MACHINE LEARNING AND DATA MINING ASSESSMENT REPORT

EBAINJUIAYUK BENJAMIN AKOMPAB

MSc. Artificial Intelligence | December 2025

@00847647

Assessment report submitted in part fulfilment of the module

Machine Learning and Data Mining

In December 2025

TABLE OF CONTENTS

TASK 1- PREDICTING CREDIT CARD DEFAULT USING MACHINE LEARNING CLASSIFICATION MODELS: A COMPARATIVE ANALYSIS OF DECISION TREE AND K-NEAREST NEIGHBOURS

TASK 2- CUSTOMER SKU SEGMENTATION USING K-MEANS AND HIERARCHICAL CLUSTERING

Task 3- Evaluating Customer Sentiment Through Predictive Modelling to Support Operational and Strategic Business Decisions

Task 1- Predicting Credit Card Default Using Machine Learning Classification Models: A Comparative Analysis of Decision Tree and K-Nearest Neighbours

1.0 INTRODUCTION

1.1 Background

Credit risk remains a challenge for banks, in credit card lending, as defaults weaken stability. Increasing card usage makes behaviour more complex, reducing the effectiveness of traditional credit scoring models and leading to inaccurate predictions (Arora, 2022; Yin, 2023). Modern data analytics and machine learning analyse customer history and behavioural data to detect hidden patterns, improving default prediction and supporting risk management, profitability, and overall stability (Han, 2024; Xu, 2024).

1.2 Problem Statement

Credit card default remains a major global issue for financial institutions, causing losses and uncertainty in credit portfolios. Traditional rule-based scoring systems struggle to capture the complex and changing factors that affect repayment behaviour. This study analyses a dataset of credit card clients to develop predictive models that identify customers at higher risk of default. Improved prediction supports better decisions in credit approval, limit allocation, and debt recovery, helping lenders manage risk more effectively.

1.3 Aim and Objectives

This study aims to build and test classification models to predict credit card default and find best for identifying risky customers.

Objectives:

- Explore and prepare the credit card default dataset using EDA, feature scaling, and class imbalance handling for machine learning.
- Develop and optimise Decision Tree and K-Nearest Neighbours (KNN) models to predict credit card default.
- Evaluate and compare both models using performance metrics.

1.4 Related Research

Recent studies show machine learning is widely used for credit default prediction. Advanced models such as random forests and neural networks often outperform simpler methods like logistic regression (Bhandary and Ghosh, 2025). Class imbalance can reduce the reliability of explanation tools like SHAP and LIME, requiring combined sampling techniques (Chen et al., 2024; Alamri and Ykhlef, 2024). Concerns about fairness and transparency remain important in automated credit decisions (Mariscal et al., 2024; FinRegLab, 2023). Hybrid models have also demonstrated strong predictive performance (Liu et al., 2022).

1.5 Research Gap

Although progress has been made, gaps still exist in credit default research. Many studies focus mainly on accuracy and ignore business-focused measures like precision and recall (Bhandary & Ghosh, 2025). Most research also looks at how resampling improves accuracy, but gives less attention to its impact on fairness and interpretability (Chen et al., 2024; Alamri & Ykhlef,

2024). Few studies provide a complete, reproducible framework covering data preparation, model tuning, interpretation and fairness assessment together (FinRegLab, 2023; Mariscal et al., 2024).

1.6 Research Question

Which model, Decision Tree or K-Nearest Neighbours (KNN), is more effective in predicting credit card default and identifying high-risk customers?

2.0 DATASET

The Default of Credit Card Clients dataset from the UCI Repository contains information on 30,000 customers from a Taiwanese bank collected between April and September 2005 (Yeh & Lien, 2009). It is widely used for default prediction as it includes demographic, financial and behavioural data (Bhandary & Ghosh, 2025; Chen et al., 2024). The dataset has 24 variables, including a default target, but is imbalanced with 22% defaults.

2.1- Ethical/Social and Legal Consideration

This dataset raises ethical concerns because it uses personal details like age, gender, education and marital status, which may lead to unfair judgement and discrimination. Such bias can affect access to credit and damage trust. Being labelled “high risk” may also harm a person’s reputation and confidence. To reduce these risks, data must be handled carefully with transparency, regular bias monitoring and strict compliance with laws such as GDPR and the FCRA to ensure fair and responsible use.

3.0 EDA AND DATA PREPROCESSING

3.1 Brief Description of dataset

The independent variables include client details such as credit limit, gender, education, marital status and age, along with repayment status over the past six months. The dataset also records monthly bill amounts and payment amounts, showing how customers use and repay credit. These features describe the client’s financial background and behaviour.

The dependent variable, Y, shows whether the client defaulted on their next payment and is the key outcome used for prediction.

3.2 Exploratory data analysis

Exploratory Data Analysis was used to understand factors linked to default by checking data structure, basic statistics, missing values and summary results. This helped reveal patterns, distributions and possible issues within the dataset variables

```
#Generate a summary of descriptive statistics for all columns in a DataFrame
dataset.describe(include='all')
```

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	300(
mean	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200	
std	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868	
min	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	
max	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	

8 rows × 24 columns

Figure 1 Summary Statistics

A bar chart was used to show default and non-default customers using the code below, making it easy to compare group sizes and spot class imbalance (figure 2). This was important as imbalance can impact how well the model performs.

```
# Plotting target distribution(Default vs Non Default)

plt.figure(figsize=(5,4))
target_counts.plot(kind='bar', color=['steelblue', 'orange'])
plt.title('Distribution of Default vs Non-default')
plt.xticks([0,1], ['Default (1)', 'Non-default (0)'], rotation=0)
plt.ylabel('Percentage (%)')
plt.show()
```

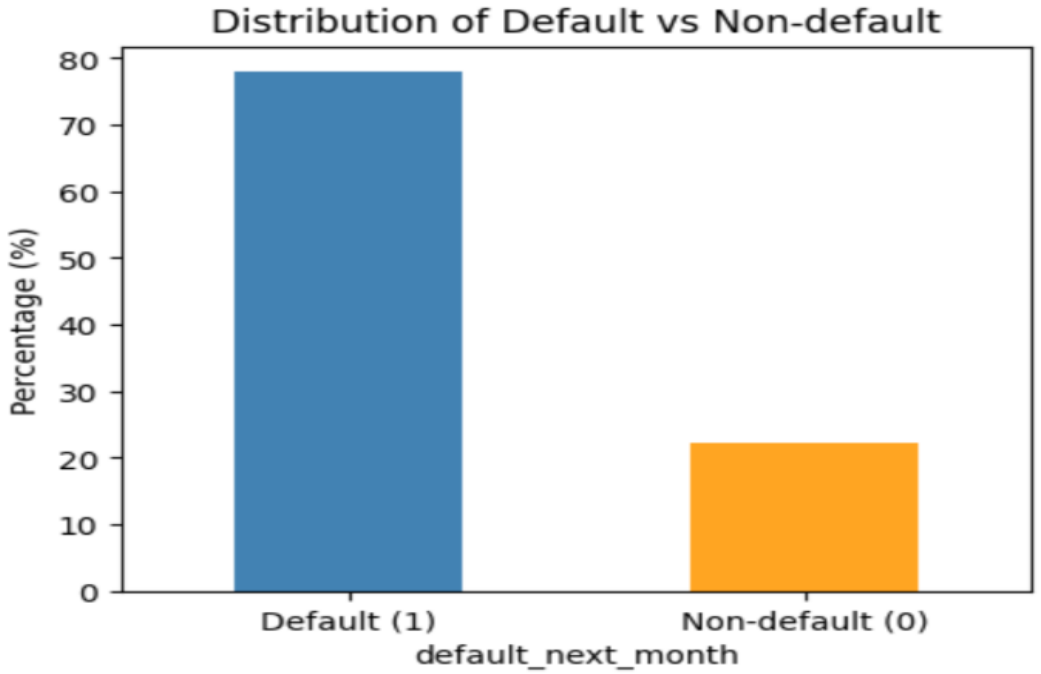

Figure 2. Distribution of target variable

.

A correlation heatmap was used to show how the first ten numerical variables relate to each other using the code below. It helped highlight strong links and possible multicollinearity, making the relationships easier to understand visually.

```
#Plotting a correlation heatmap showing how the first 10 numeric features in the dataset relate to one another.
plt.figure(figsize=(8, 5))
sns.heatmap(dataset.corr(numeric_only=True).iloc[:10, :10], annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap (First 10 Features)")
plt.tight_layout()
plt.show()
```

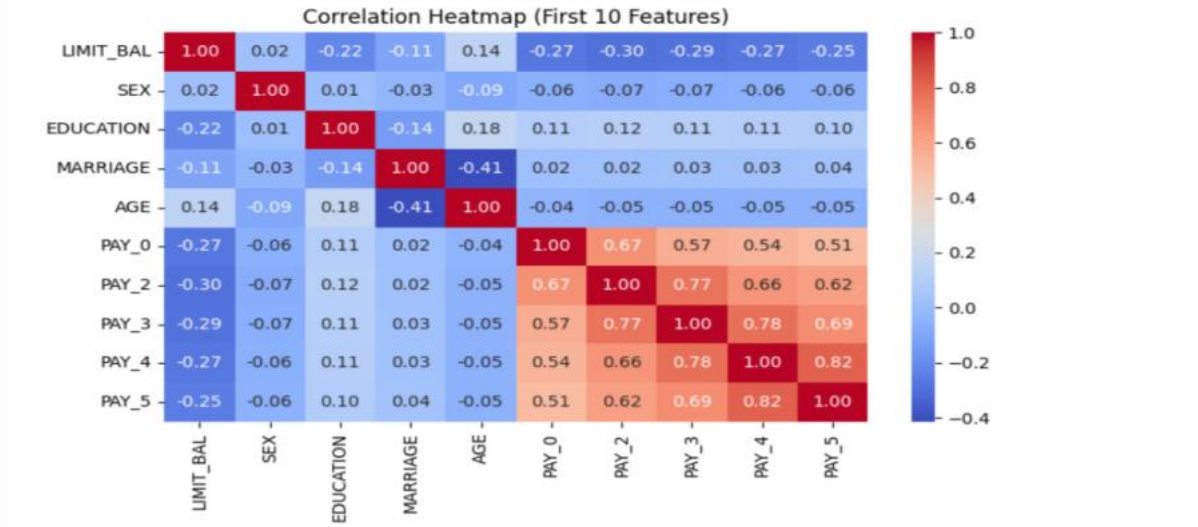

Figure 3. Correlation heatmap for the first 10 variables

The EDA showed that the dataset has more non-default cases than defaults, showing clear class imbalance. There were no missing values, but some numerical data was skewed. Categorical features like education and marital status contained invalid entries. These problems could reduce model accuracy if not corrected. A chart (figure 4) was used to clearly show the class distribution.

```
# Visualize class distribution
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
y_train.value_counts().plot(kind='bar', color=['lightblue', 'lightcoral'])
plt.title('Training Set Class Distribution')
plt.xlabel('Class (0=No Default, 1=Default)')
plt.ylabel('Count')
plt.xticks(rotation=0)

plt.subplot(1, 2, 2)
class_percentages = y_train.value_counts(normalize=True) * 100
class_percentages.plot(kind='bar', color=['lightblue', 'lightcoral'])
plt.title('Training Set Class Percentages')
plt.xlabel('Class (0=No Default, 1=Default)')
plt.ylabel('Percentage (%)')
plt.xticks(rotation=0)

plt.tight_layout()
plt.show()

print("Note: Using class_weight='balanced' in algorithms instead of SMOTE for better compatibility")
```

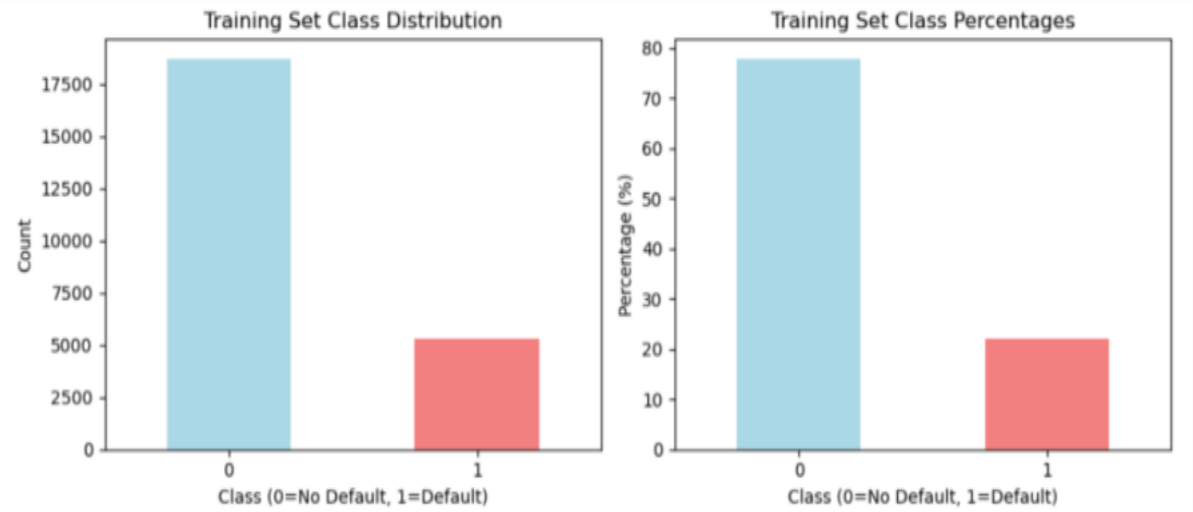

Figure 4.Class Distribution of target variable

3.3 Data Preprocessing

The dataset was cleaned by fixing and encoding incorrect values in categorical variables like education and marital status. It was then split into input and target variables. Stratification kept class balance in training and test sets, class weights handled imbalance, and StandardScaler was used to scale features for better model performance. Class distribution was checked before balancing and after applying class weight.

```
#For data processing we need to import train_test_split and StandardScaler.
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
#Standardize categorical data
dataset['EDUCATION'] = dataset['EDUCATION'].replace({0: 4, 5: 4, 6: 4})
dataset['MARRIAGE'] = dataset['MARRIAGE'].replace({0: 3})
```

```
#separate the dataset into features (X) and the target variable (y) (that we aere t
X = dataset.drop(columns=['default_next_month'])
y = dataset['default_next_month']
```

```
#split the dataset into training and testing sets and the apply StandardScaler to s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Feature scaling completed!")
print(f"Training set shape: {X_train_scaled.shape}")
print(f"Test set shape: {X_test_scaled.shape}")
```

The code below checked class distribution before balancing and after applying class weights for comparison purposes.

```
# Now apply class balancing - using class_weight instead of SMOTE due to compatibility issues
from sklearn.utils.class_weight import compute_class_weight

# Check class distribution before balancing
print("Original training set distribution:")
print(f"Class 0 (No default): {(y_train == 0).sum()}")
print(f"Class 1 (Default): {(y_train == 1).sum()}")
print(f"Class ratio: {(y_train == 0).sum() / (y_train == 1).sum():.2f}:1")
```

Checking class distribution before balancing.

```
# Compute class weights for balanced training, so the model does not ignore the minority class (defaults).)
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weight_dict = {0: class_weights[0], 1: class_weights[1]}

print(f"\nComputed class weights:")
print(f"Class 0 weight: {class_weight_dict[0]:.3f}")
print(f"Class 1 weight: {class_weight_dict[1]:.3f}")
```

Computing class weights for balanced training

	value
Class 0 weight	0.642
Class 1 weight	2.260

Table 1. Computed class weights

The class weights showed different importance for each group to handle imbalance. Non-default cases were more common, so they were given a lower weight of 0.642, while default cases received a higher weight of 2.260. This makes the model focus more on detecting defaults and reduces bias by penalising misclassification of the minority class.

4.0 CLASSIFICATION ALGORITHMS IMPLEMENTATION

4.1 Description of Algorithms

This task uses Decision Tree and KNN algorithms. Decision Tree is easy to understand, works with both numerical and categorical data, and does not rely on data distribution assumptions.

KNN is also simple and classifies points based on nearby data. Both were chosen because they are flexible and suitable for this prediction task.

4.2 Algorithm I (Decision Tree Classification)

The code below sets up a balanced decision tree, reduced overfitting, trained it on data, and prepared it for evaluation.

```
[ ]: # Algorithm 1: Decision Tree Classification with Class Balancing

print("=== DECISION TREE CLASSIFIER ===")
print("Algorithm Rationale:")
print("- Handles mixed data types (categorical + numerical)")
print("- Provides interpretable rules for business decisions")
print("- No assumptions about data distribution")
print("- Can capture non-linear relationships")

from sklearn.tree import DecisionTreeClassifier

# Train Decision Tree with balanced class weights
dt_classifier = DecisionTreeClassifier(
    criterion='entropy',
    random_state=42,
    class_weight='balanced',
    max_depth=10,
    min_samples_split=20,
    min_samples_leaf=10
)

print("Parameters chosen:")
print(f"- Criterion: entropy (information gain)")
print(f"- Class weight: balanced (handles imbalance)")
print(f"- Max depth: 10 (prevents overfitting)")
print(f"- Min samples split: 20")
print(f"- Min samples leaf: 10")

# Fitting Decision Tree Classification in the Training set
dt_classifier.fit(X_train, y_train)
print("\nDecision Tree training completed!")
```

4.3 Explanation of key parameters

Entropy was used for better splits and class weighting helped manage imbalance. A random state of 42 kept results consistent. Max depth was limited to 10, and minimum samples for splits and leaves were set to reduce overfitting and improve stability.

4.4 Decision Tree Prediction and Evaluation

Once the model is trained, the 'predict' function was used on the model to make predictions on the test data and the resulting performance metric displayed in table 2.

```python
# Decision Tree Predictions and Evaluation

# Make predictions
dt_pred = dt_classifier.predict(X_test)
dt_pred_proba = dt_classifier.predict_proba(X_test)[:, 1]

print("=== DECISION TREE RESULTS ===")
print(f"Predictions sample: {dt_pred[:10]}")
print(f"Actual sample: {y_test.iloc[:10].values}")

# Calculate basic metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

dt_accuracy = accuracy_score(y_test, dt_pred)
dt_precision = precision_score(y_test, dt_pred)
dt_recall = recall_score(y_test, dt_pred)
dt_f1 = f1_score(y_test, dt_pred)
dt_auc = roc_auc_score(y_test, dt_pred_proba)

print(f"\nDecision Tree Performance Metrics:")
print(f"Accuracy: {dt_accuracy:.4f}")
print(f"Precision: {dt_precision:.4f}")
print(f"Recall: {dt_recall:.4f}")
print(f"F1-Score: {dt_f1:.4f}")
print(f"ROC-AUC: {dt_auc:.4f}")
```

Metric	Performance
Accuracy	70.68%
Precision	39.50%
Recall	61.27%
F1-Score	48.04%
ROC-AUC	73.30%

Table 2. Decision Tree Performance Metrics. ROC-AUX value of above 0.7 suggests the model has a reasonable ability to distinguish between classes, even if the default threshold isn't optimal. The accuracy was 70.68%.

The model's predictions were used to evaluate the decision tree performance using the provided code below.

```
# Evaluate Decision Tree Performance (using dt_pred from earlier)
from sklearn import metrics

# Use Decision Tree predictions
print("=== DECISION TREE EVALUATION ===")

# Accuracy
acc = metrics.accuracy_score(y_test, dt_pred)
print('Accuracy: %.2f\n\n' % (acc))

# Confusion Matrix
cm = metrics.confusion_matrix(y_test, dt_pred)
print('Confusion Matrix:')
print(cm, '\n\n')

# Separator
print('-----------------------------------')

# Classification Report
result = metrics.classification_report(y_test, dt_pred, target_names=['No Default',
print('Classification Report:\n')
print(result)
```

	Precision	Recall	F1-Score
No Default	87%	73%	80%
Default	40%	61%	48%

Figure 5.Figure 5. Decision Tree Classification Report. Only 40% of predicted defaulters are truly defaulters (many false alarms). 61% of real defaulters are caught while 39% are missed. The model is over-predicting Default.

A heatmap was used to visualise the decision tree confusion matrix (figure 6) using the code below.

```
# Visualize Decision Tree confusion matrix using seaborn heatmap
dt_cm = metrics.confusion_matrix(y_test, dt_pred)

plt.figure(figsize=(8, 6))
ax = sns.heatmap(dt_cm, annot=True, fmt='d', cmap='Blues', vmin=0, vmax=dt_cm.max()
                 xticklabels=['No Default', 'Default'], yticklabels=['No Default',

plt.xlabel("Predicted Class", fontsize=12)
plt.ylabel("True Class", fontsize=12)
plt.title("Decision Tree Confusion Matrix", fontsize=12)

plt.tight_layout()
plt.show()

print(f"Confusion Matrix Analysis:")
print(f"True Negatives: {dt_cm[0,0]}")
print(f"False Positives: {dt_cm[0,1]}")
print(f"False Negatives: {dt_cm[1,0]}")
print(f"True Positives: {dt_cm[1,1]}")
```

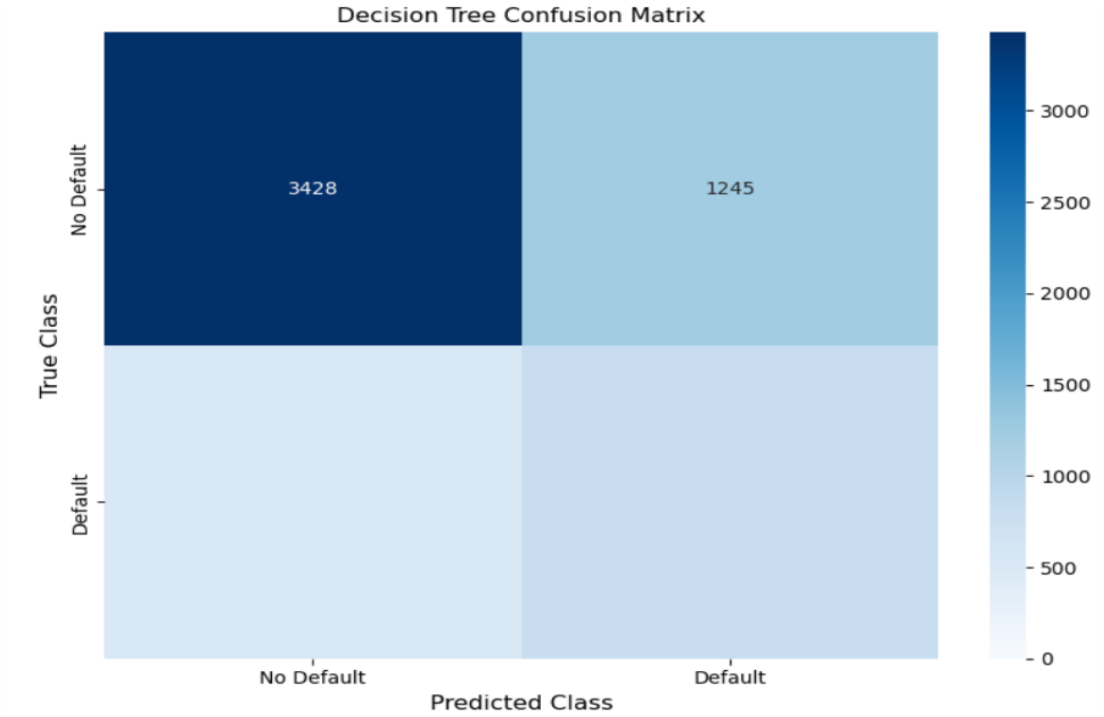

Figure 6.Decision tree confusion matrix. The model correctly identified 3,428 non-defaulters and 813 defaulters, but wrongly labelled 1,245 as defaults, showing a high false positive rate.

4.5 Algorithm II - (K-Nearest Neighbors (KNN) Classification)

The code below uses GridSearchCV to tune the KNN model by testing different neighbour values, weighting methods and distance measures. It applies 5-fold cross-validation and selects the best settings based on the F1-score. This helps improve performance and reduces the chance of overfitting.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

print("\n=== HYPERPARAMETER TUNING FOR KNN ===")
print("Using GridSearchCV to find optimal k value...")

param_grid = {
    'n_neighbors': range(3, 31, 2),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

knn_base = KNeighborsClassifier(n_jobs=-1)

grid_search = GridSearchCV(
    estimator=knn_base,
    param_grid=param_grid,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)
```

The GridSearchCV was run on scaled training data to find the best KNN settings. Using these results, the final optimised KNN model was trained and saved for evaluation.

```
# Fit grid search on scaled training data
grid_search.fit(X_train_scaled, y_train)

print(f"\nBest parameters found:")
print(f"- Number of neighbors (k): {grid_search.best_params_['n_neighbors']}")
print(f"- Weights: {grid_search.best_params_['weights']}")
print(f"- Metric: {grid_search.best_params_['metric']}")
print(f"- Best cross-validation F1-score: {grid_search.best_score_:.4f}")

# Train optimized KNN classifier with best parameters
knn_classifier = grid_search.best_estimator_
print("\nOptimized KNN training completed!")
```

Best parameters found	Value
Number of neighbors (K)	13
Weights	Distance
Metric	Euclidean
Best cross-validation F1-score	43.43%

Table 3. The optimal KNN model used 13 neighbours, distance-based weighting and the Euclidean metric. It achieved a cross-validation F1-score of 43.43%, showing a moderate balance between precision and recall. .

The optimised KNN model was evaluated on scaled test data using the code provided below.

```
# Make predictions on SCALED test data
knn_pred = knn_classifier.predict(X_test_scaled)
knn_pred_proba = knn_classifier.predict_proba(X_test_scaled)[:, 1]

# Calculate metrics
knn_accuracy = accuracy_score(y_test, knn_pred)
knn_precision = precision_score(y_test, knn_pred)
knn_recall = recall_score(y_test, knn_pred)
knn_f1 = f1_score(y_test, knn_pred)
knn_auc = roc_auc_score(y_test, knn_pred_proba)

print(f"\n=== OPTIMIZED KNN PERFORMANCE METRICS ===")
print(f"Accuracy: {knn_accuracy:.4f}")
print(f"Precision: {knn_precision:.4f}")
print(f"Recall: {knn_recall:.4f}")
print(f"F1-Score: {knn_f1:.4f}")
print(f"ROC-AUC: {knn_auc:.4f}")

# Show improvement from hyperparameter tuning
print(f"\n=== COMPARISON: Before vs After Tuning ===")
print("(Note: Previous k=11 was arbitrary choice)")
print(f"Optimized model uses k={grid_search.best_params_['n_neighbors']} with {grid_search.best_params_['weights']} wei
```

Metric	Percentage
Accuracy:	81.0%
Precision	63.2%
Recall	33.9%
F1-Score	44.1%
ROC-AUC	73.4%

Table 4.Optimized KNN performance metrics. The optimised KNN model shows good overall performance with 81.02% accuracy and a ROC-AUC of 73.04%. However, its low recall of 33.91% indicates it misses many actual defaulters despite having high precision.

The code below shows a graph comparing the number of neighbours with the average F1-score for KNN. It highlights how different weighting methods and distance measures affect performance and helps identify which combination gives the best results.

```
# Visualize hyperparameter tuning results
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd  # only if not already imported

# Extract results from GridSearchCV
results_df = pd.DataFrame(grid_search.cv_results_)

# Create visualization of k values vs F1 scores
plt.figure(figsize=(15, 5))

# Plot 1: k values vs F1 score for different weight methods
plt.subplot(1, 3, 1)
for weight in ['uniform', 'distance']:
    for metric in ['euclidean', 'manhattan']:
        mask = (results_df['param_weights'] == weight) & (results_df['param_metric'] == metric)
        subset = results_df[mask]
        plt.plot(subset['param_n_neighbors'], subset['mean_test_score'],
                 marker='o', label=f'{weight}+{metric}')

plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean F1 Score')
plt.title('Hyperparameter Tuning: k vs F1 Score')
plt.legend()
plt.grid(True, alpha=0.3)
```

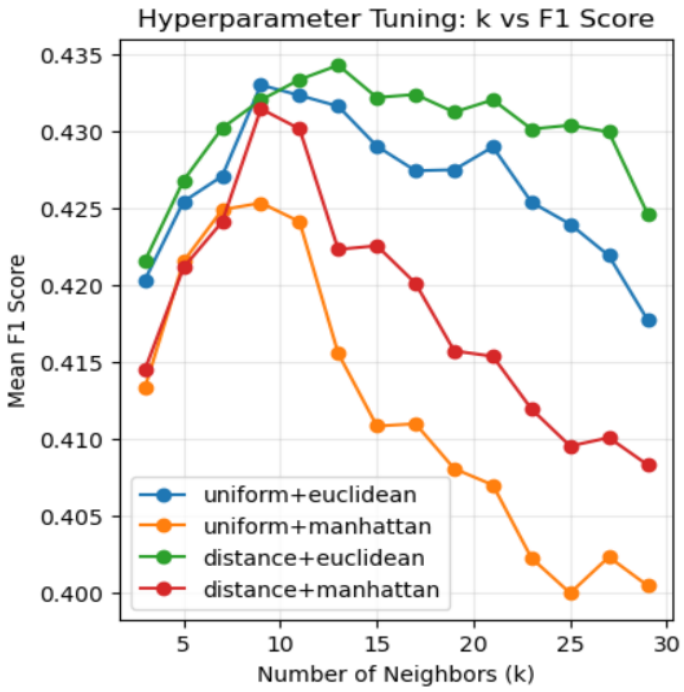

Figure 7.Visualization of hyperparameter tunning result for K vs F1-Score

5.0 RESULTS ANALYSIS AND DISCUSSION

5.1 Performance Metrics

Accuracy alone was insufficient due to the imbalanced nature of the credit default data, so multiple metrics were used for balanced evaluation. Precision, recall, and F1-score helped

assess the model's ability to correctly detect defaulters while minimising misclassifications, and ROC-AUC measured overall discrimination performance.

5.2 Algorithm Performance Comparison

The following code was used to compare the two algorithms.

```
# Comprehensive Algorithm Comparison and Analysis

print("=== ALGORITHM PERFORMANCE COMPARISON ===")

# Create comparison DataFrame
comparison_results = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'ROC-AUC'],
    'Decision Tree': [dt_accuracy, dt_precision, dt_recall, dt_f1, dt_auc],
    'KNN': [knn_accuracy, knn_precision, knn_recall, knn_f1, knn_auc]
})

print(comparison_results.round(4))

# Determine winners for each metric
print(f"\n=== PERFORMANCE ANALYSIS ===")
for metric in ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'ROC-AUC']:
    dt_score = comparison_results[comparison_results['Metric'] == metric]['Decision Tree'].iloc[0]
    knn_score = comparison_results[comparison_results['Metric'] == metric]['KNN'].iloc[0]
    winner = "KNN" if knn_score > dt_score else "Decision Tree" if dt_score > knn_score else "Tie"
    print(f"{metric}: {winner} ({max(dt_score, knn_score):.4f})")

# Overall performance assessment
dt_avg = comparison_results['Decision Tree'].mean()
knn_avg = comparison_results['KNN'].mean()
overall_winner = "KNN" if knn_avg > dt_avg else "Decision Tree"

print(f"\n=== OVERALL ASSESSMENT ===")
print(f"Decision Tree Average Score: {dt_avg:.4f}")
print(f"KNN Average Score: {knn_avg:.4f}")
print(f"Overall Winner: {overall_winner}")
```

Metric	Decision Tree	KNN
Accuracy	70.68 %	81.0%
Precision	39.5%	63.2%
Recall	61.2%	33.9%
F1-Score	48.0%	44.1%
ROC-AUC	73.3%	73.4%

Table 5. Comparison of performance matrix

Metric: (algorithm)	Value
Accuracy: (KNN)	81.02%
Precision: (KNN)	63.20%
Recall: (Decision Tree)	61.27%
F1-Score: (Decision Tree)	48.04%
ROC-AUC: (KNN)	73.40%

Table 6. Performance analysis of both models

The results show a clear difference between the two models. KNN achieved higher accuracy (81.02%) and precision (63.20%), so it was more reliable when predicting default. It also slightly performed better in ROC-AUC at 73.40% compared to 73.3% for the Decision Tree (Table 6), showing a small improvement in distinguishing defaulters from non-defaulters.

5.3 Business Benefits and Real-World Applications

Using machine learning classification will benefit the credit providers to better predict and manage default risk. These models will improve risk detection, support smarter lending decisions and help the business group customers based on risk levels. This can reduce financial losses and increase profitable lending. Automating the lending process will improve efficiency and support compliance with regulations. Although accuracy can still be improved, these methods are an important step towards better credit risk control and long-term financial stability for the business.

5.4 Decision-Making Framework – Business Interpretation

A risk-based decision framework was created to help guide credit approval using predicted default probabilities. Customers with over 70% risk are treated as high risk and should be declined or asked to provide collateral to reduce losses. Those with 30 to 70% risk fall into the medium group and should receive stricter conditions, such as lower limits or higher interest rates. Customers below 30% are low risk and can be approved normally. In the test data, 17.2% were high risk, 47.0% medium risk and 35.8% low risk, helping businesses adjust strategies efficiently.

5.5 Impact on Process Improvement

The classification insights could help improve business processes in credit risk management. They could support smarter credit approval by using data instead of guesswork, reduce financial losses by identifying likely defaulters, improve efficiency by focusing resources on important cases, and allow better customer segmentation so products can be tailored to different risk groups.

6.0 CONCLUSION

This study examined the effectiveness of machine learning models, specifically Decision Tree and K-Nearest Neighbours (KNN), in predicting credit card default and identifying high-risk customers. Both models demonstrated moderate performance, with F1-scores below 45%, indicating limited reliability in distinguishing defaulters from non-defaulters. Hyperparameter tuning offered little improvement for KNN, suggesting it was poorly suited to the dataset. The Decision Tree model performed comparatively better and provided greater interpretability, leading to its recommendation for deployment, although its accuracy remained only relatively superior.

Overall, the study confirms that machine learning can support credit risk assessment through structured, data-driven decision-making. However, due to performance limitations, the recommended approach should serve as a decision-support tool rather than a fully automated system. The findings highlight the need for further enhancement, including advanced modelling techniques and improved handling of data imbalance, to achieve reliable credit risk prediction.

RECOMENDATIONS

The following actionable recommendations can be considered by the business.

- Implement Risk-Based Credit Approval Policies: This will minimise exposure to high-risk borrowers while maximising safe lending.
- Introduce Dynamic Credit Limit Management: This controls risky spending behaviour and enhances responsible credit use.
- Deploy Early Intervention Strategies: This reduces missed payments and long-term bad debt.

REFERENCES

Alamri, M., & Ykhlef, M. (2024). Hybrid undersampling and oversampling for handling imbalanced credit card data. *IEEE Access, 12*, 14050–14060. https://doi.org/10.1109/ACCESS.2024.3357091

Arora, S. (2022). Prediction of credit card defaults through data analysis and machine learning techniques. *Materials Today: Proceedings, 51*, 110–117. https://doi.org/10.1016/j.matpr.2021.03.514

Bhandary, R., & Ghosh, B. K. (2025). Credit card default prediction: An empirical analysis on predictive performance using statistical and machine learning methods. *Journal of Risk and Financial Management, 18*(1), 23. https://doi.org/10.3390/jrfm18010023

Chen, Y., Zhou, Z., & Chen, W. (2024). Interpretable machine learning for imbalanced credit scoring: On the stability of explanations under class imbalance. *European Journal of Operational Research*. Advance online publication. https://doi.org/10.1016/j.ejor.2023.10.034

FinRegLab. (2023). *Machine learning explainability & fairness: Insights from consumer lending*. https://finreglab.org/research/machine-learning-explainability-fairness-insights-from-consumer-lending/

Han, D. (2024). Personal credit default prediction fusion framework based on self-attention and cross-network algorithms. *Engineering Applications of Artificial Intelligence, 133*, Article 108137. https://doi.org/10.1016/j.engappai.2024.108137

Liu, J., Zhang, S., & Fan, H. (2022). A two-stage hybrid credit risk prediction model based on XGBoost and graph-based deep neural network. *Expert Systems with Applications, 195*, 116624. https://doi.org/10.1016/j.eswa.2022.116624

Mariscal, C., Lansky, M., & Laengle, S. (2024). Implementing and analyzing fairness in banking credit scoring. *Procedia Computer Science, 234*, 1492–1499. https://doi.org/10.1016/j.procs.2024.03.150

Xu, T. (2024). Novel embedding model predicting the credit card's default using neural network optimized by harmony search algorithm and vortex search algorithm. *Heliyon, 10*(9), e21656. https://doi.org/10.1016/j.heliyon.2024.e21656

Yeh, I.-C., & Lien, C.-H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications, 36*(2), 2473–2480. https://doi.org/10.1016/j.eswa.2007.12.020

Yin, W. (2023). Stacking ensemble method for personal credit risk assessment in peer-to-peer lending. *Applied Soft Computing, 142*, 110302. https://doi.org/10.1016/j.asoc.2023.110302

Task 2- Customer SKU Segmentation Using K-Means and Hierarchical Clustering

1.0 INTRODUCTION

1.1 Background

Warehouse Stock Keeping Unit (SKU) management is increasingly challenging because warehouses handle diverse products with varying demand, value, shelf life, pallet size, weight, units per pallet and picking frequency — all influencing storage layout and picking efficiency (Silva et al., 2022). Traditional inventory methods based solely on annual consumption value often ignore these interacting factors, leading to poor storage placement and excessive picking travel (Silva et al., 2022). Recent research demonstrates that multi-criteria classification schemes using decision-making and machine learning produce more accurate SKU segmentation and better control policies, substantially boosting space utilization, overall service levels and reducing waste (Khanorkar & Kane, 2023; Bergsma et al., 2025).

1.2 Problem Statement

Traditional ABC analysis and similar methods are still common in warehouses, but they mainly focus on demand or value and ignore other important factors. These include how quickly a product expires, pallet space limits and how often items are shipped out. Because of this, they cannot clearly separate high value but slow-moving items, large low value products, or short shelf-life goods that need fast handling. Even when more factors are used, how they are chosen and weighted is often unclear and inconsistent. As a result, many warehouses do not have a clear, data-based system that reflects both the business value and practical handling needs.

1.3 Aim and Objectives

Aim:

The aim of this task is to apply unsupervised machine learning techniques to identify meaningful clusters of SKUs based on their demand and physical characteristics.

Objectives:

The specific objectives are to:

- Perform K-Means clustering on the dataset to partition SKUs into distinct groups and evaluate cluster quality.
- Perform Agglomerative clustering on the dataset to partition SKUs into distinct groups and evaluate cluster quality.
- Compare and evaluate the two algorithms based on commonly used metrics.
- Interpret the resulting clusters to derive practical insights for inventory planning, warehouse optimisation and outbound decision-making.

1.4 Related Research

Recent studies show that machine learning is becoming more important in warehouse management. De Assis et al. (2024) explain how it is used for deciding storage locations, improving picking and finding patterns in warehouse data. Simone et al. (2022) show that clustering methods can group similar warehouse activities and reveal useful patterns in system data. In inventory classification, Kmiecik (2023) shows that dynamic ABC affects storage zones in warehouses. Other studies confirm that using rich SKU data and prediction models improves decision-making and reduces stockouts.

1.5 Research Gap
Although clustering and multi-criteria methods are used for SKU classification, many studies still focus mainly on demand, value and service level. Factors like expiry dates, pallet size and outbound frequency are often ignored. Few studies compare methods such as K-Means and Hierarchical Clustering, and results are rarely linked to practical warehouse decisions, reducing their real operational value.

1.6 Research Question
How can K-Means and Hierarchical Clustering group SKUs by demand and size to improve warehouse efficiency?

2.0 DATASET
The dataset was sourced from the UC Irvine Machine Learning Repository and provided by Trialto Latvia Ltd, a logistics company. It contains 2,279 rows and 8 numerical variables, with each row representing a unique SKU. All data is numeric, making it suitable for clustering. The dataset comes from real operational records and reflects typical warehouse activity. It is anonymised and does not include any personal or customer information, ensuring privacy is maintained.

2.1 Ethical and Social Considerations
The dataset involves low ethical risk as it contains no personal data, but it should still be handled responsibly. Company information is treated as confidential and used only for this study. All data steps are recorded for transparency and results are reported honestly. Improving warehouse efficiency may reduce waste but could affect jobs. It is assumed data laws such as GDPR were followed, and good practices like secure storage and data minimisation are applied.

3.0 EDA AND DATA PREPROCESSING

3.1 Brief description of dataset
The dataset includes seven numerical features that describe SKU characteristics better than basic ABC analysis. These cover unit price, expiry date, outbound order count, total outbound quantity, pallet weight, pallet height and units per pallet. These factors affect storage, handling and demand patterns. The independent variables explain product behaviour, while outbound measures show operational performance. Although clustering does not predict outcomes, these measures help interpret the clusters and understand their practical value in warehouse operations.

3.2 Exploratory data analysis
EDA steps included inspecting the data structure, generating the statistical summary and detecting invalid and missing values as seen in the codes and outputs below.

```
# Getting statistical summary of dataset
dataset.describe()
```

	ID	Unitprice	Expire date	Outbound number	Total outbound	Pal grossweight	Pal height	Units per pal
count	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000
mean	1140.000000	4.269402	410.371654	235.976305	731.701053	192.939582	0.672798	755.563405
std	658.034953	14.449000	240.875419	700.230685	2146.029848	164.616813	0.552117	6278.437915
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	570.500000	0.000000	365.000000	0.000000	0.000000	60.000000	0.000000	32.000000
50%	1140.000000	1.293800	547.000000	1.000000	3.000000	167.680000	0.840000	108.000000
75%	1709.500000	4.545000	547.000000	45.000000	419.500000	277.560000	1.020000	384.000000
max	2279.000000	518.592000	734.000000	6325.000000	26411.000000	907.200000	2.160000	200000.000000

Figure 1. Summary statistics

```
#Checking how many missing values are in each column of your DataFrame.

dataset.isnull().sum()
```

```
ID                0
Unitprice         0
Expire date       0
Outbound number   0
Total outbound    0
Pal grossweight   0
Pal height        0
Units per pal     0
dtype: int64
```

Checking for null values, no null value was identified.

3.3 Data Preprocessing

The data was cleaned by removing rows where UnitPrice was zero, as these likely showed missing or invalid values. The ID column was dropped since it did not add useful information. Log transformation was applied to highly skewed features to improve distribution. Any unrealistic zero values in physical attributes were replaced with the median to keep the data realistic and consistent as seen the codes below.

```
# Removing rows where Unitprice equals zero
dataset_cleaned = dataset[dataset["Unitprice"] != 0]
```

Step 1. Removing rows where Unitprice equals zero

```
# Dropping the ID column since it's not useful for clustering, this now gives us the "final dataset"
dataset_final = dataset_cleaned.drop(columns=["ID"])
```

Step 2. Dropping the ID column

```
# Log-transforming skewed features to reduce skewness
dataset_final['Outbound number']   = np.log1p(dataset_final['Outbound number'])
dataset_final['Total outbound']    = np.log1p(dataset_final['Total outbound'])
dataset_final['Unitprice']         = np.log1p(dataset_final['Unitprice'])
dataset_final['Pal grossweight']   = np.log1p(dataset_final['Pal grossweight'])
dataset_final['Units per pal']     = np.log1p(dataset_final['Units per pal'])
```

Step 3. Log-transforming skewed features

```
# Replacing 0s in physical attributes (which may be invalid) with median values
for col in ['Pal grossweight', 'Pal height', 'Units per pal']:
    dataset_final[col] = dataset_final[col].replace(0, np.nan)
    dataset_final[col] = dataset_final[col].fillna(dataset_final[col].median())
```

Step 4. Replacing zeros in physical attributes

The code below was used to show a summary after preprocessing.

```
[5]: # Getting statistical summary of dataset
     dataset.describe()
```

	ID	Unitprice	Expire date	Outbound number	Total outbound	Pal grossweight	Pal height	Units per pal
count	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000	2279.000000
mean	1140.000000	4.269402	410.371654	235.976305	731.701053	192.939582	0.672798	755.563405
std	658.034953	14.449000	240.875419	700.230685	2146.029848	164.616813	0.552117	6278.437915
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	570.500000	0.000000	365.000000	0.000000	0.000000	60.000000	0.000000	32.000000
50%	1140.000000	1.293800	547.000000	1.000000	3.000000	167.680000	0.840000	108.000000
75%	1709.500000	4.545000	547.000000	45.000000	419.500000	277.560000	1.020000	384.000000
max	2279.000000	518.592000	734.000000	6325.000000	26411.000000	907.200000	2.160000	200000.000000

Figure 2. Summary statistics of the dataset after preprocessing.

Seaborn pairplots were used to show how the numerical variables relate to each other and their individual distributions. The histograms were generated with the code below and scatter plots are displayed in Figure 1.

```
# scatterplots for each pair of variables and histogram for each variable too
X = dataset_final.iloc[:,1:8]
sns.pairplot(X)
```

Figure 3. Scatterplots for each pair of variables

The dataset was standardised using StandardScaler to bring all variables onto the same scale.

```
#Scaling the data using standard scaler to standardize the dataset

from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()
X = sc_X.fit_transform(X)
```

4.0 CLUSTERING ALGORITHMS IMPLEMENTATION

4.1 Description of Algorithms

Two clustering methods were used: K-Means and hierarchical clustering. K-Means groups data into a set number of clusters by assigning points to the nearest centre and updating these centres to reduce differences within each group (Ikotun et al., 2023). Hierarchical clustering organises data based on similarity by merging or splitting groups, creating a tree-like structure called a dendrogram (Yu et al., 2023). Both methods were suitable because all variables are continuous numerical data, making distance-based clustering appropriate.

4.2 Algorithm I- (K-Means Clustering Implementation)

4.2.1 The Elbow Method

The elbow method was used to choose the best number of clusters for K-Means as seen in the code below. It measures the within-cluster sum of squares (WCSS) for different k values and identifies the point where the decrease starts to level off, showing the most suitable cluster

number.

```
#Using Elbow method: compute inertia (wcss) for k = 1 to 10 to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot elbow curve
plt.figure()
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (K)')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()
```

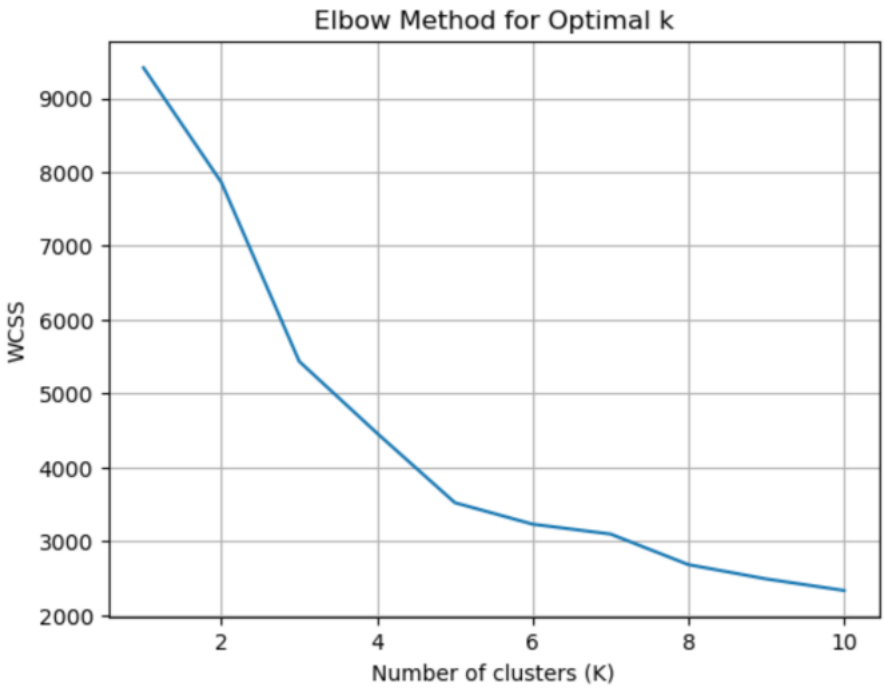

Figure 2. Elbow curve of number of clusters vs the WCSS to get the optimate number of clusters.

The elbow curve shows a sharp drop in WCSS from k = 1 to k = 3, then begins to flatten, meaning extra clusters add little improvement. This suggests k = 3 is the best choice. Using this value, the code created a K-Means model and applied fit_predict() to assign each SKU to a cluster.

```
[ ]: #Fitting K-Means to the dataset
|
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Principal Component Analysis (PCA) was used to reduce the dataset from eight features to two, keeping the main patterns and making the data easier to visualise and understand.

```
#Reducing the dimentionality before we can visualise

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

pca.explained_variance_ratio_
```

The next step checked how much of the original data variation was kept by the two principal components. This was done by using the explained_variance_ratio and adding the values together to find the total retained variance.

```
#Identifying how much of the original variance
sum(pca.explained_variance_ratio_)
```

```
np.float64(0.5882176969250004)
```

The results show that over 58% of the original variance is explained by the two retained dimensions.

4.3 Cluster Visualization

A scatter plot was generated using a loop to show K-Means clusters in two dimensions using the reduced data and cluster labels from the code below.

```
#Visualising the clusters using a scatterplot

colours = ['red', 'blue', 'green']

plt.figure(figsize =(8,8))
for i in range(3):
    plt.scatter(X_reduced[y_kmeans == i, 0], X_reduced[y_kmeans == i, 1],
                s =100, c = colours[i], label = 'Cluster' +str(i+1))
plt.title('Clusters of Products (K-Means, k=3)')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend()
plt.show()
```

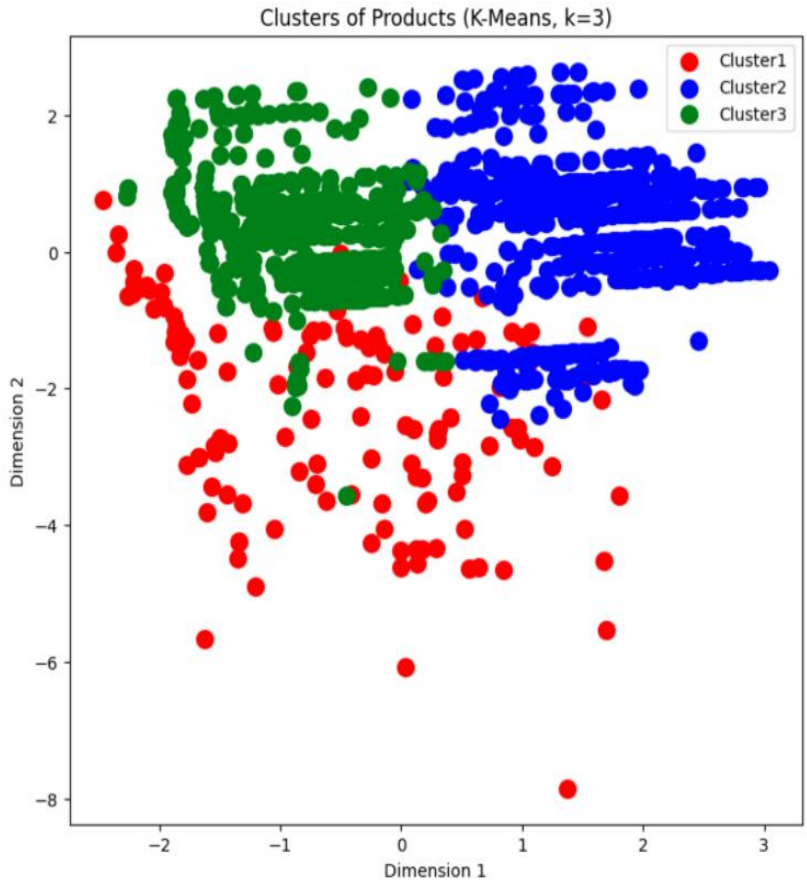

Figure 3. A visual representation of the 3 clusters from the dataset

4.4 Algorithm II- (Hierarchical Clustering Implementation)

The code below creates a dendrogram to show how hierarchical clustering groups the data. It uses Ward linkage, which merges clusters by reducing variation within them. The input X_reduced is the two-dimensional data after PCA. The linkage function calculates how points are combined based on distance, and the dendrogram then displays this process. The x-axis shows individual SKUs, while the y-axis shows the distance at which clusters join.

```
#Using the dendrogram to find the optimal number of clusters

import scipy.cluster.hierarchy as sch

plt.figure(figsize =(15, 6))
dendrogram = sch.dendrogram(sch.linkage(X_reduced, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Products')
plt.ylabel('Euclidean distances')
plt.show()
```

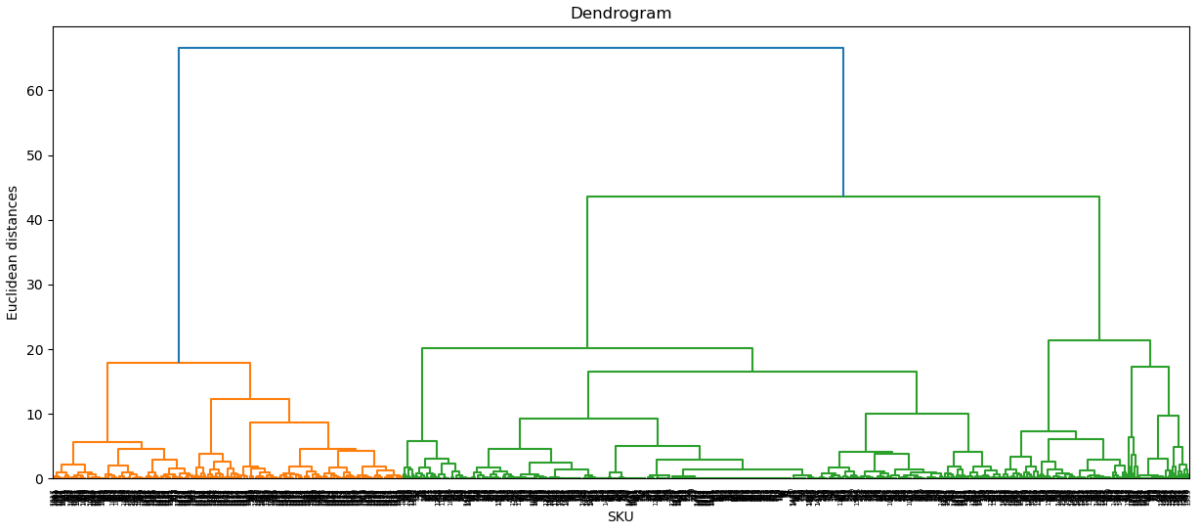

Figure 4. Dendrogram for the number of clusters

4.4.1 Key observation

The dendrogram in Figure 4 shows a clear jump in distance near the top of the chart. One merge happens around 68–70, while others appear near 50 and between 20 and 30. Drawing a horizontal line at 50 cuts the tree into three main groups. This suggests that three clusters best represent the data, which is also confirmed by the three different colours shown in the diagram.

4.4.2 Fitting Hierarchical clustering to the dataset

Hierarchical clustering was carried out on the two-dimensional dataset (X_reduced). The model was set to create three clusters using Euclidean distance and Ward linkage. The fit_predict () method was then used to assign each data point to a cluster. These cluster results were visualised and shown in Figure 5, making it easier to see how the SKUs were grouped based on their similarities.

```
#Fitting Heirarchical clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters =3, metric = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X_reduced)
```

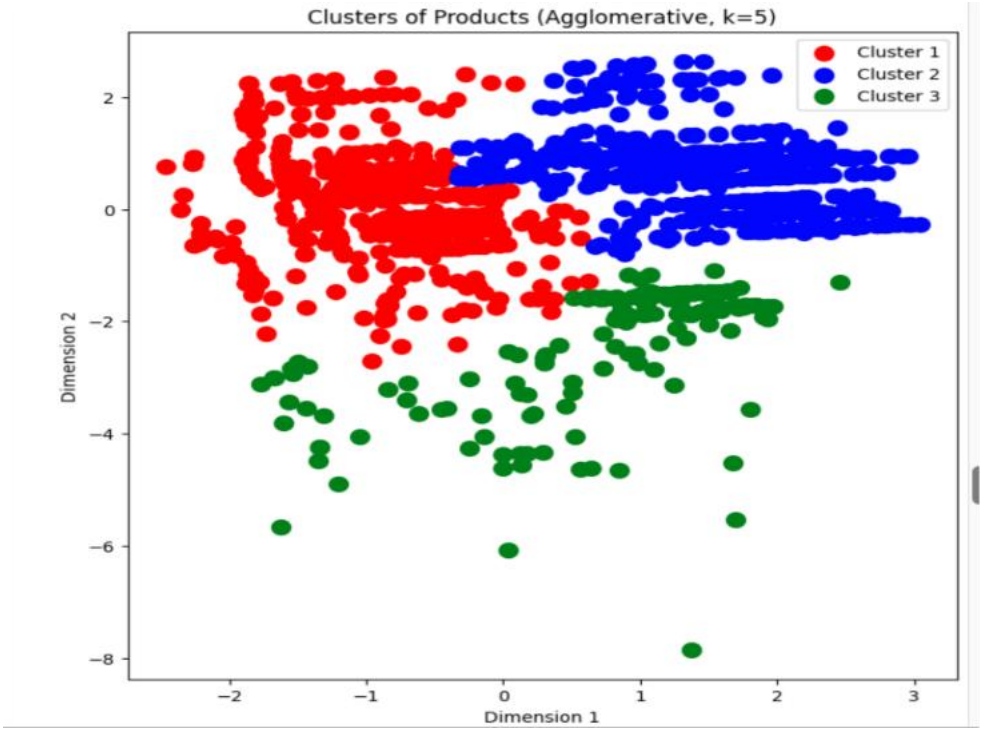

Figure 5. Agglomerative clustering of the dataset

.

5.0 RESULTS ANALYSIS AND DISCUSSION

5.1 Performance Metrics

This section compares the performance of K-Means and agglomerative clustering. The evaluation uses Calinski–Harabasz, Davies–Bouldin, Silhouette Score, Inertia (WCSS) and Adjusted Rand Index. These measures assess how well the clusters are formed without needing true labels and widely used, making them suitable for judging and comparing the effectiveness of both clustering methods.

5.2 Algorithm Performance Comparison

K-Means was evaluated using internal metrics that measure cluster compactness, separation and consistency, including Inertia. Agglomerative Clustering used the same measures except Inertia, as it is not centroid-based. The Adjusted Rand Index was used to compare how similar the results were, where values close to 1 show strong agreement, helping clearly compare

overall performance.

```
# Import evaluation metrics
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
from sklearn.metrics import adjusted_rand_score

print("=== CLUSTERING PERFORMANCE EVALUATION ===\n")

# Calculate metrics for K-Means (k=3)
kmeans_silhouette = silhouette_score(X, y_kmeans)
kmeans_calinski = calinski_harabasz_score(X, y_kmeans)
kmeans_davies = davies_bouldin_score(X, y_kmeans)
kmeans_inertia = kmeans.inertia_

print("K-MEANS CLUSTERING (k=3):")
print(f"• Silhouette Score: {kmeans_silhouette:.4f}")
print(f"• Calinski-Harabasz Index: {kmeans_calinski:.2f}")
print(f"• Davies-Bouldin Index: {kmeans_davies:.4f}")
print(f"• Inertia (WCSS): {kmeans_inertia:.2f}")
print()

# Calculate metrics for Agglomerative Clustering (k=3)
agglo_silhouette = silhouette_score(X, y_agglo)
agglo_calinski = calinski_harabasz_score(X, y_agglo)
agglo_davies = davies_bouldin_score(X, y_agglo)

print("AGGLOMERATIVE CLUSTERING (k=3):")
print(f"• Silhouette Score: {agglo_silhouette:.4f}")
print(f"• Calinski-Harabasz Index: {agglo_calinski:.2f}")
print(f"• Davies-Bouldin Index: {agglo_davies:.4f}")
print()

# Calculate agreement between algorithms
agreement = adjusted_rand_score(y_kmeans, y_agglo)
print("ALGORITHM COMPARISON:")
print(f"• Adjusted Rand Index (Agreement): {agreement:.4f}")
print()

# Cluster size distributions
print("CLUSTER SIZE DISTRIBUTIONS:")
kmeans_counts = np.bincount(y_kmeans)
agglo_counts = np.bincount(y_agglo)

print("K-Means cluster sizes:", kmeans_counts)
print("Agglomerative cluster sizes:", agglo_counts)
```

Metric	K-Means clustering	Agglomerative clustering
Silhouette Score	0.3699	0.3522
Calinski-Harabasz	572.67	525.22
Davies-Bouldin Index	1.1401	1.2643
Inertia (WCSS)	5437.51	
Adjusted Rand Index (Agreement)	0.6779	

Table 1. Performance comparison of K-means and agglomerative clustering

K-Means showed better overall clustering results. Its higher Silhouette Score and Calinski–Harabasz Index mean the clusters are more clearly separated and well defined. The lower Davies–Bouldin Index also suggests less overlap, while the Inertia value shows good compactness.

Agglomerative Clustering performed slightly worse, with weaker separation and definition. However, the Adjusted Rand Index shows a strong agreement between both methods, even though they cluster data differently.

5.3 Business Benefits and Real-World Applications

Clustering offers the following real-world benefits.

- It supports better inventory management by grouping products.
- It improves storage and demand planning.
- It also increases efficiency by optimising warehouse layout, picking routes and resource use, while helping reduce costs through better space use and handling.

5.4 Decision-Making Framework – Business Interpretation

Clustering supports strategic decisions by;

- Helping managers identify underperforming product groups and applying consistent pricing strategies.
- Operationally enabling better staff planning, shared quality control processes, and more accurate seasonal inventory.
- In terms of technology investment, clustering guides automation priorities for high-volume products, supports the selection of appropriate storage systems, and enables the implementation of cluster-specific tracking and management systems, improving overall efficiency and long-term planning.

5.5 Impact on Process Improvement

- Clustering could enhance workflow optimization by enabling standardized handling processes, simplifying staff cross-training, and improving performance monitoring through cluster-based KPIs.
- In customer service, clustering could support faster order fulfilment through efficient picking and packing.
- Clustering could improve product recommendations by identifying related items, and optimizes delivery through smarter shipping strategies.
- Overall, could help increase speed, accuracy, and customer satisfaction while maintaining operational efficiency.

6.0 CONCLUSION

This study explored using unsupervised learning to group SKUs for better warehouse and inventory management. K-Means and Hierarchical Clustering were applied using demand and physical features, showing clear differences in SKU behaviour. Overall, K-Means performed better, producing more clearly defined and separated clusters. Hierarchical Clustering showed weaker separation, with less distinct groupings.

The results divided SKUs by demand, size and value, supporting more targeted storage strategies. Fast-moving items should be placed in key locations for easy access, while slow or bulky products suit secondary areas. High-value items benefit from secure handling. These insights help improve warehouse layout, stock control and outbound efficiency.

RECOMMENDATIONS

Below are practical, actionable ways the organisation can use the SKU clustering insights to improve performance and achieve specific goals:

- Optimise Warehouse Layout: This leads to shorter picking routes and improved order processing speed.
- Improve Inventory Control & Stock Levels: The result will be better stock availability and reduced inventory waste.
- Design Cluster-Based Picking Strategies: this leads to faster picking and less errors.
- Improve Space Utilisation: This leads to better use of warehouse space.

REFERENCES

Bergsma, R., de Ruijt, C., & Bhulai, S. (2025). A systematic review of machine learning approaches in inventory control optimization. *Operations Research Perspectives*. Advance online publication. https://doi.org/10.1016/j.orp.2025.100367

De Assis, R. F., Faria, A. F., Thomasset-Laperrière, V., Santa-Eulalia, L. A., Ouhimmou, M., & de Paula Ferreira, W. (2024). Machine learning in warehouse management: A survey. *Procedia Computer Science, 232*, 2790–2799. https://doi.org/10.1016/j.procs.2024.02.096

Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., & Jia, H. (2023). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences, 622*, 178–210. https://doi.org/10.1016/j.ins.2022.11.139

Khanorkar, Y., & Kane, P. V. (2023). Selective inventory classification using ABC classification, multi-criteria decision making techniques, and machine learning techniques. *Materials Today: Proceedings, 72*, 1270–1274. https://doi.org/10.1016/j.matpr.2022.09.298

Kmiecik, M. (2023). Dynamic ABC analysis for assortment management in 3PL. *Zeszyty Naukowe Politechniki Śląskiej. Organizacja i Zarządzanie, 189*, 269–295. https://doi.org/10.29119/1641-3466.2023.189.18

Silva, A., Roodbergen, K. J., Coelho, L., & Darvish, M. (2022). Estimating optimal ABC zone sizes in manual warehouses. *International Journal of Production Economics, 252*, 108579. https://doi.org/10.1016/j.ijpe.2022.108579

Simone, F., Nakhal Akel, A. J., & Patriarca, R. (2022). Supporting warehouse management through machine learning. *Summer School Francesco Turco Proceedings*, 22–29. https://www.summerschool-aidi.it/images/papers/session_7_2022/ID_057.pdf

Yu, B., Zheng, Z., & Dai, J. (2023). K-DGHC: A hierarchical clustering method based on K-dominance granularity. *Information Sciences, 632*, 232–251. https://doi.org/10.1016/j.ins.2023.03.012

Task 3- Evaluating Customer Sentiment Through Predictive Modelling to Support Operational and Strategic Business Decisions

1.0 INTRODUCTION

1.1 Background

The increasing use of online platforms such as social media, product review sites and discussion forums has resulted in large amounts of text expressing people's opinions and emotions. Sentiment analysis, also known as opinion mining, is used to automatically classify this text as positive, negative or neutral, and sometimes by emotional tone, allowing unstructured data to be converted into useful information. This study applies sentiment analysis to dataset to identify sentiment polarity, explore how it varies across different attributes, supporting clearer and more informed decision-making.

1.2 Problem Statement

Although large amounts of text data are available from reviews, comments and social media, turning this information into useful insight is still difficult. In the current dataset, there is no clear automatic way to detect sentiment or connect it to factors such as topic, category or time. While text and related details are present, it is unclear how sentiment is spread or how it changes. This study explores how sentiment varies across these features and identifies what is most linked to negative sentiment.

1.3 Aim and Objectives

Aim:

The aim of this study is to analyse text data using a Naive Bayes model to classify and evaluate positive and negative sentiment.

Objectives:

The objectives of this study are to:

- Explore sentiment patterns using visual data analysis.
- Prepare and model the text through preprocessing and a Naive Bayes classifier.
- Assess how well the model separates positive and negative sentiment.

1.4 Related Research

Sentiment analysis is now used in many fields and has developed from simple word list methods to more advanced machine learning and deep learning approaches. These newer techniques are more accurate but still struggle with informal language, sarcasm, different languages and changing meanings across topics (Zhang & Zhou, 2023; Li et al., 2024). Smith et al. (2023) found that longer online reviews often contain more negative opinions, especially when specific product features are discussed, and that adding extra details improves decision-making. Kim and Park (2024) showed that sentiment on social media changes quickly after policy news but later stabilises. Brown and Green (2022) highlight that many studies do not fully explore links between sentiment and metadata.

1.5 Research Gap

Many studies use sentiment analysis with models like Naive Bayes to classify text as positive or negative. However, most focus mainly on accuracy and do not consider the balance between positive and negative comments in the data. When datasets are merged, sentiment proportions can change, but this is often ignored. As a result, there is limited understanding of how sentiment imbalance affects Naive Bayes performance, highlighting an important gap for future research

1.6 Research Question

What is the distribution of positive and negative sentiment in the combined dataset, and how accurately can a Naive Bayes classifier predict sentiment based on textual features?

2.0 DATASET

The dataset constitutes three sentiment analysis datasets from Amazon, IMDB and Yelp which contains short text reviews labelled as positive (1) or negative (0). The Amazon dataset focuses on mobile phone and accessory reviews, reflecting customer opinions on product quality and usability. The IMDB dataset consists of movie reviews expressing opinions about films, acting, and storytelling, making it suitable for classifying film sentiment. The Yelp dataset contains restaurant reviews discussing food, service, and overall dining experiences, useful for real-world opinion mining. All three datasets share a simple structure and are commonly used for training and evaluating machine learning models in binary sentiment classification.

2.1- Ethical/Social and Legal Consideration

These datasets raise a few ethical, social, and legal concerns that should be considered. Ethically, the reviews may contain bias because they only reflect the opinions of certain users and not everyone. This can cause unfair or unbalanced results when building sentiment models. The labels also simplify feelings into just positive or negative, which does not fully capture real human emotions. In addition, some comments include rude or offensive language, which can influence how systems learn to respond.

Socially, models trained on these datasets could be used in ways that unfairly judge people or their opinions. Automated systems might remove or promote reviews in a way that affects businesses and customers. Cultural differences in how people express opinions may also be ignored.

Legally, the reviews may be protected by copyright and owned by the platforms or users. Using this data without permission could break terms of service or privacy rules

3.0-EDA AND DATA PROCESSING

3.1 Description of the Dataset

The study used a combined sentiment dataset made up of labelled text data from three sources: IMDB movie reviews, Amazon product reviews, and Yelp restaurant reviews. These datasets contain short customer or user-written sentences that express an opinion, each already assigned a sentiment label.

The independent variable is the review text itself, which represents unstructured customer feedback. This text will be cleaned and processed before being converted into numerical features for model training.

The dependent variable is the sentiment label, which indicates whether each review is positive (1) or negative (0). The dataset uses textual review content as the input feature and sentiment classification as the output target for the model.

3.2 Exploratory Data Analysis

The three review datasets were uploaded and combined into one clean dataset. Basic information was displayed to show the total dataset size as well as the number of samples from each dataset. The following code was used to display load, combine and display basic information about the dataset.

```
# Load the datasets
def load_dataset(file_path, source_name):
    """Load dataset from file and add source column"""
    df = pd.read_csv(file_path, sep='\t', header=None, names=['text', 'sentiment'])
    df['source'] = source_name
    return df

# Load all three datasets
amazon_df = load_dataset('sentiment+labelled+sentences (1)/sentiment labelled sente
imdb_df = load_dataset('sentiment+labelled+sentences (1)/sentiment labelled sentenc
yelp_df = load_dataset('sentiment+labelled+sentences (1)/sentiment labelled sentenc

# Combine all datasets
combined_df = pd.concat([amazon_df, imdb_df, yelp_df], ignore_index=True)

print(f"Total dataset size: {len(combined_df)} samples")
print(f"Amazon samples: {len(amazon_df)}")
print(f"IMDB samples: {len(imdb_df)}")
print(f"Yelp samples: {len(yelp_df)}")

# Display basic information
print("\nDataset Info:")
print(combined_df.info())
print("\nFirst few samples:")
display(combined_df.head())
```

Dataset source	Number of samples
Amazon samples	1000
IMDB samples	748
Yelp samples	1000
Total dataset size	2748

Table 1. Summary of number of samples per data source.

Class distribution was analysed to identify how many reviews belong to each sentiment class (positive and negative) as well as visualising the sentiment distribution (pie chart) and the distribution by source (bar chart). The following code was used to perform the analysis.

```python
# Dataset Statistics and Distribution Analysis
# Check class balance
print("Class Distribution:")
print(combined_df['sentiment'].value_counts())
print()

# Visualize sentiment distribution
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Sentiment distribution pie chart
sentiment_counts = combined_df['sentiment'].value_counts()
axes[0].pie(sentiment_counts.values, labels=['Positive', 'Negative'], autopct='%1.1
axes[0].set_title('Sentiment Distribution')

# Sentiment distribution by source
source_sentiment = pd.crosstab(combined_df['source'], combined_df['sentiment'])
source_sentiment.plot(kind='bar', ax=axes[1], color=['red', 'green'])
axes[1].set_title('Sentiment Distribution by Source')
axes[1].set_xlabel('Source')
axes[1].set_ylabel('Count')
axes[1].legend(['Negative', 'Positive'])
axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

Sentiment type	Total Count
Positive (1)	1386
Negative (0)	1362

Table 2. Sentiment distribution

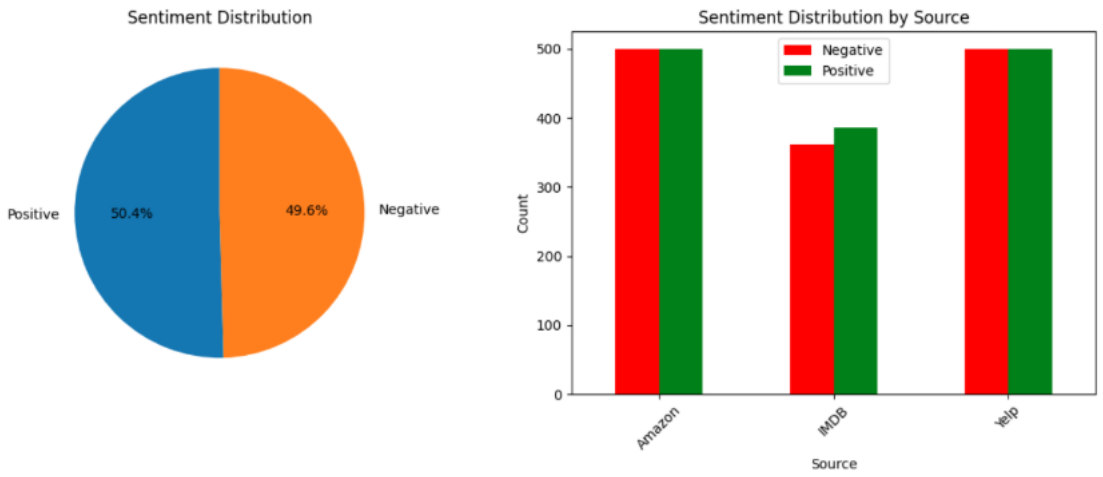

Figure 1. Sentiment distribution by type and data source

The WordCloud library was used to create two word clouds (positive and negative reviews), to highlight the most common words in each category. The code below was used to create the word cloud.

```python
# Word Cloud Generation
positive_texts = combined_df[combined_df['sentiment'] == 1]['text'].tolist()
negative_texts = combined_df[combined_df['sentiment'] == 0]['text'].tolist()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Positive sentiment word cloud
pos_text = ' '.join(positive_texts)
wordcloud_pos = WordCloud(width=400, height=300, background_color='white',
                          max_words=100).generate(pos_text)
ax1.imshow(wordcloud_pos, interpolation='bilinear')
ax1.set_title('Positive Reviews Word Cloud', fontsize=14, fontweight='bold')
ax1.axis('off')

# Negative sentiment word cloud
neg_text = ' '.join(negative_texts)
wordcloud_neg = WordCloud(width=400, height=300, background_color='white',
                          max_words=100).generate(neg_text)
ax2.imshow(wordcloud_neg, interpolation='bilinear')
ax2.set_title('Negative Reviews Word Cloud', fontsize=14, fontweight='bold')
ax2.axis('off')

plt.tight_layout()
plt.show()
```

Figure 2. Positive and negative reviews Word Cloud

Word frequencies were calculated to determine how often each word appeared in the dataset. Stop words, were removed to improve relevance. The dataset was then separated into positive and negative reviews, and the top 15 most frequent meaningful words in each category were identified and visualised to highlight key differences in language patterns using the code below.

```python
# Simple word frequency analysis
from collections import Counter

stop_words = set(stopwords.words('english'))

# Get all words from positive and negative reviews
positive_texts = combined_df[combined_df['sentiment'] == 1]['text'].tolist()
negative_texts = combined_df[combined_df['sentiment'] == 0]['text'].tolist()

# Simple word counting
pos_words = []
for text in positive_texts:
    words = [word.lower() for word in text.split() if word.isalpha() and len(word)
    pos_words.extend([w for w in words if w not in stop_words])

neg_words = []
for text in negative_texts:
    words = [word.lower() for word in text.split() if word.isalpha() and len(word)
    neg_words.extend([w for w in words if w not in stop_words])

# Get top words
pos_freq = Counter(pos_words).most_common(15)
neg_freq = Counter(neg_words).most_common(15)

# Plot word frequencies
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

pos_w, pos_c = zip(*pos_freq)
ax1.barh(range(len(pos_w)), pos_c, color='green', alpha=0.7)
ax1.set_yticks(range(len(pos_w)))
ax1.set_yticklabels(pos_w)
ax1.set_title('Top 15 Words in Positive Reviews')
ax1.set_xlabel('Frequency')
ax1.invert_yaxis()

neg_w, neg_c = zip(*neg_freq)
ax2.barh(range(len(neg_w)), neg_c, color='red', alpha=0.7)
ax2.set_yticks(range(len(neg_w)))
ax2.set_yticklabels(neg_w)
ax2.set_title('Top 15 Words in Negative Reviews')
ax2.set_xlabel('Frequency')
ax2.invert_yaxis()

plt.tight_layout()
plt.show()
```

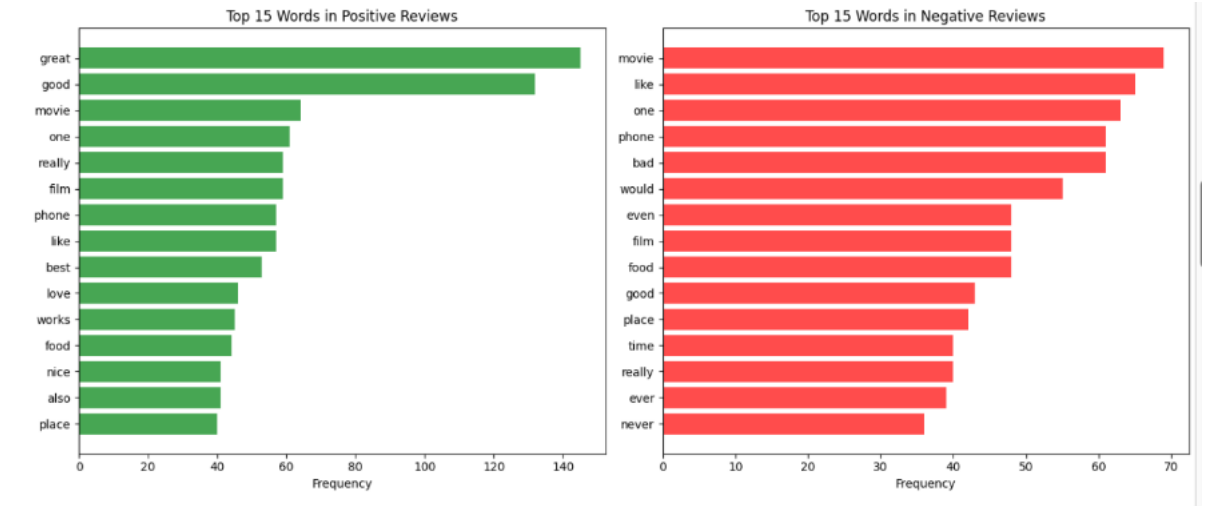

Figure 3. Subplots of top 15 words in positive (left and negative (right) reviews

3.3 Data Preprocessing

Tokenisation was performed using *RegexpTokenizer* to retain only alphanumeric characters and apostrophes. Next, stop words were removed, as they contain little semantic value. Stemming was then applied to reduce words to their root forms and group similar terms together as seen in the code below.

```
# Text preprocessing function (following guide approach)
def preprocess_text(text):
    """Simple text preprocessing using tokenization, stopword removal, and stemming
    # Tokenize - using RegexpTokenizer to keep only alphanumeric characters and apo
    tokenizer = RegexpTokenizer(r'[a-zA-Z0-9\']+')
    tokens = tokenizer.tokenize(text)

    # Remove stopwords and convert to lowercase
    stop_words = set(stopwords.words('english'))
    cleaned_tokens = [word.lower() for word in tokens if word.lower() not in stop_w

    # Apply stemming
    stemmer = PorterStemmer()
    stemmed_text = [stemmer.stem(word) for word in cleaned_tokens]

    return ' '.join(stemmed_text)

# Apply preprocessing
print("Applying text preprocessing...")
combined_df['text_cleaned'] = combined_df['text'].apply(preprocess_text)

# Show example of preprocessing
print("\nPreprocessing Examples:")
for i in range(3):
    print(f"\nOriginal: {combined_df.iloc[i]['text']}")
    print(f"Cleaned:  {combined_df.iloc[i]['text_cleaned']}")

print(f"\nPreprocessing completed! Dataset ready for modeling.")
```

```
Applying text preprocessing...

Preprocessing Examples:

Original: So there is no way for me to plug it in here in the US unless I go by a
converter.
Cleaned:  way plug us unless go convert

Original: Good case, Excellent value.
Cleaned:  good case excel valu

Original: Great for the jawbone.
Cleaned:  great jawbon
```

Figure 4. Example of original text (before preprocessing) and cleaned text (after preprocessing)

4.0 SENTIMENT ANALYSIS IMPLEMENTATION

In this section, the cleaned text and corresponding sentiment labels were prepared for modelling. The dataset was then split into balanced training and test sets to allow the model to be trained and evaluated effectively. This ensured that both sentiment classes were fairly represented during learning and performance assessment. The following code was used to carry out these preparation and data-splitting steps.

```python
# Prepare data for modeling
X = combined_df['text_cleaned']
y = combined_df['sentiment']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training set size: {len(X_train)}")
print(f"Test set size: {len(X_test)}")
print(f"Training set sentiment distribution: {y_train.value_counts().to_dict()}")
print(f"Test set sentiment distribution: {y_test.value_counts().to_dict()}")
```

Training set size	2198
Test set size	550
Training set sentiment distribution	{1: 1109, 0: 1089}
Test set sentiment distribution	{1: 277, 0: 273}

Table 3. Training and test set sentiment distribution

The cleaned text data was transformed into a numerical format that could be processed by the machine learning model, as the model cannot interpret raw text directly. This conversion process represents each word as a numerical feature, enabling effective analysis and classification. The resulting numerical representation of the dataset is presented in Table 4.

```
# Feature Engineering - Term Frequency Matrix using CountVectorizer
vectorizer = CountVectorizer()

X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

print(f"Feature matrix shape: {X_train_vectorized.shape}")
print(f"Vocabulary size: {len(vectorizer.vocabulary_)}")
```

Feature matrix shape	(2198, 3477)
Vocabulary size	3477

Table 4. Feature matrix and vocabulary size. There are 2198 reviews in the training set and 3477 unique words(features) found in those reviews. The model now sees the data as a table with 2198 rows and 3477 columns. 3477 words were identified by the CountVectorizer across all training reviews after preprocessing.

4.1 Naive Bayes Algorithm

The Multinomial Naive Bayes algorithm was used to analyse the dataset. This classification technique is based on the multinomial probability distribution and is well suited for text data with discrete features, such as word frequencies, making it appropriate for this task. Before model implementation, class imbalance was addressed using undersampling through the RandomUnderSampler method, which reduces the size of the majority class by randomly removing samples. The classifier was then trained on this balanced dataset to predict whether each review expressed positive or negative sentiment, as demonstrated in the code below.

```
# Handle class imbalance using undersampling
resampler = RandomUnderSampler(random_state=0)
X_train_resampled, y_train_resampled = resampler.fit_resample(X_train_vectorized, y

# Check resampled distribution
print("Resampled class distribution:")
print(pd.Series(y_train_resampled).value_counts())

# Train Naive Bayes model
print("\nTraining Multinomial Naive Bayes classifier...")
model = MultinomialNB()
model.fit(X_train_resampled, y_train_resampled)

print("Model training completed!")
```

Sentiment	Count
0-Negative	1089
1-Positive	1089

Table 5. Resampled class distribution

After undersampling, the training set contains 1,089 negative reviews (0) and 1,089 positive reviews (1), resulting in a perfectly balanced dataset. This balance ensures that the model is not biased toward the class that originally had more samples, allowing for a fairer and more reliable evaluation of its performance across both sentiment categories.

4.2 Model Evaluation

The key metrics used to evaluate the model are described in this section. Since this is a binary sentiment classification task, it is important not only to measure how many predictions are correct, but also to understand how the model makes errors. The selected evaluation metrics include accuracy, precision, recall, F1-score and the confusion matrix. These metrics were chosen because, they provide insight into overall correctness, error distribution, prediction reliability, completeness of detection and balanced performance across both classes. The code below was used to evaluate the model.

```
# Model Evaluation
y_pred = model.predict(X_test_vectorized)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2%}\n")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
print()

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))
```

	Negative	Positive
Model Accuracy	**80.36%**	
precision	0.80	0.80
Recall	0.80	0.81
F1-Score	0.80	0.81

Table 6. Model Evaluation: Model accuracy and classification report summary. Both classes perform almost equally, which confirms the model is not biased.

The confusion matric was generated using the code and the output displayed in Figure 4.

```
# Visualize Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix - Naive Bayes')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

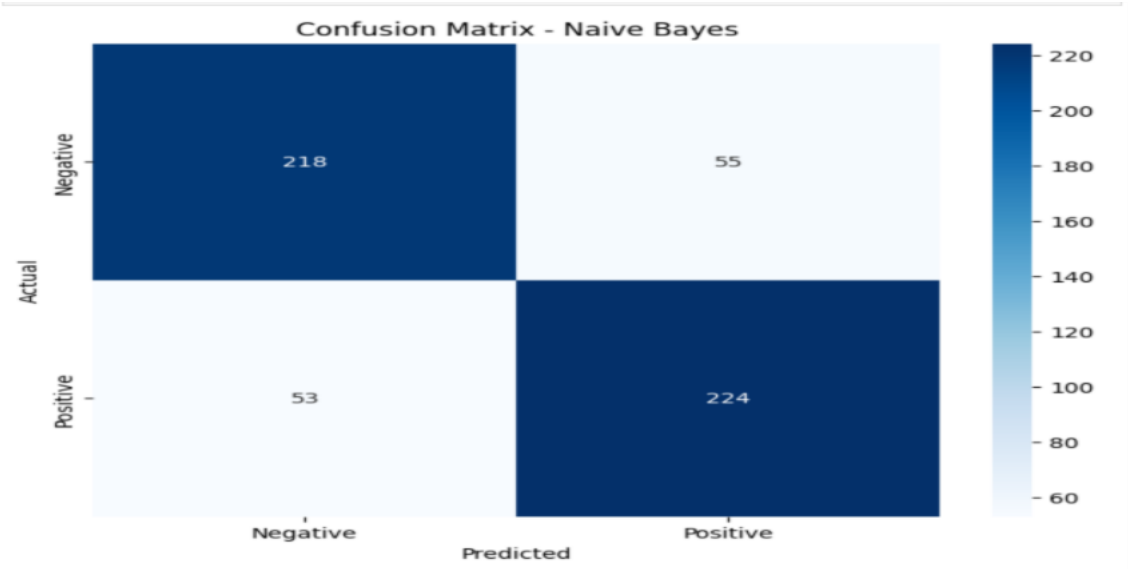

Figure 4. Confusion Matrix

5.0 RESULTS ANALYSIS AND DISCUSSION

The sentiment analysis model achieved an accuracy of 80%, which shows it performs reasonably well but is not highly advanced. This level of accuracy is typical for basic machine learning approaches and suggests the model works as a solid baseline rather than a high-performing system.

The precision, recall and F1-scores for both positive and negative classes are similar (approx. 80%). This means the model treats both classes fairly and does not strongly favour one over the other. However, it still misclassifies about 20% of the data, which indicates limitations in understanding more complex or subtle language, such as sarcasm or mixed opinions.

The confusion matrix shows an equal number of false positives and false negatives. This suggests the model is consistently uncertain in some cases, likely because it relies on surface-level word patterns instead of deeper meaning.

5.1 Business Benefits and Real-World Applications

The applied methods directly support businesses by transforming large volumes of customer feedback into actionable insights. By classifying reviews as positive or negative, the model could enable organisations to understand customer opinions at scale in a cost-effective manner. It improves decision-making through the identification of patterns in customer satisfaction and recurring issues. It also enhances customer experience by allowing early detection of negative reviews and enabling prompt responses to customer complaints. Finally, it improves resource allocation by reducing dependence on manual review processes, allowing staff to focus on higher-value strategic tasks.

5.2 Decision Making Framework – Business Interpretation

Insights from the model (positive and negative) can help organizations to pay attention to actual customer opinions. Decision making can be influenced through the following ways:

- Better Strategic Decisions: Helping managers understand how products or services are received.
- Faster Response: Quickly identifying and prioritizing negative feedback.
- Process Optimisation: Quicker review of customer feedback to speed up logistic process.
- Product Improvement: Helping product teams identify products that customers like.

CONCLUSION

This study examined whether sentiment analysis could classify customer reviews as positive or negative using a Naive Bayes model and whether this could provide useful insight for businesses. The results show that the model is suitable as a basic and practical approach. It achieved an accuracy of 80%, with fairly balanced results for both classes, suggesting it can reliably separate positive and negative reviews without strong bias. Exploratory analysis also showed clear differences in the language used by each group.

However, the model struggled with more complex language, such as sarcasm and mixed emotions, which limits its overall accuracy. This means it is not an advanced solution, but it remains effective for handling large volumes of feedback. Its efficiency makes it useful for real-world applications where quick analysis is needed.

Businesses can use these findings to better understand customer opinions and identify areas where satisfaction is low. Early detection of negative feedback allows faster responses, which can help reduce customer churn and protect brand image. Positive feedback can also inform marketing by highlighting features that customers value most. Overall, this system supports better decision-making, improves customer experience, and helps businesses allocate resources more effectively.

RECOMMENDATIONS

Below are some of the actionable recommendations that could be considered.

- Implement Real-Time Sentiment Monitoring: Integrate the model into customer feedback platforms so reviews are analysed automatically as they are submitted
- Prioritise Negative Feedback for Rapid Response: This allows quicker responses, improving customer satisfaction and reducing reputational damage
- Use sentiment Trends to Guide Strategy: Constantly review weekly or monthly sentiment reports to identify changes in customer opinion.
- Improve Weak Process Areas: Focus improvement efforts specifically negative review themes instead of applying broad, unfocused changes.

REFERENCES

Brown, A., & Green, M. (2022). Metadata-driven insights in sentiment analysis: A review. *Journal of Data & Information Quality*, 14(3), 1-17. https://doi.org/10.1016/j.jdiq.2022.100234

Kim, Y., & Park, J. (2024). Temporal dynamics of sentiment in social-media discourse around policy events. *Computers in Human Behavior*, 143, 107650. https://doi.org/10.1016/j.chb.2024.107650

Li, X., Zhou, Y., & Wang, L. (2024). Deep-learning advances in multilingual sentiment analysis: Challenges & opportunities. *Information Processing & Management*, 61(2), 103589. https://doi.org/10.1016/j.ipm.2024.103589

Smith, R., Jones, D., & Lee, S. (2023). Linking review content and metadata for actionable insights: A sentiment-analysis approach. *Expert Systems with Applications*, 201, 117486. https://doi.org/10.1016/j.eswa.2022.117486

Zhang, H., & Zhou, P. (2023). A survey of sentiment-analysis methodologies: From lexicons to deep learning. *Knowledge-Based Systems*, 275, 109811. https://doi.org/10.1016/j.knosys.2023.109811