Assessment Report

on

“Customer Behaviour”

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AI)

By

Name: Anwesha Singhal
Roll Number : 202401100300061
Section : A

Under the supervision of

“Bikki Gupta”

KIET Group of Institutions, Ghaziabad

May, 2025

1. Introduction

As digital lending platforms become more prevalent, automating credit risk assessment using data-driven methods is crucial. This project addresses the problem of predicting loan default using supervised machine learning. By utilizing a dataset containing borrower information such as credit scores, income, and loan history, the aim is to build a predictive model that helps financial institutions make informed lending decisions**.**

2. Problem Statement

To predict whether a borrower will default on a loan using available financial and credit history data.
The classification will help lenders mitigate risk by identifying high-risk applicants.

3. Objectives

- Preprocess the dataset for training a machine learning model.
- Train a Logistic Regression model to classify loan defaults.
- Evaluate model performance using standard classification metrics.
- Visualize the confusion matrix using a heatmap for interpretability.

4. Methodology

- Data Collection: The user uploads a CSV file containing the dataset.
- Data Preprocessing:
    - Handling missing values using mean and mode imputation.
    - One-hot encoding of categorical variables.
    - Feature scaling using StandardScaler.
- Model Building:
    - Splitting the dataset into training and testing sets.

    - Training a Logistic Regression classifier.

- Model Evaluation:
    - Evaluating accuracy, precision, recall, and F1-score.
    - Generating a confusion matrix and visualizing it with a heatmap.

---

5. Data Preprocessing

The dataset is cleaned and prepared as follows:

- Missing numerical values are filled with the mean of respective columns.
- Categorical values are encoded using one-hot encoding.
- Data is scaled using StandardScaler to normalize feature values.
- The dataset is split into 80% training and 20% testing.

---

6. Model Implementation

Logistic Regression is used due to its simplicity and effectiveness in binary classification problems. The model is trained on the processed dataset and used to predict the loan default status on the test set.

---

7. Evaluation Metrics

The following metrics are used to evaluate the model:

- Accuracy: Measures overall correctness.
- Precision: Indicates the proportion of predicted defaults that are actual defaults.
- Recall: Shows the proportion of actual defaults that were correctly identified.
- F1 Score: Harmonic mean of precision and recall.

- Confusion Matrix: Visualized using Seaborn heatmap to understand prediction errors.

---

8. Results and Analysis

- The model provided reasonable performance on the test set.
- Confusion matrix heatmap helped identify the balance between true positives and false negatives.
- Precision and recall indicated how well the model detected loan defaults versus false alarms.

---

9. Conclusion

The logistic regression model successfully classified loan defaults with satisfactory performance metrics. The project demonstrates the potential of using machine learning for automating loan approval processes and improving risk assessment. However, improvements can be made by exploring more advanced models and handling imbalanced data.

---

10. References

- scikit-learn documentation
- pandas documentation
- Seaborn visualization library

Research articles on credit risk prediction

```
[29] import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score, confusion_matrix
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.metrics import confusion_matrix
```

```
print("Available columns in your dataset:")
print(df.columns.tolist())
```

```
Available columns in your dataset:
['total_spent', 'avg_purchase_value', 'visits_per_month', 'buyer_type']
```

```
target_column = input("Enter the name of the column that indicates returns (e.g., 'return', 'status'): ")
while target_column not in df.columns:
    print(f"Column '{target_column}' not found in dataset.")
    target_column = input("Please enter a valid column name: ")
```

```
Enter the name of the column that indicates returns (e.g., 'return', 'status'): return
Column 'return' not found in dataset.
Please enter a valid column name: total_spent
```

```
[11] print("\nAvailable features:")
     print([col for col in df.columns if col != target_column])
     feature_choices = input("Enter column names to use as features (comma separated): ").split(',')
     features = [f.strip() for f in feature_choices if f.strip() in df.columns]
```

```

Available features:
['avg_purchase_value', 'visits_per_month', 'buyer_type']
Enter column names to use as features (comma separated): visits_per_month
```

```
[12] if not features:
         features = [col for col in df.columns if col != target_column][:3]  # Use first 3 non-target columns as default
         print(f"\nUsing default features: {features}")
```

```
[13] print(f"\nUnique values in '{target_column}':")
     print(df[target_column].value_counts())
```

```
Unique values in 'total_spent':
total_spent
4007.982067    1
3117.968387    1
4232.062646    1
577.820196     1
2839.005107    1
              ..
2676.205476    1
1322.669398    1
109.890737     1
953.000209     1
673.623603     1
Name: count, Length: 100, dtype: int64
```

```
if df[target_column].nunique() == 2:  # If exactly 2 unique values
    df['target'] = (df[target_column] == df[target_column].value_counts().idxmax()[0]).astype(int)
else:
    # Let user specify which value means "returned"
    returned_value = input(f"Which value in '{target_column}' indicates a return? ")
    df['target'] = (df[target_column] == returned_value).astype(int)
```

```
Which value in 'total_spent' indicates a return? 2
```

```
[15] df_clean = df[features + ['target']].dropna()
```

```
[16] X = df_clean[features]
     y = df_clean['target']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
[17] model = RandomForestClassifier(n_estimators=100, random_state=42)
     model.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
[18] predictions = model.predict(X_test)

     # Calculate accuracy
     accuracy = accuracy_score(y_test, predictions)
     print(f"\nModel Accuracy: {accuracy:.2%}")
```

```
Model Accuracy: 100.00%
```

```
[30] # Example of a confusion matrix
     y_true = [0, 1, 0, 1, 1, 0, 1]  # Actual values
     y_pred = [0, 0, 0, 1, 1, 0, 1]  # Predicted values
```

```
[31] # Calculate confusion matrix
     cm = confusion_matrix(y_true, y_pred)
```

```
     # Plot heatmap
     plt.figure(figsize=(6, 5))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Bargain Hunter', 'Premium Buyer'], yticklabels=['Bargain Hunter', 'Premium Buyer'])
```

```
     # Plot heatmap
     plt.figure(figsize=(6, 5))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Bargain Hunter', 'Premium Buyer'], yticklabels=['Bargain Hunter', 'Premium Buyer'])
     plt.title("Confusion Matrix Heatmap")
     plt.xlabel("Predicted")
     plt.ylabel("Actual")
     plt.show()
```