SESSION 2024/2025 (A242)

COLLEGE OF ARTS AND SCIENCES

SCHOOL OF COMPUTING

STTHK2113 DATA ANALYTICS (B)

GROUP ASSIGNMENT 3: PREDICTIVE MODELLING TASK

SUBMITTED TO:

TS. DR. MOHAMED ALI B. SAIP

PREPARED BY:

NAME	MATRIC NO.
SITI AIN ATHIQAH BINTI SAHRUN	297545
HANA SYAKIRAH BINTI HASSAN KHAIRULLAH	299403
ANISA NADIAH BINTI ALWI	299892
NUR FAIZLYANA BINTI MOHD KAMARUL ARIFFIN	300442
NUR ALIAH NADHIRA BINTI KYAIRUL NIZAM	300595

SUBMITTED ON: 30/06/2025

Table of Contents

1.0 Executive Summary

1.1 Problem Statement

The primary objective of this project is to develop a predictive model capable of determining whether a patient is eligible to donate blood or not. This is framed as a binary classification problem, where the model classifies individuals into 'Blood Donor' or 'Non-Blood Donor' categories based on their demographic and biochemical attributes.

1.2 Chosen Methodology

The methodology adopted for this predictive modeling task involved several key stages:

1. **Dataset Selection**: The Hepatitis C Virus (HCV) dataset from Kaggle was chosen, containing 615 records and 14 attributes, suitable for binary classification.
2. **Data Preprocessing**:
    - **Data Cleaning**: Missing values were imputed using the median, outliers were handled by a clipping method, restricting values within the 1st and 99th percentiles, and highly skewed numerical distributions underwent a log transformation (log1p).
    - **Data Transformation**: Categorical variables, specifically 'Sex' and 'Category', were encoded. The 'Category' column was transformed into a binary 'Target' variable (0 for blood donors, 1 for non-blood donors). Numerical features were standardized using **RobustScaler** to ensure consistent scaling and robustness to outliers.
3. **Model Development**: Logistic Regression was selected as the predictive model due to its suitability for binary classification tasks.
4. **Model Evaluation**: The model was trained and evaluated across various train-test split ratios (10:90 to 90:10). Performance was assessed using a suite of metrics including Accuracy, Precision, Recall, F1-Score, ROC-AUC, Confusion Matrix, Sensitivity, and Specificity.

1.3 Key Findings

The Logistic Regression model demonstrated strong performance in identifying eligible blood donors. A key finding is the model's consistent perfect precision, indicating no false positives, meaning that no non-blood donors were incorrectly classified as blood donors. The **80:20 train-test split** emerged as the best-performing configuration, achieving the highest accuracy, perfect ROC-AUC, and the highest recall and sensitivity among the evaluated splits, while maintaining perfect precision and specificity. This configuration provides the optimal balance for safety and effective screening in blood donation eligibility.

2.0 Dataset Description

2.1 Dataset Source

Link of dataset: https://www.kaggle.com/datasets/ardikurniawan/hcvdat0

2.2 Number of Features and Records

This dataset contains 615 records (rows) and 14 attributes (columns). These include an index column, one target variable (category) and multiple input features such as demographic and biochemical attributes. The target variable (category) includes five classes: Blood Donor, suspected Blood Donor, Hepatitis, Fibrosis and Cirrhosis.

The table below summarizes the type and role for each column:

No.	Column	Role	Value Type
1.	index	Feature/Skip	Numerical (int)
2.	category	Target	Categorical
3.	age	Feature	Numerical (int)
4.	sex	Feature	Categorical
5.	ALB (Albumin)	Feature	Numerical (float)
6.	ALP (Alkaline Phosphatase)	Feature	Numerical (float)
7.	ALT (Alanine Aminotransferase)	Feature	Numerical (float)
8.	AST (Aspartate Aminotransferase)	Feature	Numerical (float)

9.	BIL (Bilirubin)	Feature	Numerical (float)
10.	CHE (Cholinesterase)	Feature	Numerical (float)
11.	CHOL (Cholesterol)	Feature	Numerical (float)
12.	CREA (Creatinine)	Feature	Numerical (float)
13.	GGT (Gamma-Glutamyl Transferase)	Feature	Numerical (float)
14.	PROT (Total Protein)	Feature	Numerical (float)

Table 1: Number of features and records

3.0 Preprocessing Techniques

3.1 Data Cleaning

Firstly, we conduct the data cleaning to address the missing values, outliers, and skewed distributions.

Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load the dataset
df = pd.read_csv("hcvdata.csv", encoding="latin-1")

# 2. Display missing values
print("Missing data count:\n", df.isnull().sum())

# 3. Select numeric columns
numeric_cols = df.select_dtypes(include='number').columns

# 4. Outlier Detection using Z-score
z_scores = np.abs((df[numeric_cols] - df[numeric_cols].mean()) /
df[numeric_cols].std())
outlier_mask = z_scores > 3
outlier_counts = outlier_mask.sum()
print("\nNumber of outliers in each column (Z-score > 3):\n", outlier_counts)

# 5. Optional: View rows with outliers
#rows_with_outliers = df[outlier_mask.any(axis=1)]
#print("\nRows with any outliers:\n", rows_with_outliers)

# 6. Visualize outliers using boxplots
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[numeric_cols])
plt.xticks(rotation=45)
plt.title("Boxplot of Numeric Columns")
plt.tight_layout()
```

```
plt.show()

# 7. Skewness Analysis
skewness = df[numeric_cols].skew().sort_values(ascending=False)

# Categorize skewness
def categorize_skew(value):
    if value > 1 or value < -1:
        return "Highly Skewed"
    elif 0.5 < value <= 1 or -1 <= value < -0.5:
        return "Moderately Skewed"
    else:
        return "Approximately Symmetric"

skew_df = pd.DataFrame({
    'Feature': skewness.index,
    'Skewness': skewness.values,
    'Category': skewness.apply(categorize_skew).values
})

print("\nSkewness of each numeric feature:\n", skew_df)

# 8. Count features by skewness category
print("\nNumber of features in each skewness category:\n",
skew_df['Category'].value_counts())
```

Output:

```
Missing data count:
 Unnamed: 0     0
Category        0
Age             0
Sex             0
ALB             1
ALP            18
ALT             1
AST             0
BIL             0
CHE             0
CHOL           10
CREA            0
GGT             0
PROT            1
dtype: int64

Number of outliers in each column (Z-score > 3):
 Unnamed: 0     0
Age             0
ALB            13
ALP             3
ALT            10
AST            14
BIL             7
CHE             9
CHOL            7
CREA            3
GGT            10
PROT            9
dtype: int64
```

```
Skewness of each numeric feature:
        Feature   Skewness                Category
0          CREA  15.169291          Highly Skewed
1           BIL   8.385437          Highly Skewed
2           GGT   5.632734          Highly Skewed
3           ALT   5.506114          Highly Skewed
4           AST   4.940327          Highly Skewed
5           ALP   4.654921          Highly Skewed
6          CHOL   0.375828  Approximately Symmetric
7           Age   0.267134  Approximately Symmetric
8    Unnamed: 0   0.000000  Approximately Symmetric
9           CHE  -0.110233  Approximately Symmetric
10          ALB  -0.176768  Approximately Symmetric
11         PROT  -0.963687        Moderately Skewed

Number of features in each skewness category:
 Category
Highly Skewed             6
Approximately Symmetric   5
Moderately Skewed         1
Name: count, dtype: int64
```

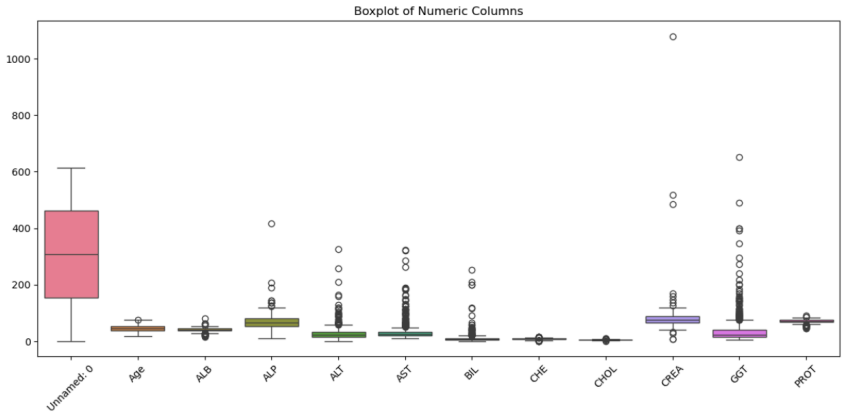

Figure 1: Missing data count, number of outliers, skewed distribution and boxplot

Based on the results, instead of dropping the row of each issue, we can handle it. This approach was taken because the dataset contains 615 rows, and removing records could affect the result or loss of valuable information.

The missing values in the dataset occur in the following columns: ALB = 1, ALP = 18, ALT = 1, CHOL = 10, and PROT = 1. Since all missing values are in numerical value, they can be handled by imputing with the median to minimize the impact of outliers. Meanwhile for the outliers, it can be handled by using a clipping method, which restricts values within the 1st and 99th percentiles to reduce the influence of extreme values.

Additionally, the skewed distributions were addressed by applying a log transformation (log1p) to features with high skewness (skewness >1). This transformation helped normalize their distributions and potentially improve model performance. Specifically, six features were identified as highly skewed and were transformed accordingly. The remaining features, which are symmetric and only moderated skewed distribution were left untransformed to preserve the original clinical interpretability and avoid unnecessary changes that could affect their real world meaning.

The code below shows how the issues contained from the dataset are handled:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load Dataset
df = pd.read_csv("hcvdata.csv", encoding="latin-1")

# Handle Missing Values (median)
cols_with_missing = ['ALB', 'ALP', 'ALT', 'CHOL', 'PROT']
for col in cols_with_missing:
    median_value = df[col].median()
    df[col] = df[col].fillna(median_value)

# Handle Outliers (Clipping)
numerical_cols = ['ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA',
'GGT', 'PROT']
for col in numerical_cols:
    lower, upper = df[col].quantile([0.01, 0.99])
    df[col] = np.clip(df[col], lower, upper)

# Apply Log Transformation to Highly Skewed Columns
highly_skewed_cols = ['CREA', 'BIL', 'GGT', 'ALT', 'AST', 'ALP']
for col in highly_skewed_cols:
    if (df[col] >= 0).all():  # Ensure no negative values
        df[col] = np.log1p(df[col])  # Safe log transform: log(1 + x)
    else:
        print(f"Skipped {col} due to negative values.")

# Confirm no missing values remain
print("\nMissing values after imputation:\n", df.isnull().sum())
```

```
# Recheck Skewness
print("\nSkewness of numerical columns after transformation:")
print(df[numerical_cols].skew())

# Plot histograms to visualize final distributions
plt.figure(figsize=(18, 10))
for i, col in enumerate(numerical_cols):
    plt.subplot(3, 4, i+1)
    sns.histplot(df[col], kde=True)
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()

# Save cleaned dataset
df.to_excel("Cleaned_Dataset.xlsx", index=False)
```

Output:

```
Missing values after imputation:
 Unnamed: 0    0
Category       0
Age            0
Sex            0
ALB            0
ALP            0
ALT            0
AST            0
BIL            0
CHE            0
CHOL           0
CREA           0
GGT            0
PROT           0
dtype: int64

Skewness of numerical columns after transformation:
ALB    -0.817669
ALP    -0.317343
ALT    -0.166088
AST     1.815691
BIL     1.101718
CHE    -0.274983
CHOL    0.359162
CREA    0.042370
GGT     0.931163
PROT   -0.738850
dtype: float64
```

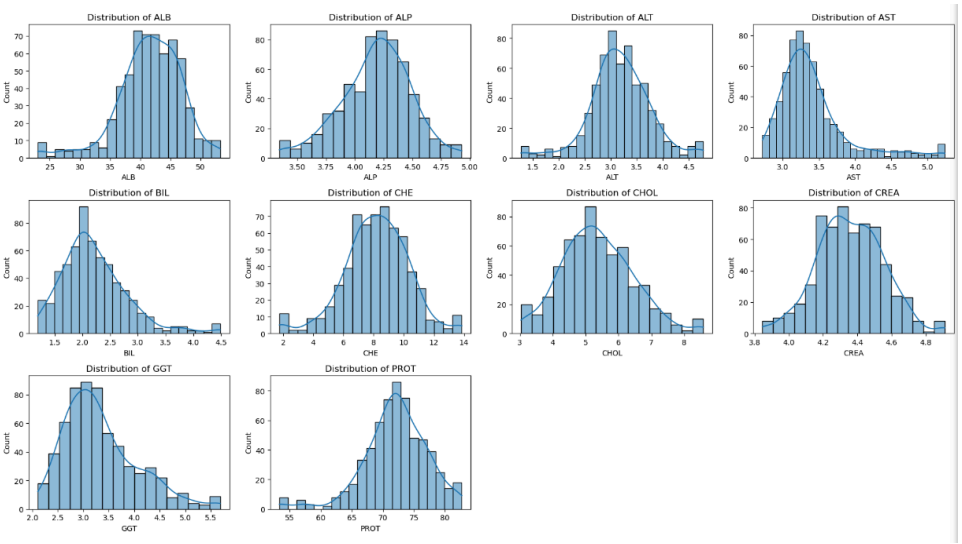

Figure 2: Missing value after imputation, skewness after transformation, and graph distribution

After applying imputation techniques to the dataset, the missing values confirmed that all columns now contain zero missing value. This ensures that the dataset is complete and ready for modeling. Besides that, the six features with high skewness have reduced skewness which is improving the effectiveness of our model choice, which is logistic regression. However, two features which remained highly skewed: AST and BIL. These two features are intentionally left untransformed to preserve original scale and clinical interpretability important in medical contexts. Instead of further transformation, we relied on RobustScaler for standardization, which is effective at handling outliers and skewed data.

3.2 Data Transformation

After performing data cleaning, appropriate preprocessing techniques were applied to prepare the dataset for model development. Since the goal is to predict the blood donors and non-blood donors, which is a binary classification problem, standardization was applied to the numerical columns to ensure consistent scaling and improve model performance.

In this case, the standardization that we use is the RobustScaler method. The reason for choosing the RobustScaler method is that it is more resistant to outliers, which are common in medical data. Besides that, RobustScaler makes it more suitable for datasets with skewed distributions or abnormal values.

Besides that, to prepare the data for classification, categorical attributes were encoded. The sex column was encoded as f (female) = 0, and m (male) = 1. For the category column, which is the target attribute, the data was mapped into binary classification format, where 0 represents the blood donor class and 1 represents the non-blood donor class. We encoded the ‘0=Blood Donor’ under blood donor class (0) while all other categories such as ‘0s=suspect Blood Donor’, ‘1=Hepatitis’, ‘2=Fibrosis’, and ‘3=Cirrhosis’ were encoded under non-blood donor (1). The reason why other categories are classified as the non-blood donor class is because the medical safety guidelines restrict individuals with liver abnormalities or infectious diseases like hepatitis, fibrosis, cirrhosis from donating blood. Meanwhile, suspected donors are also treated as non-blood donors to ensure safety, even if they are not yet diagnosed.

The code below shows how the cleaned dataset was handled using appropriate preprocessing techniques:

```
import pandas as pd
from sklearn.preprocessing import RobustScaler, LabelEncoder

# Load the cleaned dataset
df = pd.read_excel("Cleaned_Dataset.xlsx")

# Step 1: Encode the 'Sex' column (categorical)
if 'Sex' in df.columns:
    le_sex = LabelEncoder()
    df['Sex'] = le_sex.fit_transform(df['Sex'])  # 'm' → 1, 'f' → 0

# Step 2: Encode the 'Category' column to create a binary 'Target'
df['Target'] = df['Category'].map({
    '0=Blood Donor': 0, #blood donor (0)
    '0s=suspect Blood Donor': 1, #non-blood donor (1)
    '1=Hepatitis': 1,
    '2=Fibrosis': 1,
    '3=Cirrhosis': 1
}).astype(int)

# Step 3: Drop unnecessary columns (optional)
df.drop(columns=['Unnamed: 0', 'Category'], inplace=True)

# Step 4: Define numerical columns for standardization
numerical_cols = ['Age', 'ALB', 'ALP', 'ALT', 'AST', 'BIL',
                  'CHE', 'CHOL', 'CREA', 'GGT', 'PROT']

# Step 5: Standardize numerical features using RobustScaler
scaler = RobustScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Preview the result
print(df.head())

# Step 6: (Optional) Save the fully preprocessed dataset
df.to_excel("Preprocessed_Dataset.xlsx", index=False)

# Confirmation message
print("Preprocessed dataset saved as 'Preprocessed_Dataset.xlsx'")
```

Output:

```
   Age  Sex       ALB       ALP       ALT       AST       BIL       CHE  \
0 -1.0    1 -0.539062 -0.573258 -1.511458 -0.375602  0.036028 -0.500942
1 -1.0    1 -0.539062  0.148909 -0.347973 -0.112550 -0.797445  1.096045
2 -1.0    1  0.773437  0.299475  0.652788  1.700326 -0.236290  0.218456
3 -1.0    1  0.195312 -0.596867  0.409771 -0.322789  1.323169 -0.350282
4 -1.0    1 -0.429687  0.279467  0.501181 -0.102973  0.370107  0.335217

       CHOL      CREA       GGT      PROT  Target
0 -1.442509  1.174605 -0.684213 -0.524590       0
1 -0.348432 -0.145732 -0.421995  0.704918       0
2 -0.069686  0.405751  0.378448  1.163934       0
3 -0.390244  0.140232  0.397707  0.573770       0
4 -0.682927 -0.047945  0.266082 -0.573770       0
Preprocessed dataset saved as 'Preprocessed_Dataset.xlsx'
```

Figure 3: Data transformation (preprocessing)

After applying the appropriate preprocessing techniques, including standardization of numerical features and encoding the categorical variables from the cleaned dataset. The updated dataset is now fully prepared for the subsequent modelling process. The updated preprocessing dataset has been successfully saved in Excel format under the name Preprocessed_Dataset.

4.0 Modelling Process

4.1 Model Selected

The model selected to perform predictive analytics is Logistic Regression.
Based on our dataset, our final target is to determine whether a patient can donate blood or not, which falls under binary classification. The outcome variable 'Target' column was derived by encoding the original 'Category' column, where the values were set to 0, which is for blood donors, and 1 for non-blood donors. Logistic Regression suits this type of problem as it is designed to handle binary outcomes effectively.

4.2 Model Development

From the preprocessed dataset, the Logistic Regression model were trained and evaluated using various train-test ratios which were 10:90, 20:80, 30:70, 40:60, 50:50, 60:40, 70:30, 80:20, and 90:10 respectively. These splits were implemented to observe and evaluate the model's performance under different amounts of training data. The training set, in this context, is used to fit the model and allows it to learn patterns from the data, while the test set is used for predictions and evaluate how well the model performs on unseen data (data that the model has never encountered during training), and check how well it performs on real-world scenarios. This approach helps to assess the model's stability and its generalization under realistic scenarios where data availability may vary, which is common in medical datasets.

The target variable for this model is 'Target' column, which represents the blood donation eligibility category (0 for donors, 1 for non-blood donors), while the features selected to determine the classification were 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', and 'PROT'. These features were selected as they were relevant to liver function and blood chemistry, where it is biologically important to determine whether they are eligible to donate blood or otherwise.

Logistic Regression Performance Summary:

	Train:Test Ratio	Accuracy	Precision	Recall	F1 Score	ROC AUC
0	90:10	1.000000	1.000000	1.000000	1.000000	1.000000
1	80:20	0.991870	1.000000	0.937500	0.967742	1.000000
2	70:30	0.978378	1.000000	0.840000	0.913043	0.996500
3	60:40	0.975610	1.000000	0.818182	0.900000	0.996301
4	50:50	0.967532	0.969697	0.780488	0.864865	0.987942
5	40:60	0.953930	0.900000	0.734694	0.808989	0.964286
6	30:70	0.955916	0.931818	0.719298	0.811881	0.973637
7	20:80	0.945122	0.914894	0.651515	0.761062	0.967172
8	10:90	0.945848	0.958333	0.621622	0.754098	0.959600

Figure 4: Logistic Regression Performance Summary

Figure 5: Model Performance vs. Train-Test Split Graph

The figures above show the summarization and graph of model performance of each train-test ratio with these evaluation matrices which are:

Accuracy: Proportion of correct predictions made by model,

Precision: Proportion of correctly identified non-donors (positive class) among all cases predicted as non-donors. High precision means the model makes less false positive errors,

Recall (Sensitivity): Ability of the model to correctly identify actual non-donors,

F1-Score: Mean of precision and recall, which provides balanced measures when both false negatives and false negatives are considered equally important, and

ROC-AUC (Receiver Operating Characteristic- Area Under Curve): Quantifies the model's ability to distinguish between donors and non-donors across various thresholds. A score closer to 1.0 means excellent class separability.

The summary and graph above also shows that the Accuracy, Precision, Recall, F1 Score and ROC-AUC are the highest at 1.000000 for the train-test split ratio at 90:10. Despite these scores, this split is not chosen as a top configuration due to small test size, hence there will be risks of the model's performance being overestimated due to the overfitting on the training data, why may lead to potential misclassification (e.g. identifying an ineligible donor as eligible).

On the other hand, the Accuracy, F1 Score are the lowest at train-test split 10:90, while Precision and ROC-AUC are the lowest at train-test ratio 40:60. The train-test splits of 80:20, 70:30, and 60:40 are chosen instead as they are more balanced and realistic by offering significant data for training and testing. These ratios still have strong performance metrics while reducing the possibility of evaluation bias.

The code below shows how the logistic regression model was developed (showing the target chosen and its features), alongside with its performance summary and graph:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score
)

# Load the preprocessed dataset
df = pd.read_excel("Preprocessed_Dataset.xlsx")

# Initialize lists to store metrics
train_test_labels = []
accuracies = []
precisions = []
recalls = []
f1_scores = []
roc_aucs = []

# Define features and target
selected_features = ['Age','Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL','CHE', 'CHOL',
'CREA', 'GGT', 'PROT']
X = df[selected_features]
y = df['Target']

# Define train-test split ratios
split_ratios = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

# Evaluate model across different splits
for test_ratio in split_ratios:
    train_ratio = 1 - test_ratio
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_ratio, random_state=42, stratify=y
    )

    # Train logistic regression model
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)

    # Predict labels and probabilities
```

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

    # Store results
    train_test_labels.append(f"{round(train_ratio*100)}:{round(test_ratio*100)}")
    accuracies.append(accuracy_score(y_test, y_pred))
    precisions.append(precision_score(y_test, y_pred))
    recalls.append(recall_score(y_test, y_pred))
    f1_scores.append(f1_score(y_test, y_pred))
    roc_aucs.append(roc_auc_score(y_test, y_proba))

# Combine results into a single DataFrame
results_df = pd.DataFrame({
    'Train:Test Ratio': train_test_labels,
    'Accuracy': accuracies,
    'Precision': precisions,
    'Recall': recalls,
    'F1 Score': f1_scores,
    'ROC AUC': roc_aucs
})

# Display the final evaluation summary
print("\nLogistic Regression Performance Summary:")
print(results_df)
```

```
import matplotlib.pyplot as plt
import pandas as pd

# Assuming `results_df` already exists with metric columns
# If not, load it from Excel or generate it before this

# Plot metrics vs Train:Test Ratio
plt.figure(figsize=(12, 6))
for metric in ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']:
    plt.plot(results_df['Train:Test Ratio'], results_df[metric], marker='o',
label=metric)

plt.title('Model Performance vs Train-Test Split')
plt.xlabel('Train:Test Ratio')
plt.ylabel('Score')
plt.ylim(0.5, 1.05)
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

5.0 Performance Evaluation

5.1 Analysis of Top Three Ratios

Train:Test Ratio	Accuracy	Precision	Recall	F1 Score	ROC-AUC
80:20	0.991870	1.000000	0.937500	0.967742	1.000000
70:30	0.978378	1.000000	0.840000	0.913043	0.996500
60:40	0.975610	1.000000	0.818182	0.900000	0.996301

Table 2: Analysis of top three ratios

We selected the top 3 train test splits which are 80:20, 70:30 and 60:40, using the five key evaluation metrics mentioned previously. These metrics were chosen to assess how well the model can classify both blood donor and non-blood donor.

In terms of Precision, all three splits achieved the same perfect value of 1.000000. This means that for each split, whenever predicting a person as non-blood donor, it will always result as correct. There were no false positives in any of these three cases. This is highly important in a healthcare-related classification because wrong predictions can have serious consequences.

In the 80:20 split, the model recorded the highest overall Accuracy of 0.991870. It also had the highest Recall value which is 0.937500. This indicates that the model correctly identified 93.75% of actual non-blood donors. The F1 Score was 0.967742 showing a strong balance between Precision and Recall, and the ROC-AUC was 1.000000. This perfect score reflects excellent classification ability.

The 70:30 split also performed well with Accuracy of 0.978378, however the Recall dropped slightly to 0.840000. This means that the model missed more non-blood donor cases compared to 80:20 split. Its F1 Score was 0.913043 and the ROC-AUC was 0.996500, which is still extremely high and indicates good model discrimination between classes.

In the 60:40 split, the model achieved an accuracy of 0.975610 but the Recall further dropped to 0.818182. This indicates that it missed some much more actual non-blood donors compared to the 70:30 split. The F1 Score decreased to 0.900000 and the ROC-AUC remained high at 0.996301 showing that the model differed well between blood donors and non-blood donors. This suggests that the model is very precise but less sensitive. This means that it accurately identifies donors but misclassifies some non-donors as donors.

To summarize, we selected these top three splits because they all delivered strong and consistent results across all metrics. As a note, the Precision remained perfect at value 1.000000 across all three ratios. This means that the model made no incorrect non-blood donor predictions. Even so, Recall and F1 Score declined slightly from 80:20 ratios to 60:40 but it still remained within acceptable and effective ranges. The 80:20 split is considered the best overall as it achieved the highest values in Accuracy, Recall, F1 Score and ROC-AUC. It makes it the most reliable configuration for this classification task.

5.2 Top Three Ratios: Confusion Matrix

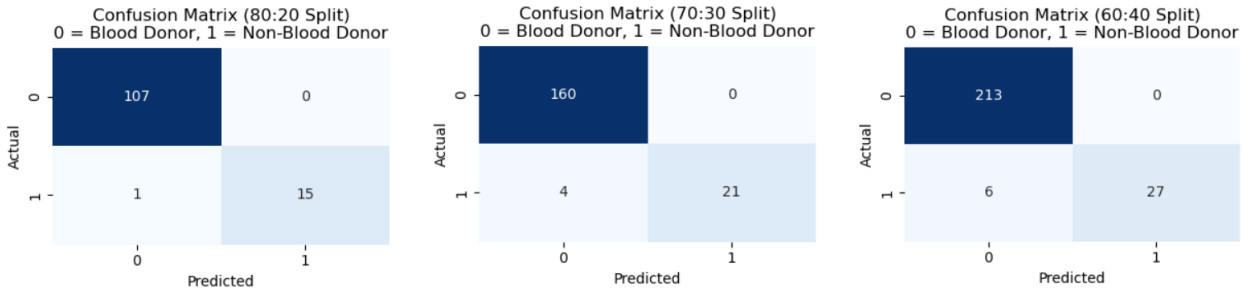

Figure 6: Confusion matrix with train-test split ratios of 80:20, 70:30, 60:40

In the comparison of these three confusion matrices for the 80:20,70:30 and 60:40 train-test split, we observed consistent trends that align with the performance metrics.

For the 80:20 split, the confusion matrix showed a high number of true negatives and true positives with no false positives and only a small number of false negatives. This means that the model accurately predicted both blood donors and non-blood donors. The few false negatives indicate that almost all non-blood donors were correctly identified. The 80:20 split had the highest recall and the perfect precisions. It makes the most balanced and reliable configuration.

In the 70:30 split, the confusion matrix still showed a strong number of predictions with no false positives and a slight high number of false negatives compared to the 80:20 split. While the model continued to avoid misclassifying donors as non blood donors, it actually missed more actual non-blood donors. As a result, the recall was slightly lower and yet the precision remained perfect. This indicates that the model was still highly accurate, but its ability to identify all non-donors had slightly decreased.

For the 60:40 split, the confusion matrix reflected the largest number of false negatives among all three splits and the false positives were still zero. Although the precision stayed at 1.00000, the recall dropped far further and this led to a slightly lower F1 Score. Among these 3 ratios, this split was the least effective in identifying non-blood donors.

In conclusion, all three splits with the perfect precisions which is 1.00000, it indicated that the model consistently avoided false negatives. However as the test size increased from 20% to 40% more false negatives appeared. This caused a decline in recall. The 80:20 split had the best balance of precision and recall. The confusion matrix reflected the most reliable and accurate performance in predicting both blood and non-blood donors.

5.3 Top Three Ratios: Sensitivity and Specificity

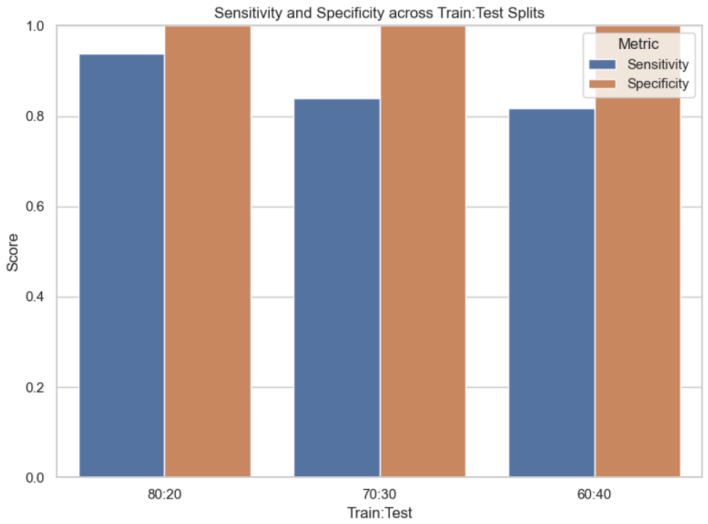

Figure 7: Sensitivity and Specificity based on ratios 80:20, 70:30, 60:40

In all three train-test splits, the model achieved a specificity of 1.0. Specificity measures the model's ability to correctly identify negative cases such as the blood donors. A specificity means that it consistently and correctly identified all blood donors. It also indicates that there were no false positives, so there are no blood donors wrongly predicted as non-blood donors in any of the splits.

The model's sensitivity, also known as recall for the positive class, refers to its ability to correctly identify positive cases such as, the non-blood donors. High sensitivity means the model accurately detects most non-blood donors while minimizing false negatives.

However, the sensitivity, also known as recall, decreased as the test size increased. In the 80:20 split, the model achieved a high sensitivity of 0.937500 which indicates that it correctly identified 93.75% of non-blood donors. For the sensitivity in the

70:30 split, it dropped to 0.840000, meaning that the model correctly identified 84% of non-blood donors. Meanwhile in the 60:40 split, sensitivity decreased to 0.818, which shows that only about 81.8& of non-blood donors were correctly detected.

This pattern suggests that the model remains accurate in identifying blood donors while becoming slightly less effective at detecting non-blood donors as the proportion of test data increases. This trade-off highlights that the 80:20 split provides the best balance with the highest sensitivity and specificity.

6.0 Best Model Selection

The most effective model configuration, based on a comprehensive evaluation of various metrics, is achieved with the **80:20 Split**. This configuration consistently outperformed other splits across key performance indicators.

This is because the 80:20 split yielded the **highest overall accuracy**, indicating that the model made correct predictions for approximately 99.2% of the test cases. This signifies excellent generalization capability on unseen data.

A crucial aspect in medical applications like blood donation, the model consistently **achieved a precision of 1.0**. This means that every individual predicted as a 'Blood Donor' was genuinely a blood donor. There were no false positives, which is paramount for ensuring the safety and integrity of the blood supply by preventing ineligible individuals from donating.

For the positive class, non-blood donors, the 80:20 split achieved the **highest recall and sensitivity**. This demonstrates the model's strong ability to correctly identify nearly 94% of actual non-donors. This is vital for minimizing false negatives, thereby reducing potential risks associated with ineligible donations.

A **perfect ROC-AUC score** indicates that the model has an exceptional ability to distinguish between the two classes, blood donors and non-blood donors. It effectively separates true positives from false positives across all classification thresholds.

The model maintained **perfect specificity** across all top splits, including 80:20. This means that 100% of actual blood donors were correctly identified as such. No eligible donors were wrongly rejected, which is important for maintaining the donor pool and avoiding unnecessary rejections.

While other splits also showed strong performance, particularly with perfect precision and specificity, their recall and F1-score values were incrementally lower as the test set size increased. The 80:20 split provided the optimal balance, leveraging a larger training set to build a robust model that most effectively identifies non-donors while ensuring safety and accurate identification of true donors.

7.0 Conclusion And Recommendations

The project successfully developed a Logistic Regression model for predicting blood donation eligibility based on the Hepatitis C Virus (HCV) dataset. Through meticulous data preprocessing, including median imputation for missing values, outlier clipping, and log transformation for skewed distributions, the dataset was prepared for robust modeling. Categorical features like 'Sex' and 'Category' were effectively encoded, transforming the 'Category' into a binary 'Target' variable representing blood donors and non-blood donors. The strategic use of RobustScaler for numerical feature standardization ensured the model's resilience to outliers prevalent in medical data. The 80:20 train-test split emerged as the optimal configuration, demonstrating exceptional accuracy, perfect ROC-AUC, and perfect precision, which is crucial for safely identifying eligible blood donors while minimizing false positives. This comprehensive approach underscores the model's strong performance and its potential as a reliable tool for blood donation screening.

There are a few recommendations that can be suggested to further enhance the model's capabilities and ensures its practical capability which are:

- Explore advanced models.
    - Examine the effectiveness of more sophisticated machine learning techniques like Random Forests, Support Vector Machines (SVMs), and Gradient Boosting Machines (GBMs). Particularly for the features that remained significantly skewed following the initial modifications (AST and BIL), these models may be able to capture more complex relationships within the data, which could result in further gains in predicting accuracy.
- Acquire a larger and more diverse dataset.
    - The current dataset, while effective for this study, contains only 615 records. Expanding the dataset with more diverse demographic and biochemical profiles would enhance the model's generalizability and robustness, making it more reliable for broader applications in various populations and medical contexts.
- Implement a real-world validation and continuous monitoring system.
    - To ensure the model's practical utility and safety in medical context, it should be thoroughly validated in a clinical setting with real-time data. A system for continuous monitoring of its performance would allow for adaptive learning and retraining, ensuring it remains accurate and reliable as new data becomes available and medical guidelines evolve.