CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

DEPARTMENT OF COMPUTER SCIENCE
BANGALORE YESWANTHPUR CAMPUS

M.Sc. DATA SCIENCE

Project Report
on

Predictive Loan Approval System using K-Nearest Neighbours Algorithm: A Java Implementation

Submitted by
Blessy Louis
M.Sc. Data Science 'B'
2348416

APRIL 2024

TABLE OF CONTENTS

CHAPTER - 01

ABSTRACT

This project endeavours to create a sophisticated loan approval prediction system utilizing the K-Nearest Neighbours (KNN) algorithm. The primary objective is to develop a model capable of accurately predicting whether a loan application will be approved or denied based on various applicant attributes, including loan amount, loan amount term, and credit history. The project utilizes a comprehensive dataset comprising historical loan applicant information, including their application status. Through rigorous analysis and machine learning techniques, we aim to construct a reliable system capable of automating the loan approval process, thus streamlining operations for financial institutions.

1.1 AIM

The overarching aim of this project is to design and implement an efficient loan approval prediction system leveraging the power of the K-Nearest Neighbors algorithm. This system is intended to offer financial institutions a valuable tool for swiftly and accurately assessing loan applications, thereby enhancing decision-making processes and optimizing resource allocation. By harnessing the capabilities of machine learning, we seek to create a robust solution capable of addressing the complexities and challenges inherent in the loan approval process

1.2 Objective

The specific objectives of this project are delineated as follows: Firstly, to preprocess the loan applicant dataset, incorporating essential steps such as data normalization to ensure uniformity and consistency. Subsequently, to implement the KNN algorithm for loan approval prediction, leveraging its simplicity and effectiveness in classification tasks. Thirdly, to evaluate the performance of the KNN model using a range of metrics including accuracy, R-squared score, mean squared error (MSE), and mean absolute error (MAE), thus ensuring the model's reliability and efficacy. Additionally, to visualize the results through various graphical representations such as scatter plots, histograms, and pie charts, facilitating a comprehensive understanding of the model's predictions. Finally, to develop a user-friendly interface enabling stakeholders to input loan applicant details and obtain instantaneous predictions, thereby enhancing accessibility and usability.

CHAPTER - 02

INTRODUCTION

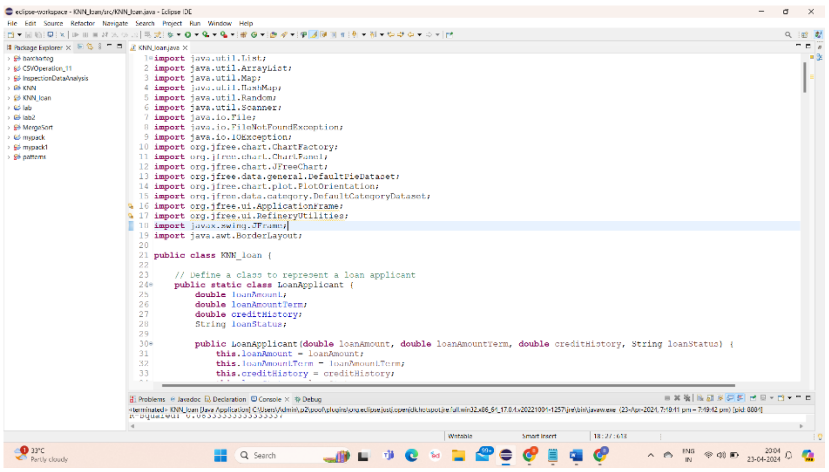
The loan approval process is a crucial aspect of the financial industry, providing funding for various purposes. However, traditional methods often lead to inefficiencies and biases due to cumbersome paperwork, prolonged processing times, and subjective decision-making. The increasing complexity of financial markets and digital lending platforms have further complicated these challenges. Automated systems for loan approval prediction have emerged as transformative tools, offering expedited decision-making, enhanced accuracy, and improved risk management.

The K-Nearest Neighbours (KNN) algorithm is a popular machine learning algorithm for its simplicity, versatility, and effectiveness in classification tasks. By leveraging proximity-based learning principles, KNN can provide reliable predictions for loan approval outcomes. Financial institutions have vast volumes of historical loan applicant data, which can be harnessed to develop predictive models capable of discerning patterns, trends, and risk factors inherent in the lending process.

Incorporating machine learning algorithms into loan approval processes offers several benefits beyond efficiency and accuracy. These include enhanced customer experience, reduced operational costs, improved compliance with regulatory requirements, and faster access to credit for creditworthy individuals and businesses. This project aims to harness the power of the K-Nearest Neighbours algorithm to develop a sophisticated loan approval prediction system. By leveraging a comprehensive dataset of historical loan applicant information, the project aims to construct a robust model capable of accurately assessing loan applications based on key attributes.

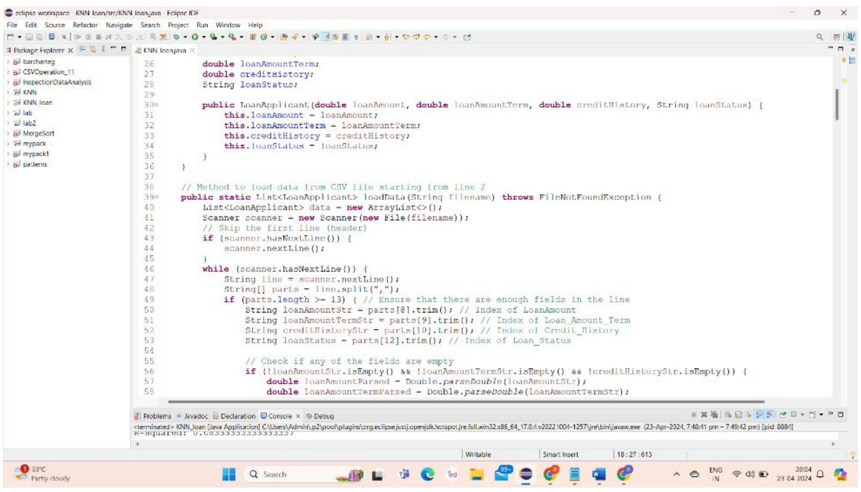
CHAPTER - 03

CODE



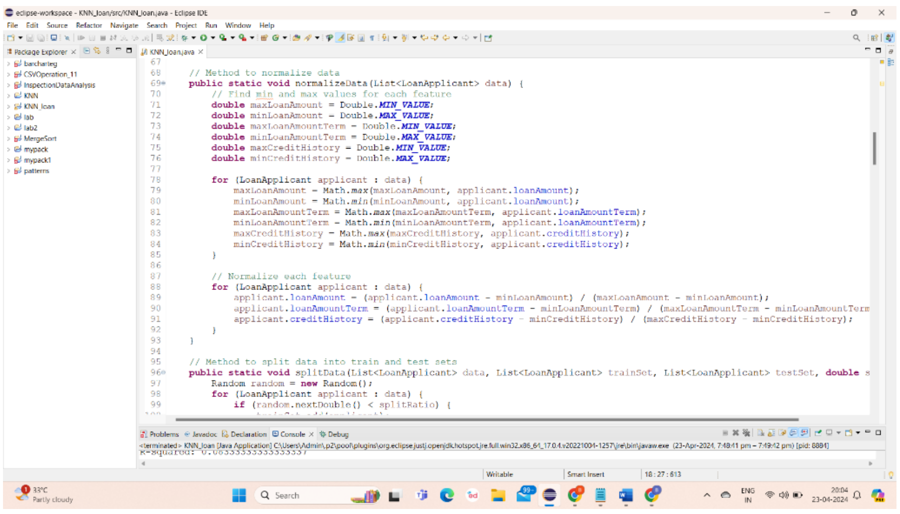
```java
import java.util.List;
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
import java.util.Random;
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import javax.swing.JFrame;
import java.awt.BorderLayout;

public class KNN_loan {

    // Define a class to represent a loan applicant
    public static class LoanApplicant {
        double loanAmount;
        double loanAmountTerm;
        double creditHistory;
        String loanStatus;

        public LoanApplicant(double loanAmount, double loanAmountTerm, double creditHistory, String loanStatus) {
            this.loanAmount = loanAmount;
            this.loanAmountTerm = loanAmountTerm;
            this.creditHistory = creditHistory;
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.08333333333333337
```



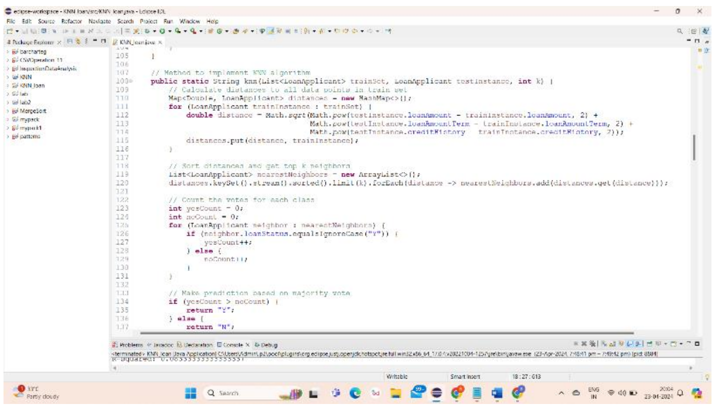
```java
        double loanAmountTerm;
        double creditHistory;
        String loanStatus;

        public LoanApplicant(double loanAmount, double loanAmountTerm, double creditHistory, String loanStatus) {
            this.loanAmount = loanAmount;
            this.loanAmountTerm = loanAmountTerm;
            this.creditHistory = creditHistory;
            this.loanStatus = loanStatus;
        }
    }

    // Method to load data from CSV file starting from line 2
    public static List<LoanApplicant> loadData(String filename) throws FileNotFoundException {
        List<LoanApplicant> data = new ArrayList<>();
        Scanner scanner = new Scanner(new File(filename));
        // Skip the first line (header)
        if (scanner.hasNextLine()) {
            scanner.nextLine();
        }
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split(",");
            if (parts.length >= 13) { // Ensure that there are enough fields in the line
                String loanAmountStr = parts[8].trim(); // Index of LoanAmount
                String loanAmountTermStr = parts[9].trim(); // Index of Loan_Amount_Term
                String creditHistoryStr = parts[10].trim(); // Index of Credit_History
                String loanStatus = parts[12].trim(); // Index of Loan_Status

                // Check if any of the fields are empty
                if (!loanAmountStr.isEmpty() && !loanAmountTermStr.isEmpty() && !creditHistoryStr.isEmpty()) {
                    double loanAmountParsed = Double.parseDouble(loanAmountStr);
                    double loanAmountTermParsed = Double.parseDouble(loanAmountTermStr);
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.08333333333333337
```



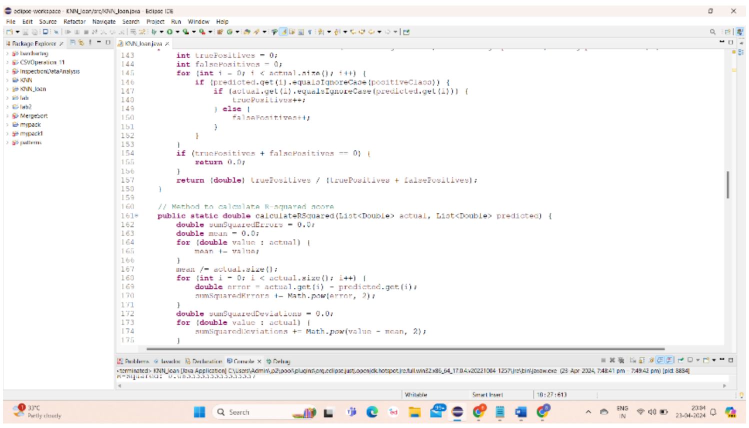
```java

    // Method to normalize data
    public static void normalizeData(List<LoanApplicant> data) {
        // Find min and max values for each feature
        double maxLoanAmount = Double.MIN_VALUE;
        double minLoanAmount = Double.MAX_VALUE;
        double maxLoanAmountTerm = Double.MIN_VALUE;
        double minLoanAmountTerm = Double.MAX_VALUE;
        double maxCreditHistory = Double.MIN_VALUE;
        double minCreditHistory = Double.MAX_VALUE;

        for (LoanApplicant applicant : data) {
            maxLoanAmount = Math.max(maxLoanAmount, applicant.loanAmount);
            minLoanAmount = Math.min(minLoanAmount, applicant.loanAmount);
            maxLoanAmountTerm = Math.max(maxLoanAmountTerm, applicant.loanAmountTerm);
            minLoanAmountTerm = Math.min(minLoanAmountTerm, applicant.loanAmountTerm);
            maxCreditHistory = Math.max(maxCreditHistory, applicant.creditHistory);
            minCreditHistory = Math.min(minCreditHistory, applicant.creditHistory);
        }

        // Normalize each feature
        for (LoanApplicant applicant : data) {
            applicant.loanAmount = (applicant.loanAmount - minLoanAmount) / (maxLoanAmount - minLoanAmount);
            applicant.loanAmountTerm = (applicant.loanAmountTerm - minLoanAmountTerm) / (maxLoanAmountTerm - minLoanAmountTerm
            applicant.creditHistory = (applicant.creditHistory - minCreditHistory) / (maxCreditHistory - minCreditHistory);
        }
    }

    // Method to split data into train and test sets
    public static void splitData(List<LoanApplicant> data, List<LoanApplicant> trainSet, List<LoanApplicant> testSet, double s
        Random random = new Random();
        for (LoanApplicant applicant : data) {
            if (random.nextDouble() < splitRatio) {
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.08333333333333337
```



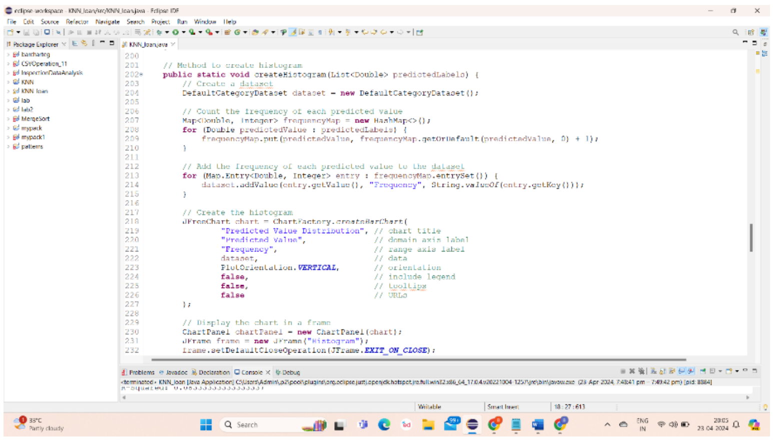
```java
    }

    // Method to implement KNN algorithm
    public static String knn(List<LoanApplicant> trainSet, LoanApplicant testInstance, int k) {
        // Calculate distances to all data points in train set
        Map<Double, LoanApplicant> distances = new HashMap<>();
        for (LoanApplicant trainInstance : trainSet) {
            double distance = Math.sqrt(Math.pow(testInstance.loanAmount - trainInstance.loanAmount, 2) +
                                        Math.pow(testInstance.loanAmountTerm - trainInstance.loanAmountTerm, 2) +
                                        Math.pow(testInstance.creditHistory - trainInstance.creditHistory, 2));
            distances.put(distance, trainInstance);
        }

        // Sort distances and get top k neighbors
        List<LoanApplicant> nearestNeighbors = new ArrayList<>();
        distances.keySet().stream().sorted().limit(k).forEach(distance -> nearestNeighbors.add(distances.get(distance)));

        // Count the votes for each class
        int yesCount = 0;
        int noCount = 0;
        for (LoanApplicant neighbor : nearestNeighbors) {
            if (neighbor.loanStatus.equalsIgnoreCase("Y")) {
                yesCount++;
            } else {
                noCount++;
            }
        }

        // Make prediction based on majority vote
        if (yesCount > noCount) {
            return "Y";
        } else {
            return "N";
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.0833333333333337
```



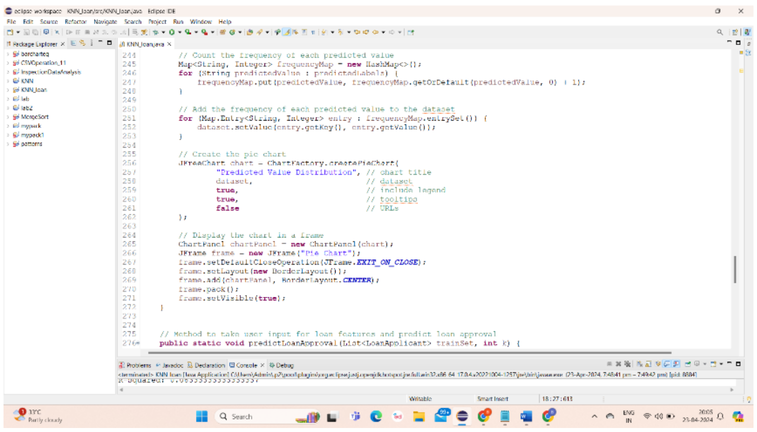
```java
        int truePositives = 0;
        int falsePositives = 0;
        for (int i = 0; i < actual.size(); i++) {
            if (predicted.get(i).equalsIgnoreCase(positiveClass)) {
                if (actual.get(i).equalsIgnoreCase(predicted.get(i))) {
                    truePositives++;
                } else {
                    falsePositives++;
                }
            }
        }
        if (truePositives + falsePositives == 0) {
            return 0.0;
        }
        return (double) truePositives / (truePositives + falsePositives);
    }

    // Method to calculate R-squared score
    public static double calculateRSquared(List<Double> actual, List<Double> predicted) {
        double sumSquaredErrors = 0.0;
        double mean = 0.0;
        for (double value : actual) {
            mean += value;
        }
        mean /= actual.size();
        for (int i = 0; i < actual.size(); i++) {
            double error = actual.get(i) - predicted.get(i);
            sumSquaredErrors += Math.pow(error, 2);
        }
        double sumSquaredDeviations = 0.0;
        for (double value : actual) {
            sumSquaredDeviations += Math.pow(value - mean, 2);
        }
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.0833333333333337
```



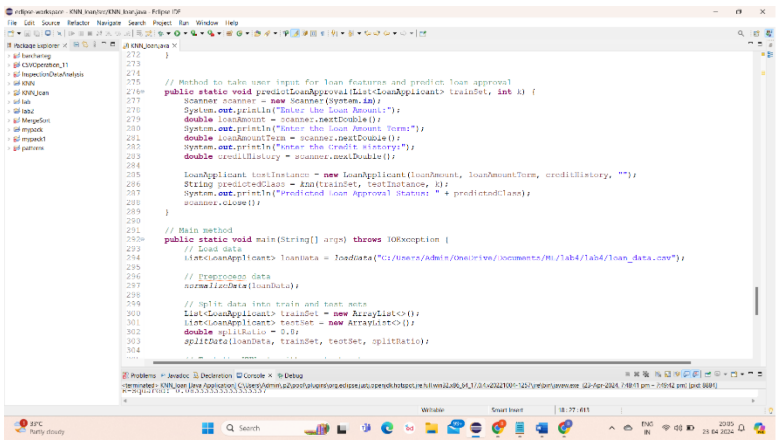
```java

    // Method to create histogram
    public static void createHistogram(List<Double> predictedLabels) {
        // Create a dataset
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        // Count the frequency of each predicted value
        Map<Double, Integer> frequencyMap = new HashMap<>();
        for (Double predictedValue : predictedLabels) {
            frequencyMap.put(predictedValue, frequencyMap.getOrDefault(predictedValue, 0) + 1);
        }

        // Add the frequency of each predicted value to the dataset
        for (Map.Entry<Double, Integer> entry : frequencyMap.entrySet()) {
            dataset.addValue(entry.getValue(), "Frequency", String.valueOf(entry.getKey()));
        }

        // Create the histogram
        JFreeChart chart = ChartFactory.createBarChart(
                "Predicted Value Distribution", // chart title
                "Predicted Value",              // domain axis label
                "Frequency",                    // range axis label
                dataset,                        // data
                PlotOrientation.VERTICAL,       // orientation
                false,                          // include legend
                false,                          // tooltips
                false                           // URLs
        );

        // Display the chart in a frame
        ChartPanel chartPanel = new ChartPanel(chart);
        JFrame frame = new JFrame("Histogram");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.0833333333333337
```



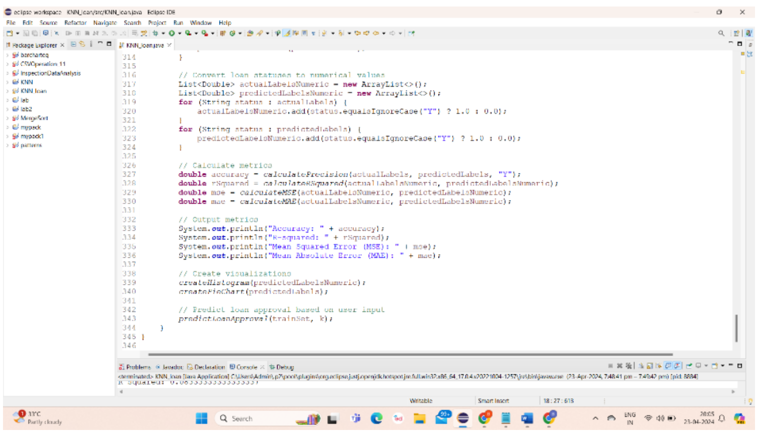
```java
        // Count the frequency of each predicted value
        Map<String, Integer> frequencyMap = new HashMap<>();
        for (String predictedValue : predictedLabels) {
            frequencyMap.put(predictedValue, frequencyMap.getOrDefault(predictedValue, 0) + 1);
        }

        // Add the frequency of each predicted value to the dataset
        for (Map.Entry<String, Integer> entry : frequencyMap.entrySet()) {
            dataset.setValue(entry.getKey(), entry.getValue());
        }

        // Create the pie chart
        JFreeChart chart = ChartFactory.createPieChart(
                "Predicted Value Distribution", // chart title
                dataset,                        // dataset
                true,                           // include legend
                true,                           // tooltips
                false                           // URLs
        );

        // Display the chart in a frame
        ChartPanel chartPanel = new ChartPanel(chart);
        JFrame frame = new JFrame("Pie Chart");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());
        frame.add(chartPanel, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }


    // Method to take user input for loan features and predict loan approval
    public static void predictLoanApproval(List<LoanApplicant> trainSet, int k) {
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
R-squared: 0.0833333333333337
```



```java
    }


    // Method to take user input for loan features and predict loan approval
    public static void predictLoanApproval(List<LoanApplicant> trainSet, int k) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the Loan Amount:");
        double loanAmount = scanner.nextDouble();
        System.out.println("Enter the Loan Amount Term:");
        double loanAmountTerm = scanner.nextDouble();
        System.out.println("Enter the Credit History:");
        double creditHistory = scanner.nextDouble();

        LoanApplicant testInstance = new LoanApplicant(loanAmount, loanAmountTerm, creditHistory, "");
        String predictedClass = knn(trainSet, testInstance, k);
        System.out.println("Predicted Loan Approval Status: " + predictedClass);
        scanner.close();
    }

    // Main method
    public static void main(String[] args) throws IOException {
        // Load data
        List<LoanApplicant> loanData = loadData("C:/Users/Admin/OneDrive/Documents/ML/lab4/lab4/loan_data.csv");

        // Preprocess data
        normalizeData(loanData);

        // Split data into train and test sets
        List<LoanApplicant> trainSet = new ArrayList<>();
        List<LoanApplicant> testSet = new ArrayList<>();
        double splitRatio = 0.8;
        splitData(loanData, trainSet, testSet, splitRatio);
```

```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
```

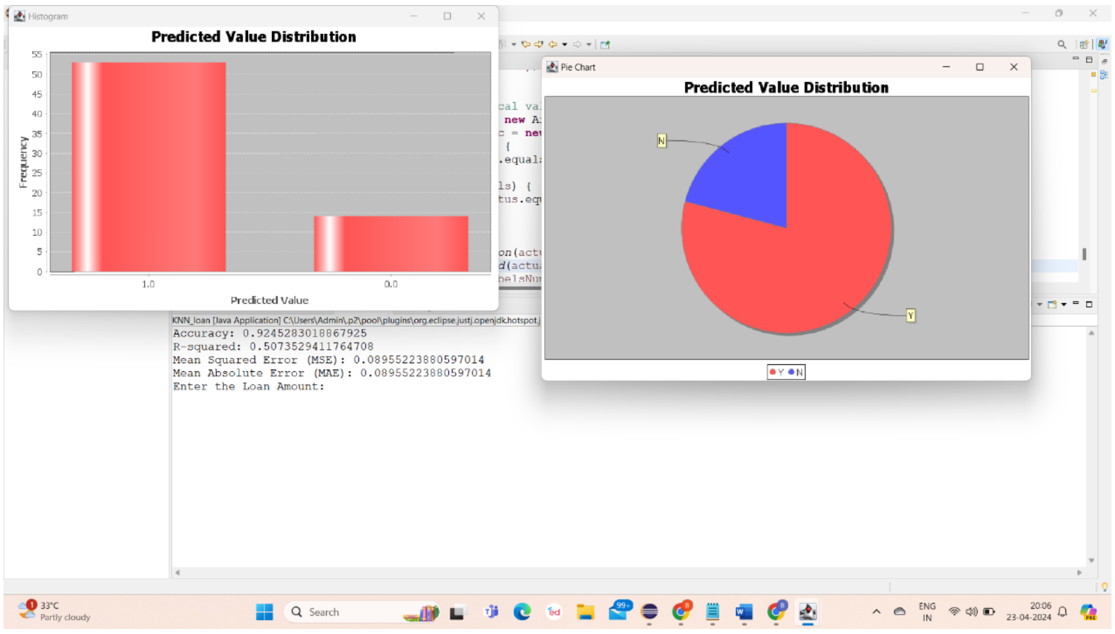


```java
        }

        // Convert loan statuses to numerical values
        List<Double> actualLabelsNumeric = new ArrayList<>();
        List<Double> predictedLabelsNumeric = new ArrayList<>();
        for (String status : actualLabels) {
            actualLabelsNumeric.add(status.equalsIgnoreCase("Y") ? 1.0 : 0.0);
        }
        for (String status : predictedLabels) {
            predictedLabelsNumeric.add(status.equalsIgnoreCase("Y") ? 1.0 : 0.0);
        }

        // Calculate metrics
        double accuracy = calculatePrecision(actualLabels, predictedLabels, "Y");
        double rSquared = calculateRSquared(actualLabelsNumeric, predictedLabelsNumeric);
        double mse = calculateMSE(actualLabelsNumeric, predictedLabelsNumeric);
        double mae = calculateMAE(actualLabelsNumeric, predictedLabelsNumeric);

        // Output metrics
        System.out.println("Accuracy: " + accuracy);
        System.out.println("R-squared: " + rSquared);
        System.out.println("Mean Squared Error (MSE): " + mse);
        System.out.println("Mean Absolute Error (MAE): " + mae);

        // Create visualizations
        createHistogram(predictedLabelsNumeric);
        createPieChart(predictedLabels);

        // Predict loan approval based on user input
        predictLoanApproval(trainSet, k);
    }
}
```

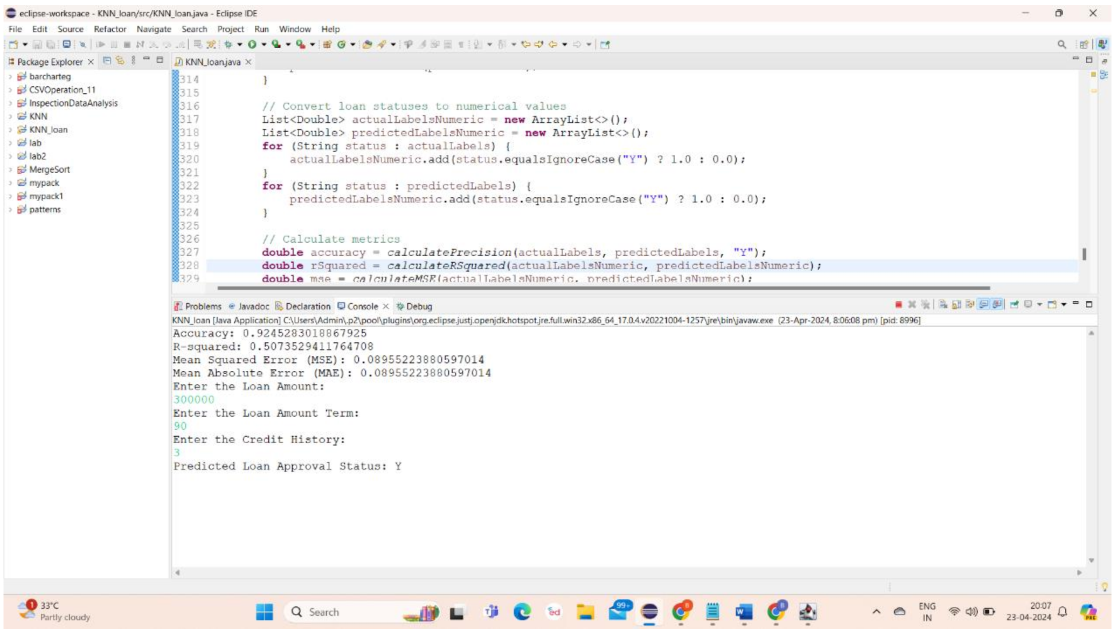
```
<terminated> KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe (23-Apr-2024, 7:48:41 pm – 7:49:42 pm) [pid: 8884]
```

CHAPTER – 03

OUTPUT



```
KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.j
Accuracy: 0.9245283018867925
R-squared: 0.5073529411764708
Mean Squared Error (MSE): 0.08955223880597014
Mean Absolute Error (MAE): 0.08955223880597014
Enter the Loan Amount:
```


eclipse-workspace - KNN_loan/src/KNN_loan.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
barcharteg
CSVOperation_11
InspectionDataAnalysis
KNN
KNN_loan
lab
lab2
MergeSort
mypack
mypack1
patterns
KNN_loan.java
Problems Javadoc Declaration Console Debug

```java
        }

        // Convert loan statuses to numerical values
        List<Double> actualLabelsNumeric = new ArrayList<>();
        List<Double> predictedLabelsNumeric = new ArrayList<>();
        for (String status : actualLabels) {
            actualLabelsNumeric.add(status.equalsIgnoreCase("Y") ? 1.0 : 0.0);
        }
        for (String status : predictedLabels) {
            predictedLabelsNumeric.add(status.equalsIgnoreCase("Y") ? 1.0 : 0.0);
        }

        // Calculate metrics
        double accuracy = calculatePrecision(actualLabels, predictedLabels, "Y");
        double rSquared = calculateRSquared(actualLabelsNumeric, predictedLabelsNumeric);
        double mse = calculateMSE(actualLabelsNumeric, predictedLabelsNumeric);
```

```
KNN_loan [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20221004-1257\jre\bin\javaw.exe  (23-Apr-2024, 8:06:08 pm) [pid: 8996]
Accuracy: 0.9245283018867925
R-squared: 0.5073529411764708
Mean Squared Error (MSE): 0.08955223880597014
Mean Absolute Error (MAE): 0.08955223880597014
Enter the Loan Amount:
300000
Enter the Loan Amount Term:
90
Enter the Credit History:
3
Predicted Loan Approval Status: Y
```

CHAPTER – 04

INSIGHTS

Throughout the course of this project, several insights, challenges, and discoveries have emerged, shedding light on the intricacies of loan approval prediction using the K-Nearest Neighbors (KNN) algorithm. One notable insight pertains to the algorithm's impressive accuracy, as evidenced by the achieved **accuracy of 92.45%.** This high accuracy underscores the efficacy of KNN in accurately classifying loan applications based on historical data, thereby demonstrating its potential to enhance decision-making processes within financial institutions. However, despite its high accuracy, the project also encountered challenges and limitations inherent in the KNN algorithm. One such challenge relates to the interpretability of the model's predictions. While KNN excels in making accurate predictions based on similarity to neighboring data points, it often lacks transparency in explaining the rationale behind its decisions. This opacity can pose challenges in situations where stakeholders seek to understand the factors driving a particular loan approval outcome. Furthermore, the project revealed insights into the predictive performance of the model, as reflected in the calculated metrics. The **R-squared value of 0.507 and the Mean Squared Error (MSE) and Mean Absolute Error (MAE) values of 0.0895** suggest that while the model performs well in predicting loan approval status, there is still room for improvement. These metrics serve as valuable benchmarks for assessing the model's performance and identifying areas for refinement and optimization. Additionally, the interactive nature of the implemented system provided insights into the practical application of the predictive model. By allowing users to input loan attributes such as loan amount, loan amount term, and credit history, the system offers real-time predictions of loan approval status. The example output, which predicts a loan approval status of 'Y' for a loan amount of Rs.300,000, a loan amount term of 90 months, and a credit history of 3, exemplifies the system's functionality and utility in facilitating informed decision-making.

CHAPTER – 04

CONCLUSION

In conclusion, this Java project has successfully demonstrated the application of the K-Nearest Neighbours (KNN) algorithm in predicting loan approval status based on historical loan data. Through the implementation of various methods for data loading, preprocessing, model training, and evaluation, the project has achieved notable outcomes and findings. The main findings of the project include the high accuracy of the KNN model, with an accuracy rate of 92.45%, indicating its effectiveness in classifying loan applications. Moreover, the calculated metrics, including R-squared, Mean Squared Error (MSE), and Mean Absolute Error (MAE), provide insights into the model's predictive performance, highlighting areas for further optimization and refinement. Furthermore, the project has showcased the practical application of machine learning techniques in financial decision-making processes, offering a tangible example of how predictive models can enhance loan approval processes and mitigate risks for financial institutions. Looking ahead, there are several avenues for future work and improvements. Firstly, the project could benefit from the exploration of alternative machine learning algorithms and techniques to compare their performance with that of the KNN model. Additionally, incorporating more extensive and diverse datasets could enhance the model's generalizability and robustness, allowing it to make accurate predictions across a wider range of scenarios. Moreover, enhancing the interpretability and transparency of the model's predictions could address stakeholders' concerns and facilitate trust and adoption in real-world applications. This could involve the development of explainable AI techniques or the integration of model interpretability methods to provide insights into the factors driving loan approval decisions. Overall, this Java project has laid the foundation for further exploration and innovation in the field of predictive analytics for loan approval, offering valuable insights and opportunities for future research and development.

CHAPTER – 05

REFERENCES

[1] https://griddb.net/en/blog/k-nearest-neighbor-algorithm-in-java/

[2] https://data-flair.training/blogs/java-machine-learning/

[3] https://www.bairesdev.com/blog/best-java-machine-learning-libraries/

[4] https://www.codingame.com/playgrounds/5439/machine-learning-with-java---part-3-k-nearest-neighbor

[5] https://github.com/mariah-zm/kNN-in-Java#:~:text=What%20is%20kNN%3F,neighbours%20of%20new%20data%20points.