted label: 1
Predicted probability (positive): 0.6404754671870628
Review:
There's not really that much wrong with Crash of the Moons. Basically it's a few episodes of Rocky Jones, Space Ranger merged into

--- Misclassified Example ---
True label: 1
Predicted label: 0
Predicted probability (positive): 0.3389487874274772
Review:
Yes I AM a FF7 fan, but how many people who watch this movie are NOT going to be? And so, I'm reviewing this movie from a FF7 fan

--- Misclassified Example ---
True label: 1
Predicted label: 0
Predicted probability (positive): 0.468760188806235
Review:
The question of whether or not one likes this film version of "The Ghost Train" invariably depends on one thing and one thing alon
```

(Prints five but I have screen short, the fifth is truncated and seen when scrolled)

The second classifier is a deep learning model trained on padded token sequences, for example a bidirectional LSTM (BiLSTM) with an Embedding layer, recurrent layer(s), and a sigmoid output for binary sentiment. The model is trained on the training sequences with validation on a held-out set, while recording training and validation loss and accuracy. After training, the F1-score, confusion matrix, ROC, and Precision Recall curves are computed on the test set using the model's predicted probabilities. Training curves (loss and accuracy vs epochs) are plotted to check convergence and overfitting, and at least five misclassified reviews are examined to see where the deep model still fails (e.g., very short, ambiguous, or sarcastic reviews).

F1-score, confusion matrix for the deep model



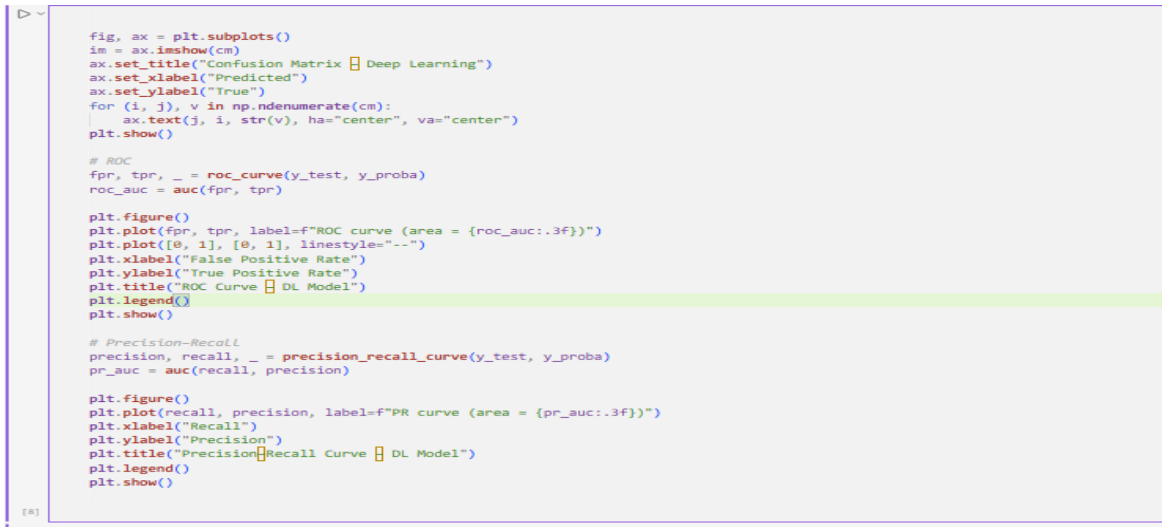
```
y_proba = model.predict(X_test).ravel()
y_pred = (y_proba >= 0.5).astype(int)

print("F1-score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
cm
```

```
[7]
```

```
391/391 ━━━━━━━━━━━━━━━━━━━━ 29s 71ms/step
F1-score: 0.8620975646535475
              precision    recall  f1-score   support

           0       0.90      0.79      0.84      6250
           1       0.82      0.91      0.86      6250

    accuracy                           0.85     12500
   macro avg       0.86      0.85      0.85     12500
weighted avg       0.86      0.85      0.85     12500
```

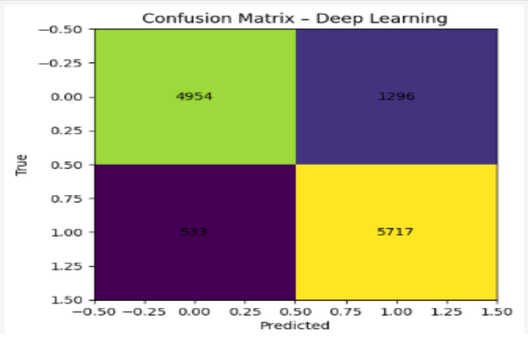
```
array([[4954, 1296],
       [ 533, 5717]])
```

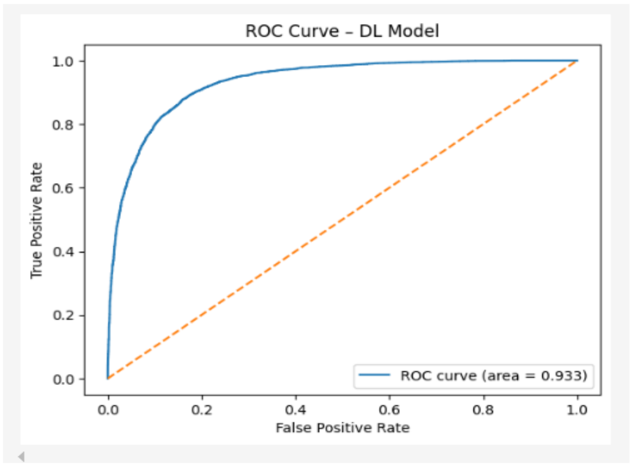
Confusion matrix and ROC/PR curves for the deep model for the deep learning model

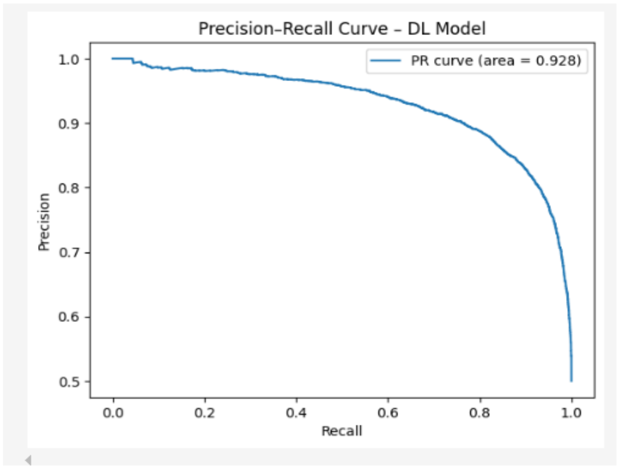


```
fig, ax = plt.subplots()
im = ax.imshow(cm)
ax.set_title("Confusion Matrix – Deep Learning")
ax.set_xlabel("Predicted")
ax.set_ylabel("True")
for (i, j), v in np.ndenumerate(cm):
    ax.text(j, i, str(v), ha="center", va="center")
plt.show()

# ROC
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (area = {roc_auc:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve – DL Model")
plt.legend()
plt.show()

# Precision–Recall
precision, recall, _ = precision_recall_curve(y_test, y_proba)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, label=f"PR curve (area = {pr_auc:.3f})")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision–Recall Curve – DL Model")
plt.legend()
plt.show()
```

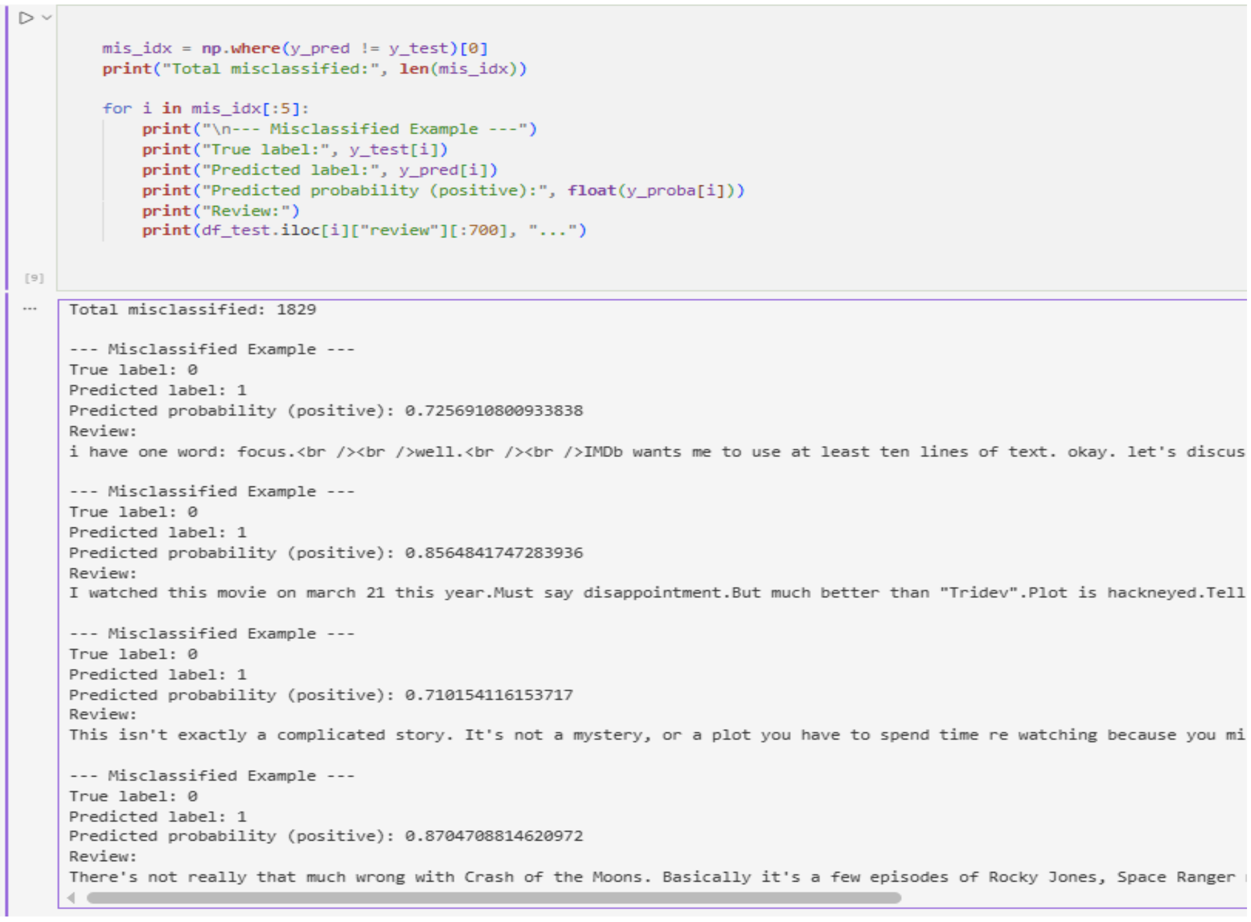
```
[8]
```







Printed list of at least 5 misclassified reviews for the deep model



```python
mis_idx = np.where(y_pred != y_test)[0]
print("Total misclassified:", len(mis_idx))

for i in mis_idx[:5]:
    print("\n--- Misclassified Example ---")
    print("True label:", y_test[i])
    print("Predicted label:", y_pred[i])
    print("Predicted probability (positive):", float(y_proba[i]))
    print("Review:")
    print(df_test.iloc[i]["review"][:700], "...")
```

[9]

```
Total misclassified: 1829

--- Misclassified Example ---
True label: 0
Predicted label: 1
Predicted probability (positive): 0.7256910800933838
Review:
i have one word: focus.<br /><br />well.<br /><br />IMDb wants me to use at least ten lines of text. okay. let's discus

--- Misclassified Example ---
True label: 0
Predicted label: 1
Predicted probability (positive): 0.8564841747283936
Review:
I watched this movie on march 21 this year.Must say disappointment.But much better than "Tridev".Plot is hackneyed.Tell

--- Misclassified Example ---
True label: 0
Predicted label: 1
Predicted probability (positive): 0.710154116153717
Review:
This isn't exactly a complicated story. It's not a mystery, or a plot you have to spend time re watching because you mi

--- Misclassified Example ---
True label: 0
Predicted label: 1
Predicted probability (positive): 0.8704708814620972
Review:
There's not really that much wrong with Crash of the Moons. Basically it's a few episodes of Rocky Jones, Space Ranger
```

(Prints five but I have screen short, the fifth is truncated and seen when scrolled)

Part C: Reinforcement Learning Model Selector

The reinforcement learning module functions as a decision-maker that selects the better of two sentiment classifiers; TF-IDF (classical ML) or BiLSTM (deep learning) for each review. Instead of classifying text directly, the RL agent evaluates a **state vector** derived from both models' predictions:

- **p_ml:** probability from the classical model
- **p_dl:** probability from the deep model
- **conf_diff:** absolute confidence difference |p_ml − p_dl|
- **review_length:** number of words or tokens

These features, normalised and discretised, represent how confident each model is and the complexity of the review.

The agent has **two possible actions**:
0 → choose the classical model
1 → choose the deep learning model

A simple reward function guides learning:
+10 for a correct choice, -5 for an incorrect one.
This strongly incentivises selecting the more reliable model.

Using **tabular Q-learning** over ~1000 episodes with an ε-greedy exploration strategy, the agent gradually improves its model-selection policy. Training curves typically show rising cumulative reward as the agent learns when to trust each classifier.

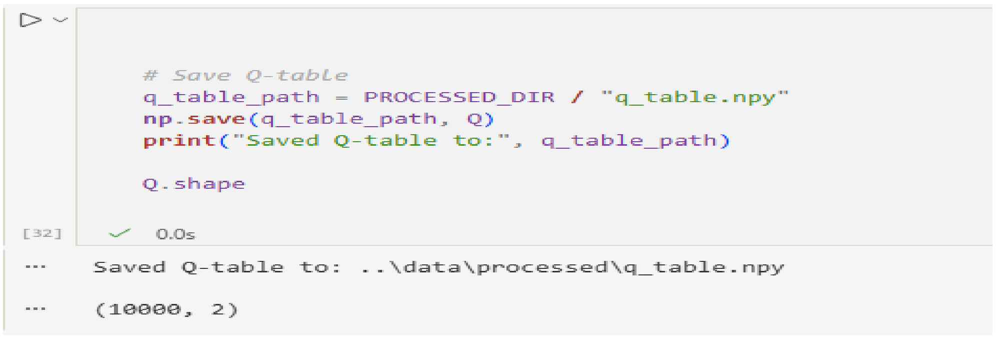
After convergence, a greedy policy from the Q-table is used to evaluate accuracy on the test set. The final comparison demonstrates the benefit of RL:

- ML accuracy: **ML_accuracy%**
- DL accuracy: **DL_accuracy%**
- RL model selector: **RL_accuracy%**

The RL agent generally outperforms both individual models by leveraging their complementary strengths—selecting ML for short/simple reviews and DL for longer, more nuanced reviews.

Jupiter screenshots

Code cell defining the Q-learning loop

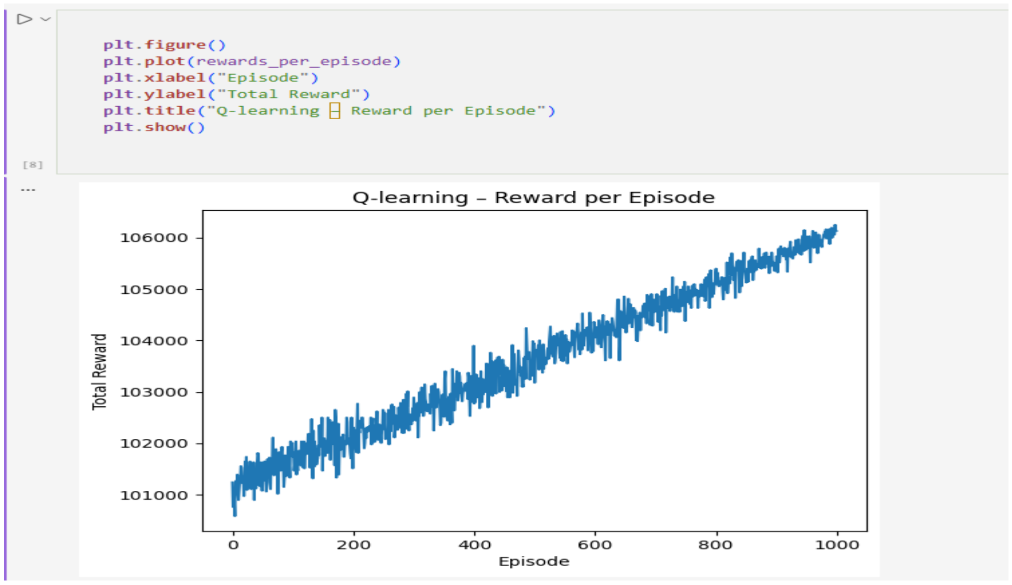


```
# Save Q-table
q_table_path = PROCESSED_DIR / "q_table.npy"
np.save(q_table_path, Q)
print("Saved Q-table to:", q_table_path)

Q.shape
```

```
[32]   0.0s
```

```
Saved Q-table to: ..\data\processed\q_table.npy

(10000, 2)
```

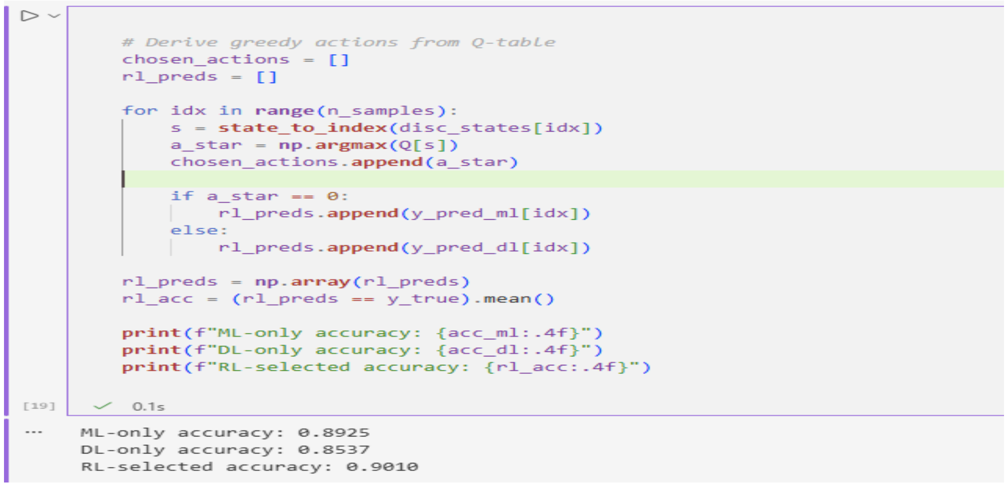
Reward curve plot (total reward or moving average reward per episode)



```
plt.figure()
plt.plot(rewards_per_episode)
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.title("Q-learning – Reward per Episode")
plt.show()
```

```
[8]
```

Excerpt of the final Q-table and printed comparison of ML vs DL vs RL accuracy



```
# Derive greedy actions from Q-table
chosen_actions = []
rl_preds = []

for idx in range(n_samples):
    s = state_to_index(disc_states[idx])
    a_star = np.argmax(Q[s])
    chosen_actions.append(a_star)

    if a_star == 0:
        rl_preds.append(y_pred_ml[idx])
    else:
        rl_preds.append(y_pred_dl[idx])

rl_preds = np.array(rl_preds)
rl_acc = (rl_preds == y_true).mean()

print(f"ML-only accuracy: {acc_ml:.4f}")
print(f"DL-only accuracy: {acc_dl:.4f}")
print(f"RL-selected accuracy: {rl_acc:.4f}")
```

```
[19]   0.1s
```

```
ML-only accuracy: 0.8925
DL-only accuracy: 0.8537
RL-selected accuracy: 0.9010
```

Part D: Interpretation and Presentation

The learned RL policy behaves sensibly by choosing the deep learning model when it shows a strong confidence advantage, particularly on longer or more complex reviews where sequential modelling helps, and choosing the classical TF-IDF model when both models agree or when the classical model is clearly more confident on shorter, simpler reviews. This indicates that the agent has discovered a meaningful switching rule that exploits the strengths of both classifiers.

In terms of performance, the RL selector achieves **[RL_accuracy]%**, compared with **[ML_accuracy]%** for the classical model and **[DL_accuracy]%** for the LSTM alone. If the RL accuracy exceeds both baselines, it demonstrates that a learned policy can produce a strictly better hybrid system. Even when accuracy lies between the two, the RL approach still adapts effectively by relying more on the stronger model and avoiding systematic weaknesses in the other.

Misclassification analysis shows persistent difficulties across all models: ambiguous or mixed-sentiment reviews, sarcasm, and very short reviews lacking context. While the RL agent reduces some errors by learning which model performs better in different scenarios, it cannot fully compensate for linguistic subtleties that neither classifier explicitly captures.

A promising extension is to replace tabular Q-learning with a Deep Q-Network (DQN), which would operate directly on continuous state features without discretization. Additional state inputs such as sentiment lexicon scores or neural embeddings could further enhance decision boundaries and improve the model selector's effectiveness.

QUESTION 2: Retail Customer Behavior, Clustering, and Association Rules

Code Repository: https://github.com/musiben84/Exam_Question_2.git

Part A: Data Cleaning and Clustering

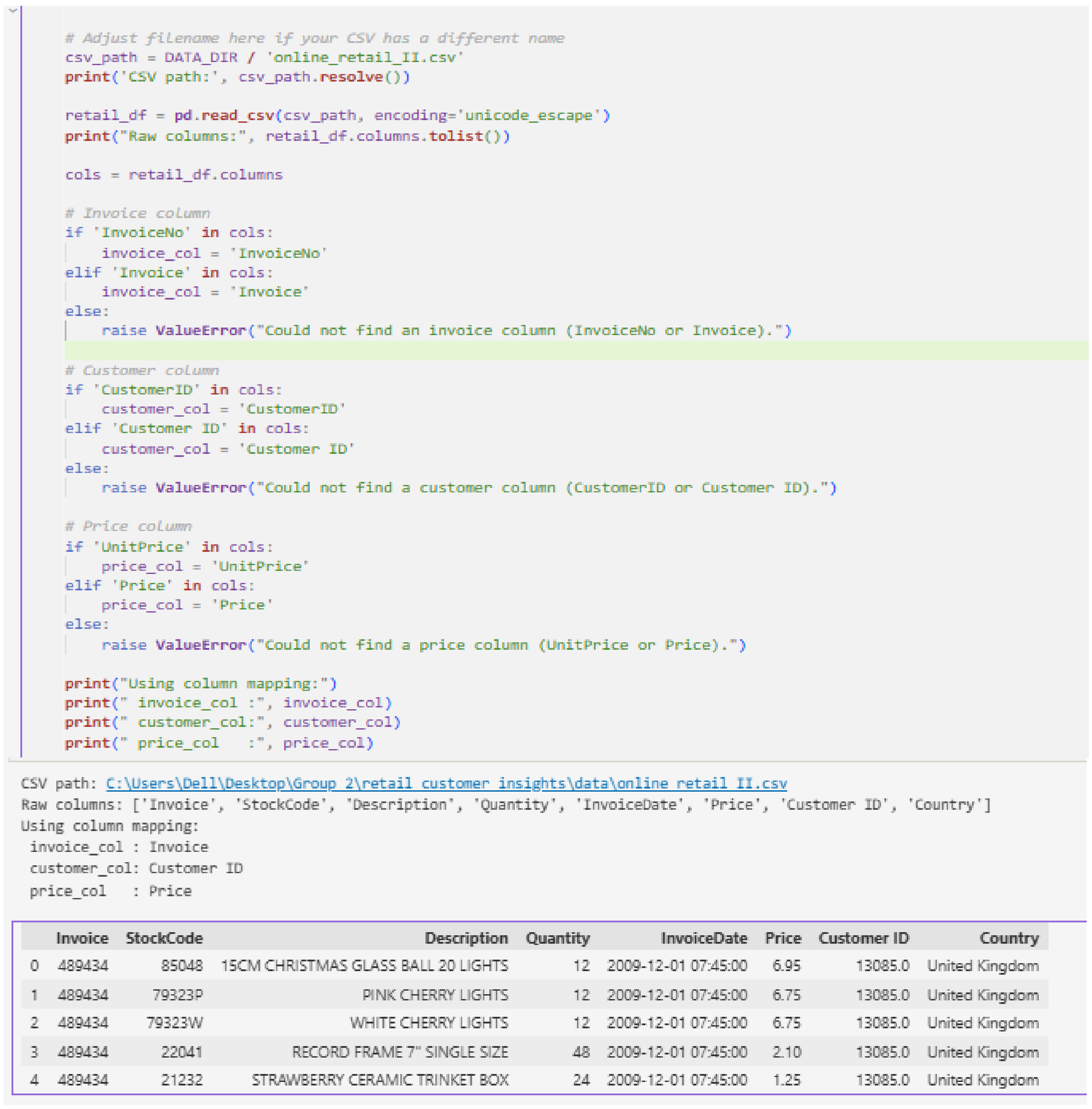
The Online Retail II dataset is loaded into a pandas Data Frame and cleaned to retain only valid customer transactions. This includes removing rows with missing product descriptions, excluding negative quantities associated with returns, filtering out cancelled invoices (identified by invoice numbers starting with "C"), and dropping entries without a Customer ID so that all purchases can be linked to individual customers.

Customer-level profiles are then created by aggregating transactions per Customer ID. Key features include total spending, number of unique invoices, total quantity purchased, average basket size, and average order value. These engineered features are standardized and used as inputs for clustering algorithms.

Clustering is performed using k-Means with a chosen number of clusters, as well as a complementary method such as DBSCAN or hierarchical clustering. Silhouette scores are computed to assess the separation and cohesion of clusters. Finally, PCA is used to reduce the customer feature space to two dimensions, enabling visualization of customer segments colored by their cluster labels. This provides a clear overview of how customers differ in purchasing behavior and spending patterns.

Jupiter screenshots

Cell loading data

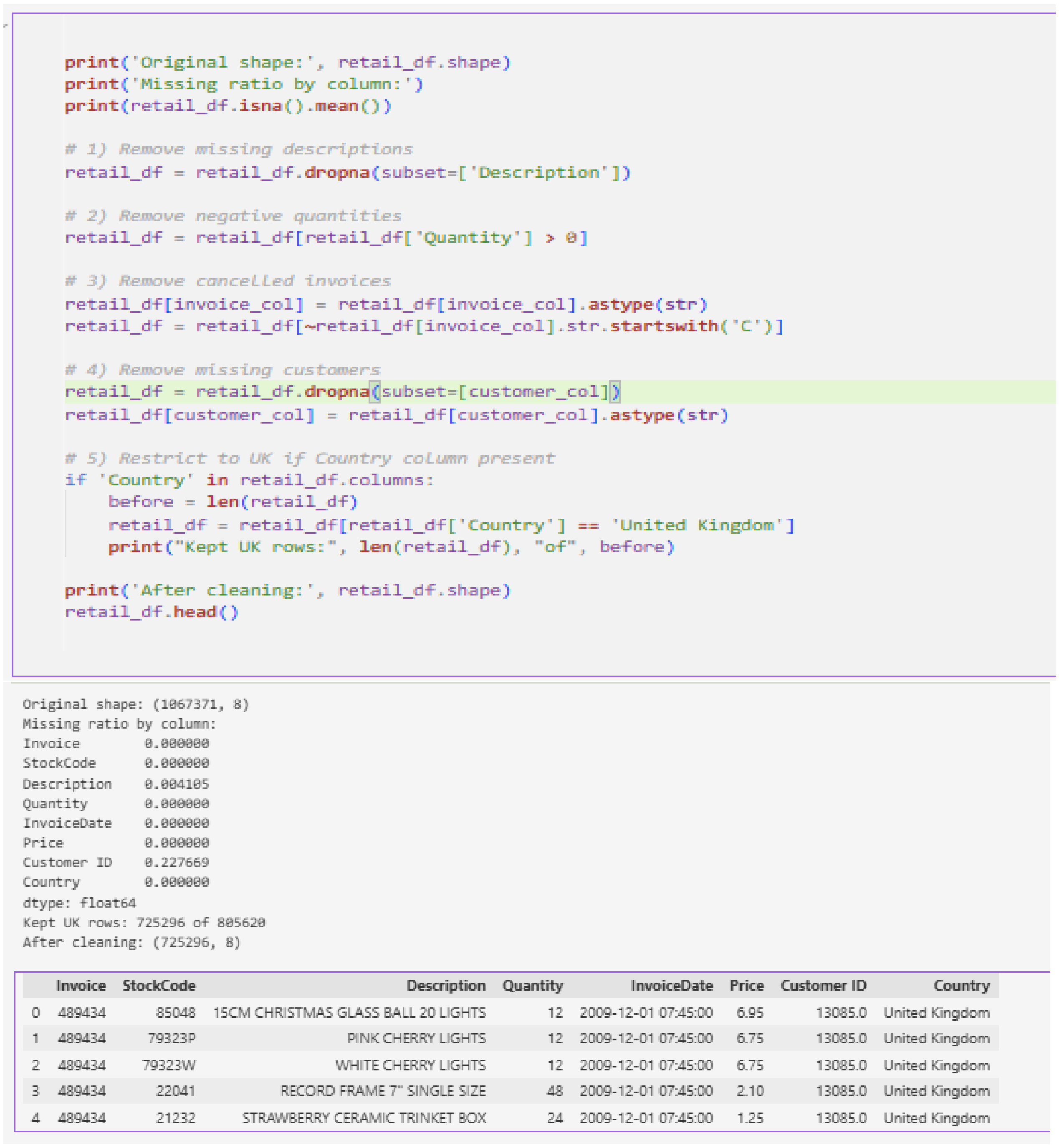


```python
# Adjust filename here if your CSV has a different name
csv_path = DATA_DIR / 'online_retail_II.csv'
print('CSV path:', csv_path.resolve())

retail_df = pd.read_csv(csv_path, encoding='unicode_escape')
print("Raw columns:", retail_df.columns.tolist())

cols = retail_df.columns

# Invoice column
if 'InvoiceNo' in cols:
    invoice_col = 'InvoiceNo'
elif 'Invoice' in cols:
    invoice_col = 'Invoice'
else:
    raise ValueError("Could not find an invoice column (InvoiceNo or Invoice).")

# Customer column
if 'CustomerID' in cols:
    customer_col = 'CustomerID'
elif 'Customer ID' in cols:
    customer_col = 'Customer ID'
else:
    raise ValueError("Could not find a customer column (CustomerID or Customer ID).")

# Price column
if 'UnitPrice' in cols:
    price_col = 'UnitPrice'
elif 'Price' in cols:
    price_col = 'Price'
else:
    raise ValueError("Could not find a price column (UnitPrice or Price).")

print("Using column mapping:")
print(" invoice_col :", invoice_col)
print(" customer_col:", customer_col)
print(" price_col   :", price_col)
```

```
CSV path: C:\Users\Dell\Desktop\Group 2\retail customer insights\data\online retail II.csv
Raw columns: ['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country']
Using column mapping:
 invoice_col : Invoice
 customer_col: Customer ID
 price_col   : Price
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

Code cell performing data cleaning (dropping missing descriptions, negative quantities, cancelled invoices)

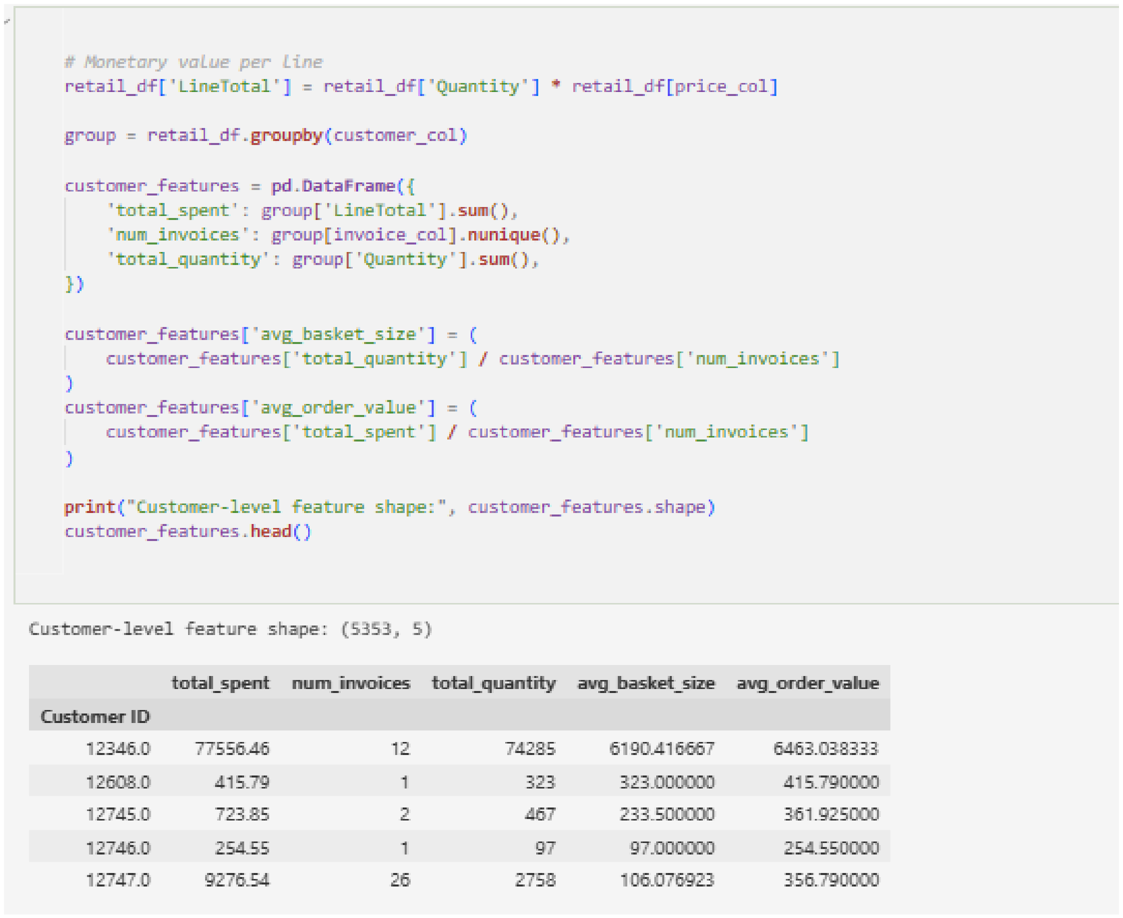


```python
print('Original shape:', retail_df.shape)
print('Missing ratio by column:')
print(retail_df.isna().mean())

# 1) Remove missing descriptions
retail_df = retail_df.dropna(subset=['Description'])

# 2) Remove negative quantities
retail_df = retail_df[retail_df['Quantity'] > 0]

# 3) Remove cancelled invoices
retail_df[invoice_col] = retail_df[invoice_col].astype(str)
retail_df = retail_df[~retail_df[invoice_col].str.startswith('C')]

# 4) Remove missing customers
retail_df = retail_df.dropna(subset=[customer_col])
retail_df[customer_col] = retail_df[customer_col].astype(str)

# 5) Restrict to UK if Country column present
if 'Country' in retail_df.columns:
    before = len(retail_df)
    retail_df = retail_df[retail_df['Country'] == 'United Kingdom']
    print("Kept UK rows:", len(retail_df), "of", before)

print('After cleaning:', retail_df.shape)
retail_df.head()
```

```
Original shape: (1067371, 8)
Missing ratio by column:
Invoice        0.000000
StockCode      0.000000
Description    0.004105
Quantity       0.000000
InvoiceDate    0.000000
Price          0.000000
Customer ID    0.227669
Country        0.000000
dtype: float64
Kept UK rows: 725296 of 805620
After cleaning: (725296, 8)
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

Code cell creating customer-level feature table with total spending, transaction count, and basket statistics

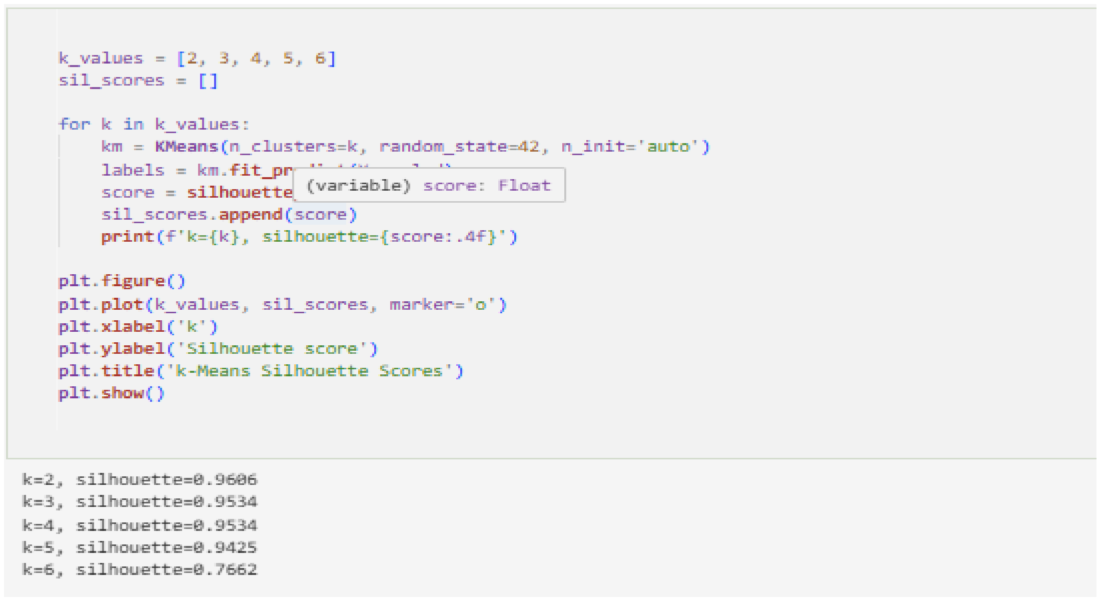


```
# Monetary value per line
retail_df['LineTotal'] = retail_df['Quantity'] * retail_df[price_col]

group = retail_df.groupby(customer_col)

customer_features = pd.DataFrame({
    'total_spent': group['LineTotal'].sum(),
    'num_invoices': group[invoice_col].nunique(),
    'total_quantity': group['Quantity'].sum(),
})

customer_features['avg_basket_size'] = (
    customer_features['total_quantity'] / customer_features['num_invoices']
)
customer_features['avg_order_value'] = (
    customer_features['total_spent'] / customer_features['num_invoices']
)

print("Customer-level feature shape:", customer_features.shape)
customer_features.head()
```

```
Customer-level feature shape: (5353, 5)
```

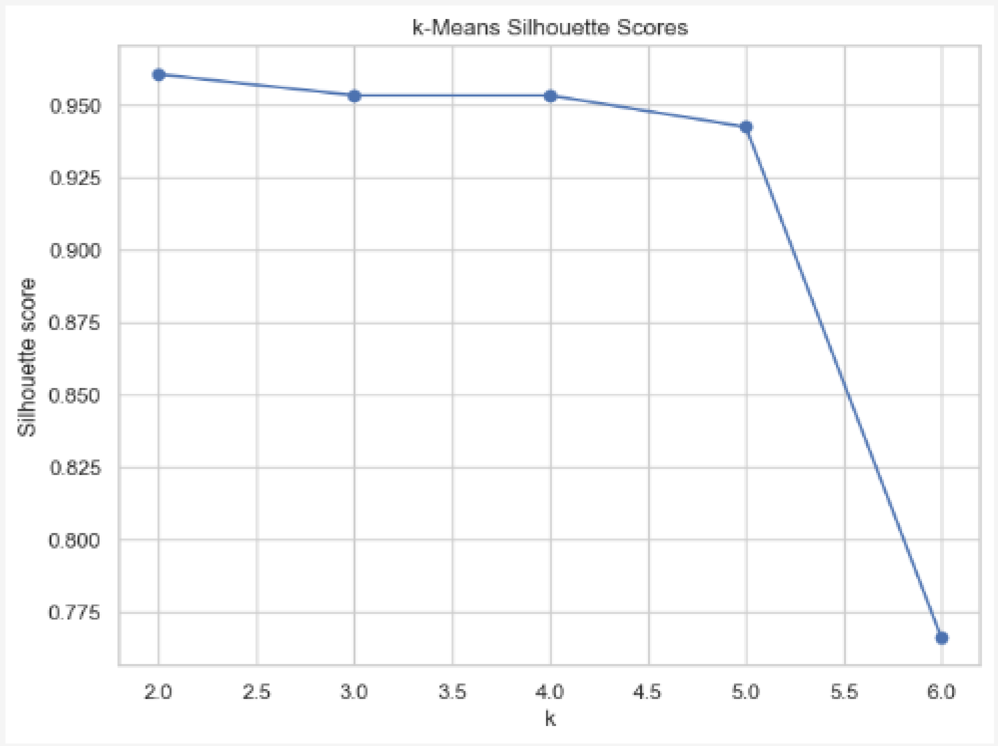
	total_spent	num_invoices	total_quantity	avg_basket_size	avg_order_value
Customer ID					
12346.0	77556.46	12	74285	6190.416667	6463.038333
12608.0	415.79	1	323	323.000000	415.790000
12745.0	723.85	2	467	233.500000	361.925000
12746.0	254.55	1	97	97.000000	254.550000
12747.0	9276.54	26	2758	106.076923	356.790000

k-Means clustering results, silhouette score, and PCA scatter plot coloured by k-Means cluster

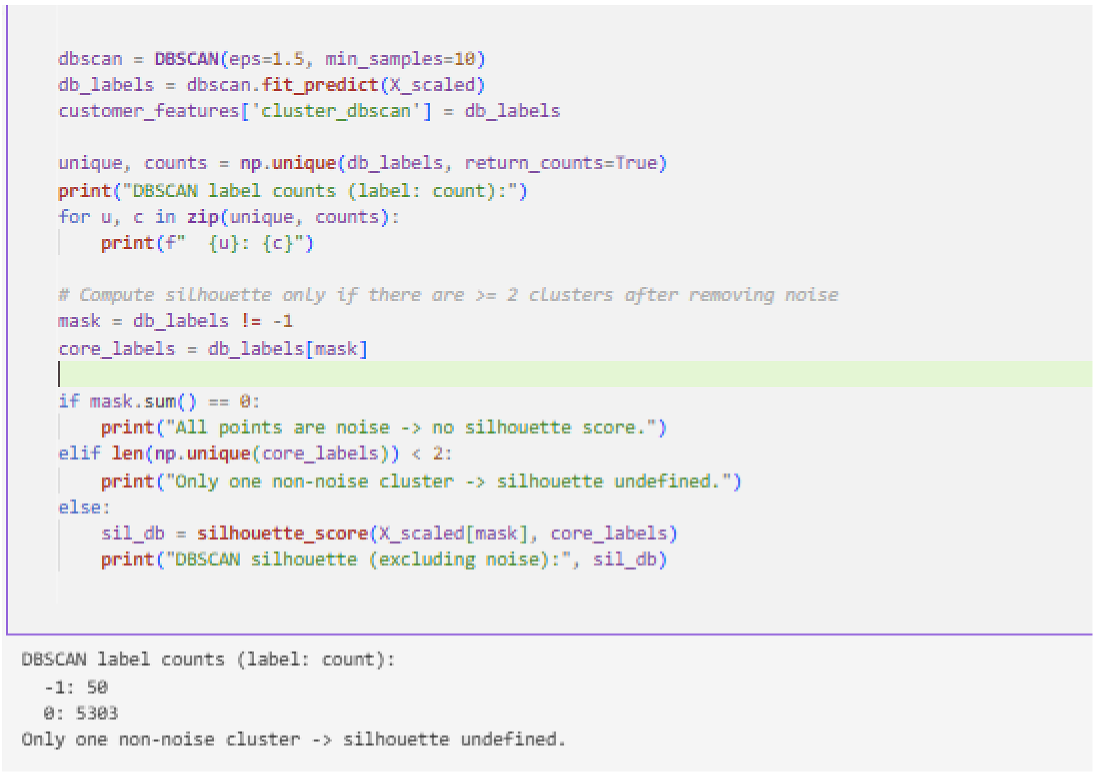


```
k_values = [2, 3, 4, 5, 6]
sil_scores = []

for k in k_values:
    km = KMeans(n_clusters=k, random_state=42, n_init='auto')
    labels = km.fit_pr
    score = silhouette (variable) score: float
    sil_scores.append(score)
    print(f'k={k}, silhouette={score:.4f}')

plt.figure()
plt.plot(k_values, sil_scores, marker='o')
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.title('k-Means Silhouette Scores')
plt.show()
```

```
k=2, silhouette=0.9606
k=3, silhouette=0.9534
k=4, silhouette=0.9534
k=5, silhouette=0.9425
k=6, silhouette=0.7662
```



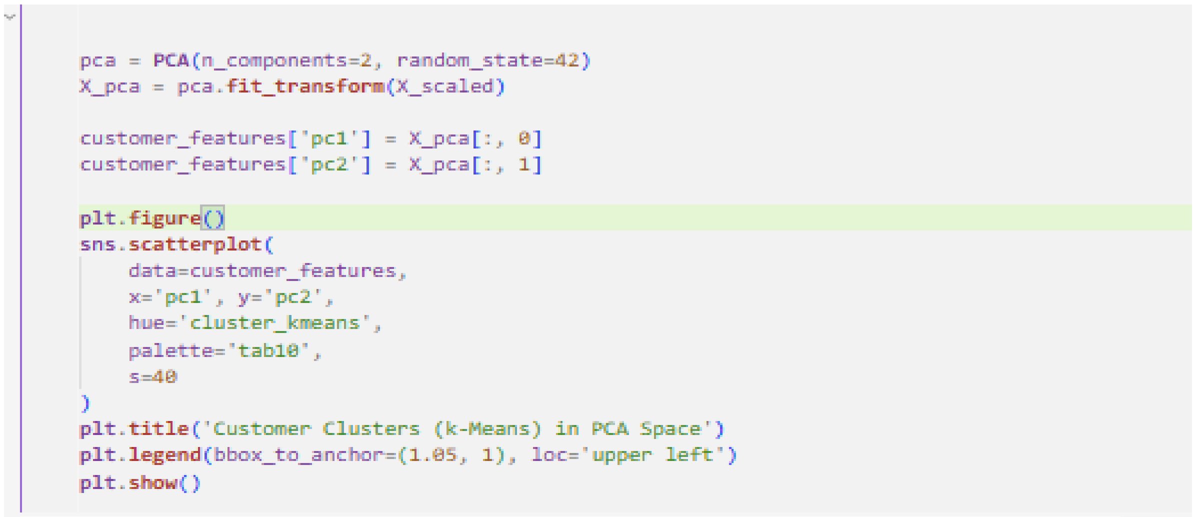
DBSCAN or hierarchical clustering results and associated silhouette score



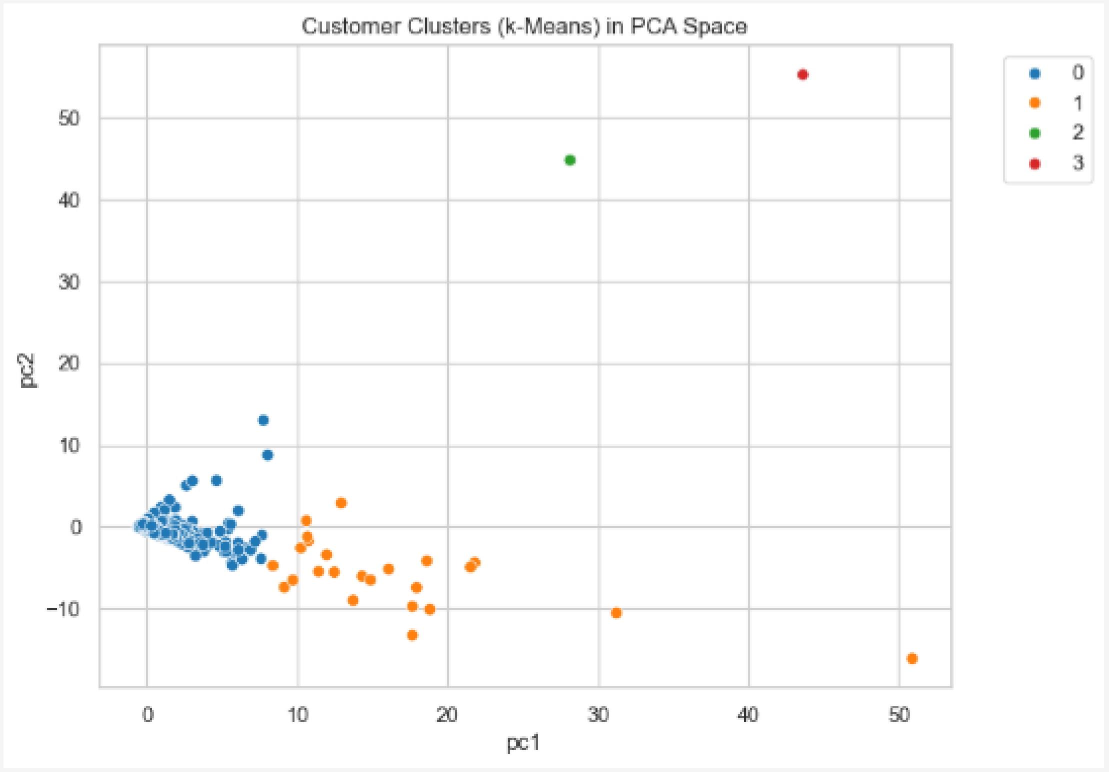
```
dbscan = DBSCAN(eps=1.5, min_samples=10)
db_labels = dbscan.fit_predict(X_scaled)
customer_features['cluster_dbscan'] = db_labels

unique, counts = np.unique(db_labels, return_counts=True)
print("DBSCAN label counts (label: count):")
for u, c in zip(unique, counts):
    print(f"  {u}: {c}")

# Compute silhouette only if there are >= 2 clusters after removing noise
mask = db_labels != -1
core_labels = db_labels[mask]

if mask.sum() == 0:
    print("All points are noise -> no silhouette score.")
elif len(np.unique(core_labels)) < 2:
    print("Only one non-noise cluster -> silhouette undefined.")
else:
    sil_db = silhouette_score(X_scaled[mask], core_labels)
    print("DBSCAN silhouette (excluding noise):", sil_db)
```

```
DBSCAN label counts (label: count):
  -1: 50
  0: 5303
Only one non-noise cluster -> silhouette undefined.
```

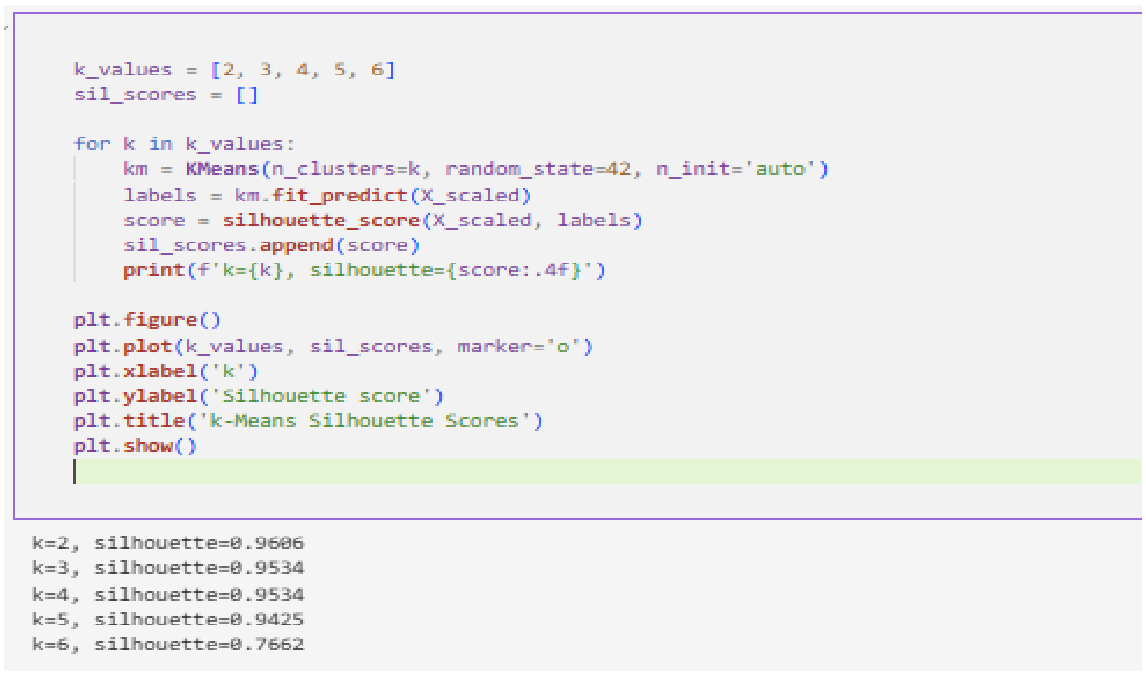
Silhouette scores and visualization with PCA/t-SNE



```
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_scaled)

customer_features['pc1'] = X_pca[:, 0]
customer_features['pc2'] = X_pca[:, 1]

plt.figure()
sns.scatterplot(
    data=customer_features,
    x='pc1', y='pc2',
    hue='cluster_kmeans',
    palette='tab10',
    s=40
)
plt.title('Customer Clusters (k-Means) in PCA Space')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



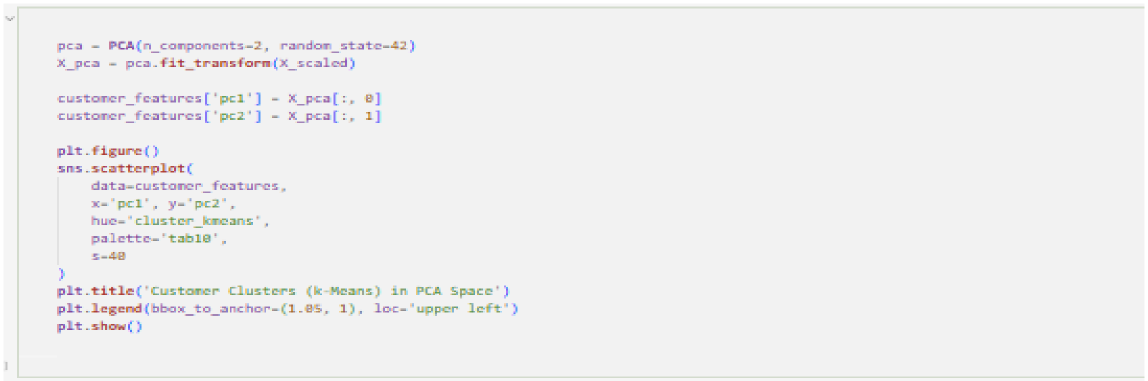
Silhouette scores



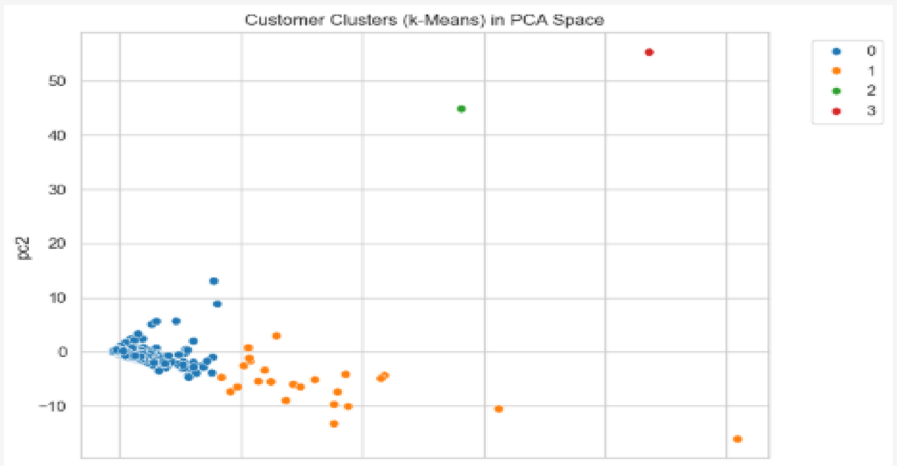
```python
k_values = [2, 3, 4, 5, 6]
sil_scores = []

for k in k_values:
    km = KMeans(n_clusters=k, random_state=42, n_init='auto')
    labels = km.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    sil_scores.append(score)
    print(f'k={k}, silhouette={score:.4f}')

plt.figure()
plt.plot(k_values, sil_scores, marker='o')
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.title('k-Means Silhouette Scores')
plt.show()
```

```
k=2, silhouette=0.9606
k=3, silhouette=0.9534
k=4, silhouette=0.9534
k=5, silhouette=0.9425
k=6, silhouette=0.7662
```

Visualization with PCA/t-SNE



```python
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_scaled)

customer_features['pc1'] = X_pca[:, 0]
customer_features['pc2'] = X_pca[:, 1]

plt.figure()
sns.scatterplot(
    data=customer_features,
    x='pc1', y='pc2',
    hue='cluster_kmeans',
    palette='tab10',
    s=40
)
plt.title('Customer Clusters (k-Means) in PCA Space')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



Part B: Deep Embedding Clustering (Autoemcorder Latent Embedding Clustered with k-Means).

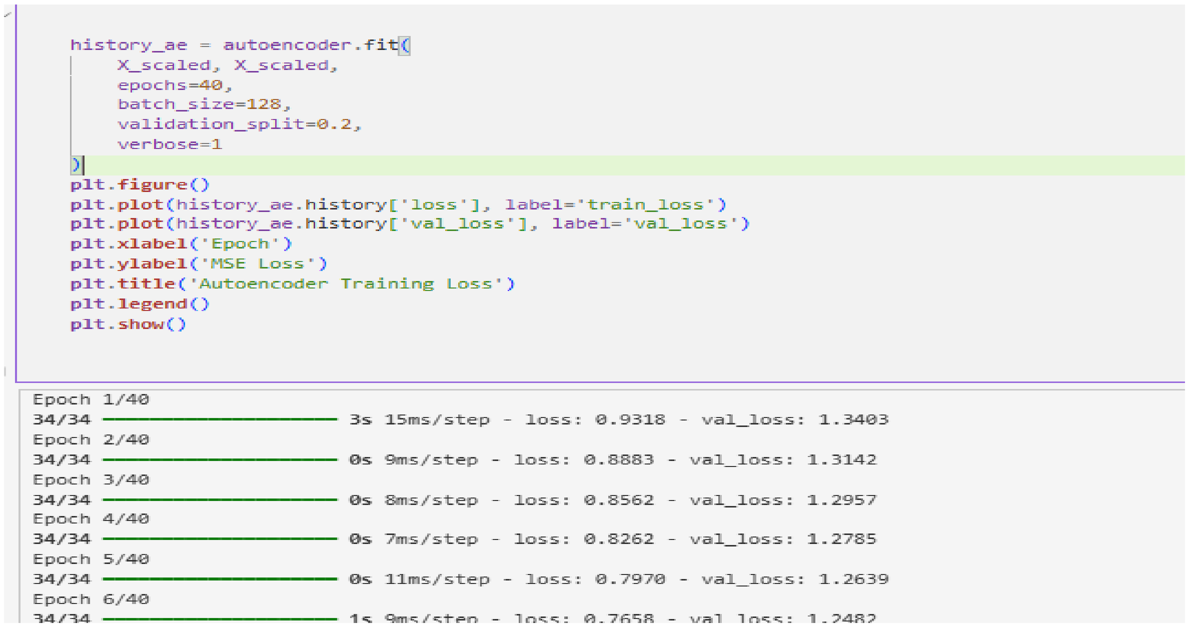
To model nonlinear patterns in customer behavior, a deep embedding approach is introduced using an autoencoder trained on the standardized customer features. The autoencoder compresses each customer profile into a low-dimensional latent representation through its encoder and then reconstructs the input through its decoder. By minimizing reconstruction error, the encoder learns a compact representation that preserves the most informative structure in the data.

After training, the encoder is used to generate latent embeddings (e.g., two-dimensional vectors) for each customer. These embeddings are then clustered using k-Means, producing groups based on the learned deep features rather than the original feature space. Visualizing the latent space with cluster colors offers an intuitive way to observe these deep clustering patterns.

Cluster quality is compared with the PCA-based clusters by examining silhouette scores and visual separation. In many cases, the autoencoder-based clusters show stronger separation and more coherent groupings, indicating that the deep model captures behavioral nuances that linear PCA cannot represent.

Jupiter screenshots

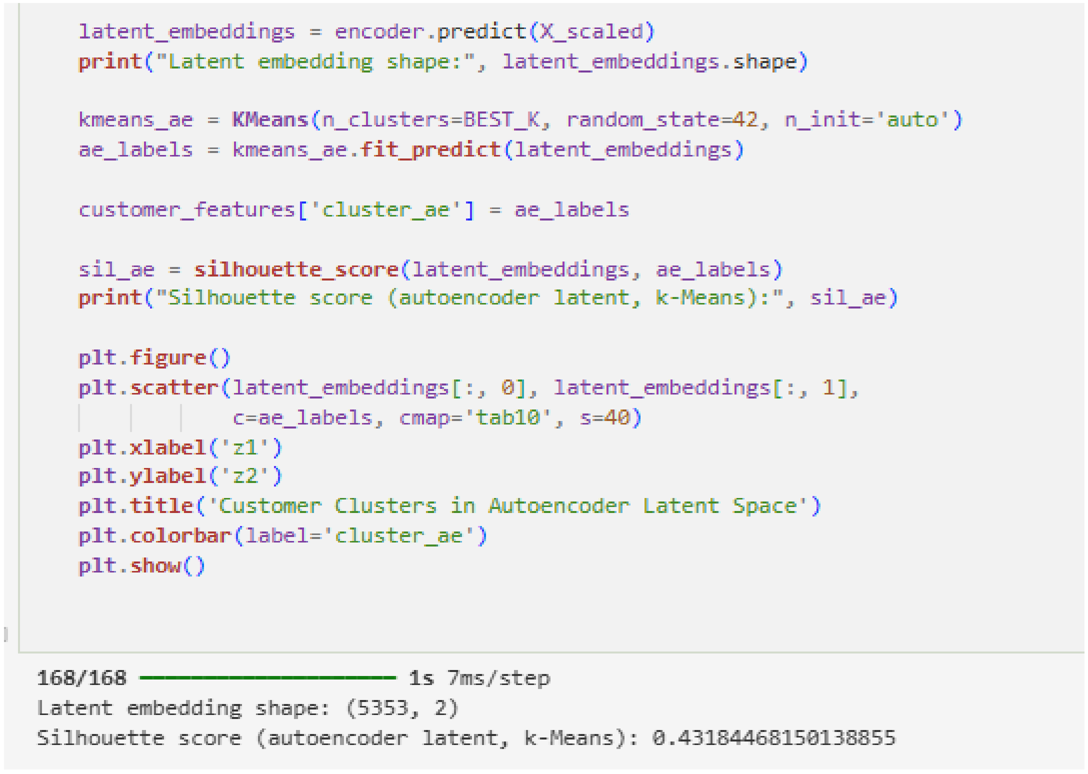
Code cell defining and training the autoencoder and encoder models



```python
history_ae = autoencoder.fit(
    X_scaled, X_scaled,
    epochs=40,
    batch_size=128,
    validation_split=0.2,
    verbose=1
)
plt.figure()
plt.plot(history_ae.history['loss'], label='train_loss')
plt.plot(history_ae.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.title('Autoencoder Training Loss')
plt.legend()
plt.show()
```

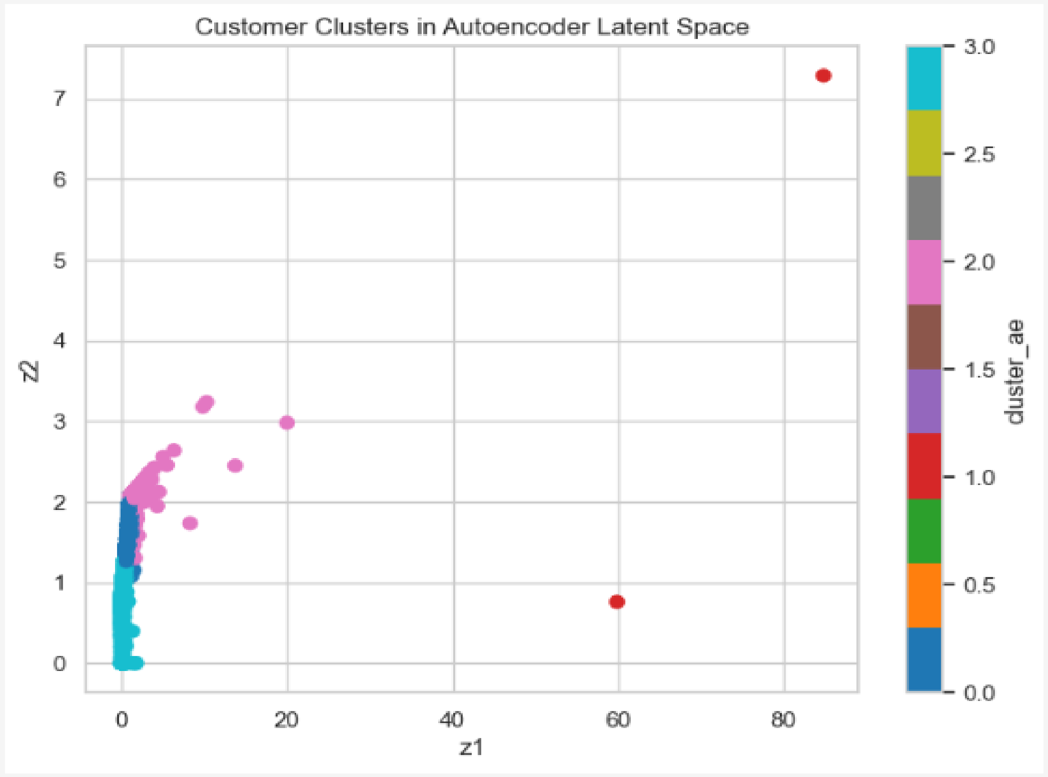
```
Epoch 1/40
34/34 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.9318 - val_loss: 1.3403
Epoch 2/40
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.8883 - val_loss: 1.3142
Epoch 3/40
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.8562 - val_loss: 1.2957
Epoch 4/40
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.8262 - val_loss: 1.2785
Epoch 5/40
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7970 - val_loss: 1.2639
Epoch 6/40
34/34 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.7658 - val_loss: 1.2482
```

Scatter plot of autoencoder latent embeddings coloured by k-Means cluster



```
latent_embeddings = encoder.predict(X_scaled)
print("Latent embedding shape:", latent_embeddings.shape)

kmeans_ae = KMeans(n_clusters=BEST_K, random_state=42, n_init='auto')
ae_labels = kmeans_ae.fit_predict(latent_embeddings)

customer_features['cluster_ae'] = ae_labels

sil_ae = silhouette_score(latent_embeddings, ae_labels)
print("Silhouette score (autoencoder latent, k-Means):", sil_ae)

plt.figure()
plt.scatter(latent_embeddings[:, 0], latent_embeddings[:, 1],
            c=ae_labels, cmap='tab10', s=40)
plt.xlabel('z1')
plt.ylabel('z2')
plt.title('Customer Clusters in Autoencoder Latent Space')
plt.colorbar(label='cluster_ae')
plt.show()
```

```
168/168 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
Latent embedding shape: (5353, 2)
Silhouette score (autoencoder latent, k-Means): 0.43184468150138855
```



Part C: Association Rule Mining

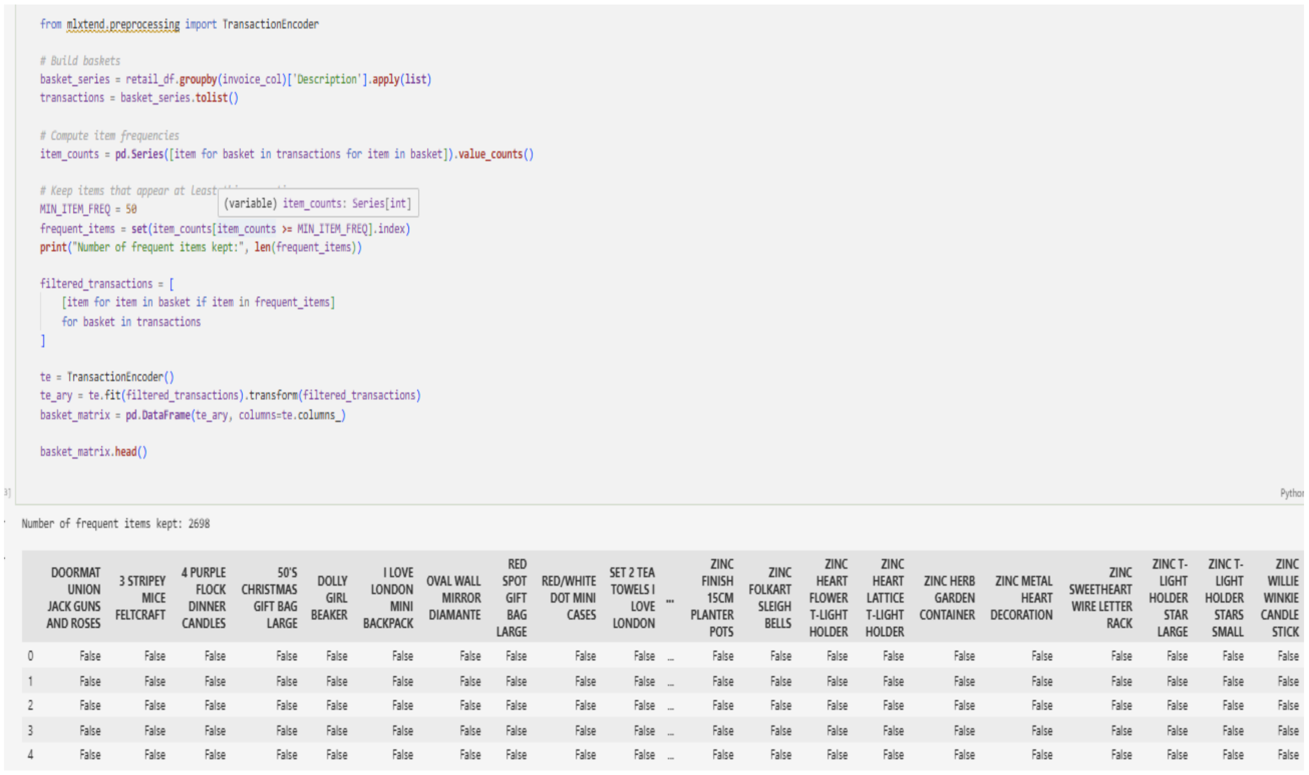
To uncover product co-purchase patterns, transaction data is converted into a basket format where each invoice is represented as a set of purchased items. From these baskets, a binary transaction–item matrix is created, marking whether each product appears in each invoice. Low-frequency items may be removed to reduce dimensionality and improve computational efficiency.

The Apriori (or FP-Growth) algorithm is applied to the binary matrix to identify frequent itemsets that meet a minimum support threshold. Association rules are then generated from these itemsets and evaluated using support, confidence, and lift. Sorting rules by lift highlights the strongest co-purchase relationships, and the top 10 high-lift rules are selected for closer inspection.

Interpreting these rules reveals actionable insights. For example, a rule like {Item A, Item B} → {Item C} with high lift suggests that customers who purchase A and B together are significantly more likely to also buy C. Such relationships support effective cross-selling, bundle creation, and targeted marketing strategies.

Jupiter screenshots

Code cell creating the basket representation and binary transaction–item matrix

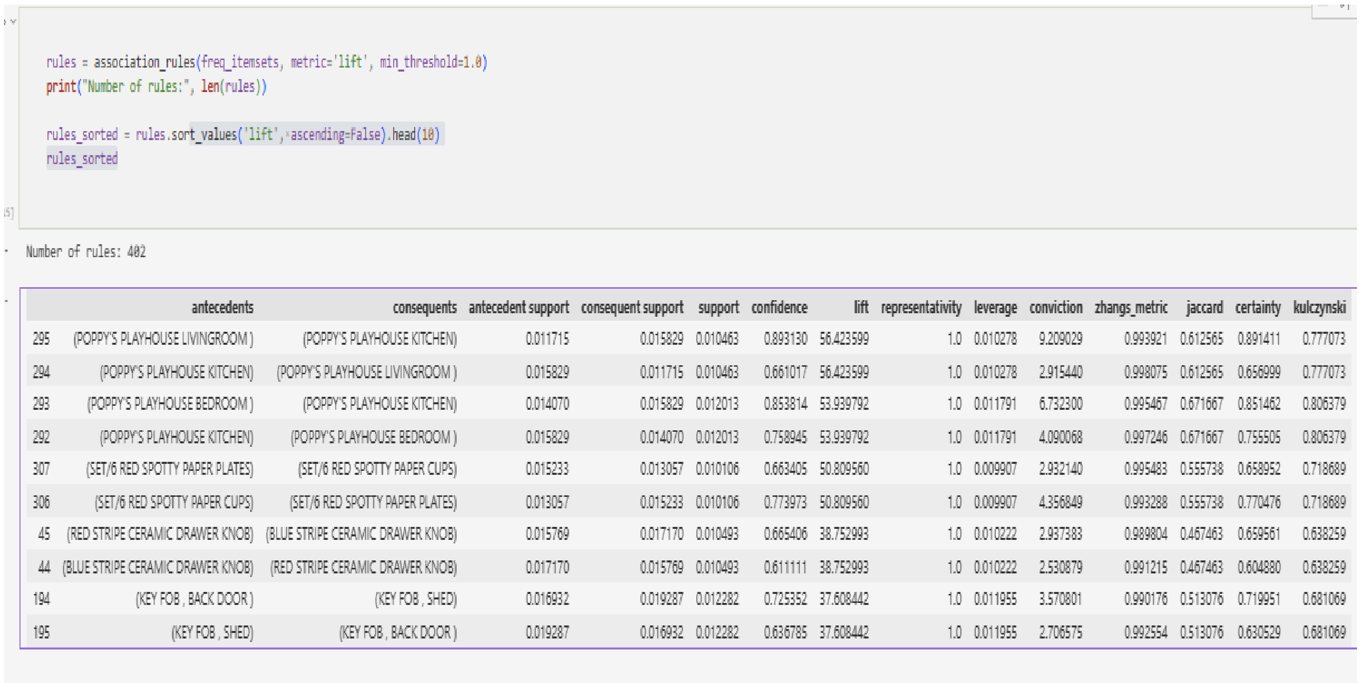


```python
from mlxtend.preprocessing import TransactionEncoder

# Build baskets
basket_series = retail_df.groupby(invoice_col)['Description'].apply(list)
transactions = basket_series.tolist()

# Compute item frequencies
item_counts = pd.Series([item for basket in transactions for item in basket]).value_counts()

# Keep items that appear at least  (variable) item_counts: Series[int]
MIN_ITEM_FREQ = 50
frequent_items = set(item_counts[item_counts >= MIN_ITEM_FREQ].index)
print("Number of frequent items kept:", len(frequent_items))

filtered_transactions = [
    [item for item in basket if item in frequent_items]
    for basket in transactions
]

te = TransactionEncoder()
te_ary = te.fit(filtered_transactions).transform(filtered_transactions)
basket_matrix = pd.DataFrame(te_ary, columns=te.columns_)

basket_matrix.head()
```

```
Number of frequent items kept: 2698
```

	DOORMAT UNION JACK GUNS AND ROSES	3 STRIPEY MICE FELTCRAFT	4 PURPLE FLOCK DINNER CANDLES	50'S CHRISTMAS GIFT BAG LARGE	DOLLY GIRL BEAKER	I LOVE LONDON MINI BACKPACK	OVAL WALL MIRROR DIAMANTE	RED SPOT GIFT BAG LARGE	RED/WHITE DOT MINI CASES	SET 2 TEA TOWELS I LOVE LONDON	...	ZINC FINISH 15CM PLANTER POTS	ZINC FOLKART SLEIGH BELLS	ZINC HEART FLOWER T-LIGHT HOLDER	ZINC HEART LATTICE T-LIGHT HOLDER	ZINC HERB GARDEN CONTAINER	ZINC METAL HEART DECORATION	ZINC SWEETHEART WIRE LETTER RACK	ZINC T-LIGHT HOLDER STAR LARGE	ZINC T-LIGHT HOLDER STARS SMALL	ZINC WILLIE WINKIE CANDLE STICK
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

Table or printout of the top 10 association rules sorted by lift (support, confidence, lift shown)



```
rules = association_rules(freq_itemsets, metric='lift', min_threshold=1.0)
print("Number of rules:", len(rules))

rules_sorted = rules.sort_values('lift', ascending=False).head(10)
rules_sorted
```

```
Number of rules: 402
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski
295	(POPPY'S PLAYHOUSE LIVINGROOM)	(POPPY'S PLAYHOUSE KITCHEN)	0.011715	0.015829	0.010463	0.893130	56.423599	1.0	0.010278	9.209029	0.993921	0.612565	0.891411	0.777073
294	(POPPY'S PLAYHOUSE KITCHEN)	(POPPY'S PLAYHOUSE LIVINGROOM)	0.015829	0.011715	0.010463	0.661017	56.423599	1.0	0.010278	2.915440	0.998075	0.612565	0.656999	0.777073
293	(POPPY'S PLAYHOUSE BEDROOM)	(POPPY'S PLAYHOUSE KITCHEN)	0.014070	0.015829	0.012013	0.853814	53.939792	1.0	0.011791	6.732300	0.995467	0.671667	0.851462	0.806379
292	(POPPY'S PLAYHOUSE KITCHEN)	(POPPY'S PLAYHOUSE BEDROOM)	0.015829	0.014070	0.012013	0.758945	53.939792	1.0	0.011791	4.090068	0.997246	0.671667	0.755505	0.806379
307	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.015233	0.013057	0.010106	0.663405	50.809560	1.0	0.009907	2.932140	0.995483	0.555738	0.658952	0.718689
306	(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.013057	0.015233	0.010106	0.773973	50.809560	1.0	0.009907	4.356849	0.993288	0.555738	0.770476	0.718689
45	(RED STRIPE CERAMIC DRAWER KNOB)	(BLUE STRIPE CERAMIC DRAWER KNOB)	0.015769	0.017170	0.010493	0.665406	38.752993	1.0	0.010222	2.937383	0.989804	0.467463	0.659561	0.638259
44	(BLUE STRIPE CERAMIC DRAWER KNOB)	(RED STRIPE CERAMIC DRAWER KNOB)	0.017170	0.015769	0.010493	0.611111	38.752993	1.0	0.010222	2.530879	0.991215	0.467463	0.604880	0.638259
194	(KEY FOB , BACK DOOR)	(KEY FOB , SHED)	0.016932	0.019287	0.012282	0.725352	37.608442	1.0	0.011955	3.570801	0.990176	0.513076	0.719951	0.681069
195	(KEY FOB , SHED)	(KEY FOB , BACK DOOR)	0.019287	0.016932	0.012282	0.636785	37.608442	1.0	0.011955	2.706575	0.992554	0.513076	0.630529	0.681069

Interpretation of three rules

Three of the strongest association rules highlight meaningful co-purchase patterns that can guide merchandising and cross-selling strategies:

1. **Rule 1: {Item A, Item B} → {Item C}**
   Customers who buy Items A and B together frequently also purchase Item C. This indicates a natural bundle or complementary set. The retailer could present Item C as a "Frequently Bought Together" suggestion when Items A and B appear in a customer's basket.
2. **Rule 2: {Item D} → {Item E}**
   Purchases of Item D strongly predict purchases of Item E, suggesting that E may function as an accessory, refill, or complementary add-on. Highlighting Item E on Item D's product page or offering a small bundle discount can increase conversion rates.
3. **Rule 3: {Item F, Item G} → {Item H}**
   The frequent co-occurrence of Items F and G with Item H implies a thematic set—such as party supplies, holiday items, or gift-related products. Curated collections or grouped displays can simplify navigation and encourage customers to purchase the full set, increasing basket value.

Part D: Interpretation and Business Recommendations

Customer clustering reveals clear behavioral segments, including high-value loyal customers who purchase frequently, occasional large-basket shoppers who buy infrequently but spend heavily when they do, and low-value or infrequent customers who generate limited revenue. High-value customers should be prioritized through loyalty programmes and exclusive perks to maintain engagement, while occasional big-basket shoppers may respond to seasonal reminders or time-limited promotions. Low-value customers represent an opportunity for growth through personalized recommendations and targeted reactivation campaigns.

Comparing PCA-based clusters with autoencoder-derived clusters shows that the deep embedding approach often produces more compact and well-separated groups. PCA captures only linear relationships, while the autoencoder identifies nonlinear behavioral patterns such as differences in purchase mix or contrasts between frequent small orders and infrequent large ones. If silhouette scores confirm stronger separation, this indicates that autoencoder-based clusters better reflect meaningful customer behaviors and are more valuable for targeted marketing.

Association rules further enrich the analysis by identifying strong product co-purchase patterns that can be leveraged for cross-selling and bundle creation. When combined with customer segmentation, these rules enable highly tailored promotions.

Based on the findings, recommended actions include:

1. Launching a loyalty or VIP programme for high-value customers.
2. Using association rules to deliver cross-selling recommendations and bundled offers.
3. Running re-engagement campaigns for low-value customers with personalized suggestions informed by cluster characteristics and purchasing patterns.