



**L**OVELY  
**P**ROFESSIONAL  
**U**NIVERSITY

# **INT234 -PREDICTIVE ANALYTICS**

## **Data Science Loan Report**

Submitted by

Yarram Gowtham Kumar Reddy

Registration No. 12208301

Programme and Section: Computer Science and Engineering-RK22MRB44

Course Code: INT234

**Under the Guidance of**

**DR. Mrinalini Rana:**

**22138**

**Discipline of CSE**

**Lovely Professional University, Phagwara**

**School of Computer Science and Engineering**

# DECLARATION

I hereby declare that I have successfully completed my project titled "Data Science salaries Report Using R Programming." This project was conducted as part of my academic requirements for the degree of B.Tech. in Computer Science & Engineering at Lovely Professional University, Phagwara.

The project commenced on “20-10-2024” and concluded on “15-11-2024”. Throughout this period, I dedicated myself fully to the training and implementation of various machine learning algorithms using R, including K-Nearest Neighbors, Naive Bayes, Support Vector Machines, and Decision Tree. My work involved extensive data analysis, model training, and performance evaluation to derive insights from the dataset.

I affirm that the learning outcomes achieved during this project meet the necessary requirements for my degree program. I appreciate the support and guidance provided by my faculty and peers throughout this endeavor.

Yarram Gowtham Kumar reddy  
12106946

## **Acknowledgement**

I would like to express my sincere gratitude to my faculty and peers for their unwavering support and guidance throughout the duration of this project. Their insights and encouragement were invaluable in helping me navigate the complexities of predictive analysis. I am especially thankful to my mentor, DR. Mrinalini Rana, whose expertise and constructive feedback significantly enhanced the quality of my work.

I would also like to extend my appreciation to the developers of the R programming language, whose powerful tools and libraries facilitated my data analysis and modelling efforts. The resources available through the R community were instrumental in overcoming challenges encountered during the project. Additionally, I am grateful to my fellow students for their collaborative spirit and for sharing knowledge that enriched my learning experience.

With Regards

Yarram Gowtham Kumar Reddy

## Introduction:

The **Loan Project** aims to explore salary trends within the data science field by utilizing various machine learning algorithms. We will work with the dataset `ds_salaries.csv`, which provides detailed information about Loan status, including critical factors such as loan amount, person income, person home, loan ratio, loan grade, and etc. To uncover patterns that influence loan status levels across different roles, we will employ techniques like **K-Nearest Neighbor (KNN)**, **Decision Trees**, **Naive Bayes**, and **Support Vector Machines (SVM)**. Our analysis will focus on key indicators including loan status, loan intent, loan grade, loan amount and person income.

These factors have a significant impact on loan status outcomes, making them essential for our predictive models. By categorizing the data based on loan grade(e.g., A, B, C, D), empl (e.g., Full-Time, Contract), and company sizes (e.g., 0 ,), we can gain a deeper understanding of how these elements affect salary distributions among different groups of professionals. The insights generated from this analysis will be valuable for stakeholders such as policymakers and industry leaders by providing them with information on salary benchmarks and trends. This project highlights the potential of machine learning to analyze complex datasets and derive actionable insights that can enhance decision- making in the labor market.

## Objectives / Scope of the Analysis:

The primary objectives of this analysis are:

1. **Data Preprocessing:** Clean and normalize the dataset for effective modeling.
2. **Model Implementation:** Apply multiple machine learning algorithms to classify happiness levels.
3. **Performance Evaluation:** Compare the accuracy of different models to identify the most effective approach for predicting happiness levels.
4. **Visualization:** Present findings through visual representations for better understanding.

## Dataset Overview:

The dataset used, `Datasetoncrime.csv`, consists of various attributes of loan applicants, including:

- **Demographic Information:** Age, income, employment status, etc.
- **Financial Information:** Loan amount, credit score, and any additional factors affecting loan status.
- **Target Variable:** loan status represents the classification label, with possible values of

Normal: Standard or approved loans

Suspect: Applications that need additional review

Pathologic: High-risk applications likely to be rejected

## **Data Preprocessing**

The data preprocessing steps included:

**Handling Missing Values:** Checking and addressing any missing data points.

**Converting Data Types:** Ensuring all character variables are converted to factors for categorical classification.

**Scaling Features:** For KNN and SVM, scaling numeric features was essential to standardize data across varying ranges.

**Train-Test Split:** Dividing the data into 80% for training and 20% for testing

## **Algorithms Used:**

To analyze salary trends and predict salary levels based on key features from the dataset, several machine learning algorithms were implemented:

**1.K-Nearest Neighbors (KNN):** A simple yet effective algorithm used for classification and regression. KNN predicts salaries based on the majority class among k-nearest neighbors in the feature space.

**2.Naive Bayes:** The Naive Bayes classifier is a probabilistic model that assumes independence between features. This algorithm was chosen for its simplicity and speed, particularly effective with categorical data. The model assigns a probability to each class based on the training data and uses this probability to classify test data.

**3.Decision Trees:** A non-parametric method used for classification and regression tasks. Decision Trees create a model that predicts salary based on decision rules derived from input features. This algorithm helps visualize decision-making processes.

**4.Support Vector Machines (SVM):** SVM finds the hyperplane that best separates different classes in the feature space. It was used to classify salary ranges based on various features, effectively handling high-dimensional spaces.

## **Performance Comparison:**

We evaluate the performance of the different machine learning algorithms used to predict salaries in the data science sector. The models were assessed using key metrics that indicate their accuracy and effectiveness in making predictions. The following metrics were utilized:

**Accuracy:** It is the proportion of correct predictions made by the model out of all predictions. Accuracy gives an overall measure of how often the classifier is correct across all classes. A higher accuracy indicates that the model is correctly classifying most of the data, but it may not be reliable if the data is imbalanced

**Precision:** It is the proportion of positive predictions that are correct. It measures the accuracy of the positive predictions. High precision means that when the model predicts a positive class, it is highly likely to be correct. It's particularly important in cases where false positives are costly.

**Recall:** It is the proportion of actual positive instances that were correctly identified by the model. It measures how well the model captures all actual positive cases. High recall indicates that the model correctly identifies most of the positive cases. It's particularly useful in situations where missing a positive case (false negative) is costly, such as in medical diagnoses where missing a disease can have serious consequences.

**F1 – Score:** It is the harmonic mean of precision and recall. It balances the trade-off between precision and recall, especially when there is an uneven class distribution. The F1-Score provides a balance between precision and recall. A high F1 score means that both precision and recall are performing well. It is particularly useful when you need to balance the importance of both false positives and false negatives.

**Error Rate:** It is the proportion of incorrect predictions made by the model. It is the complement of accuracy. The error rate tells you how often the model is wrong. A lower error rate indicates better performance. It is a simple metric, but it doesn't distinguish between the types of errors

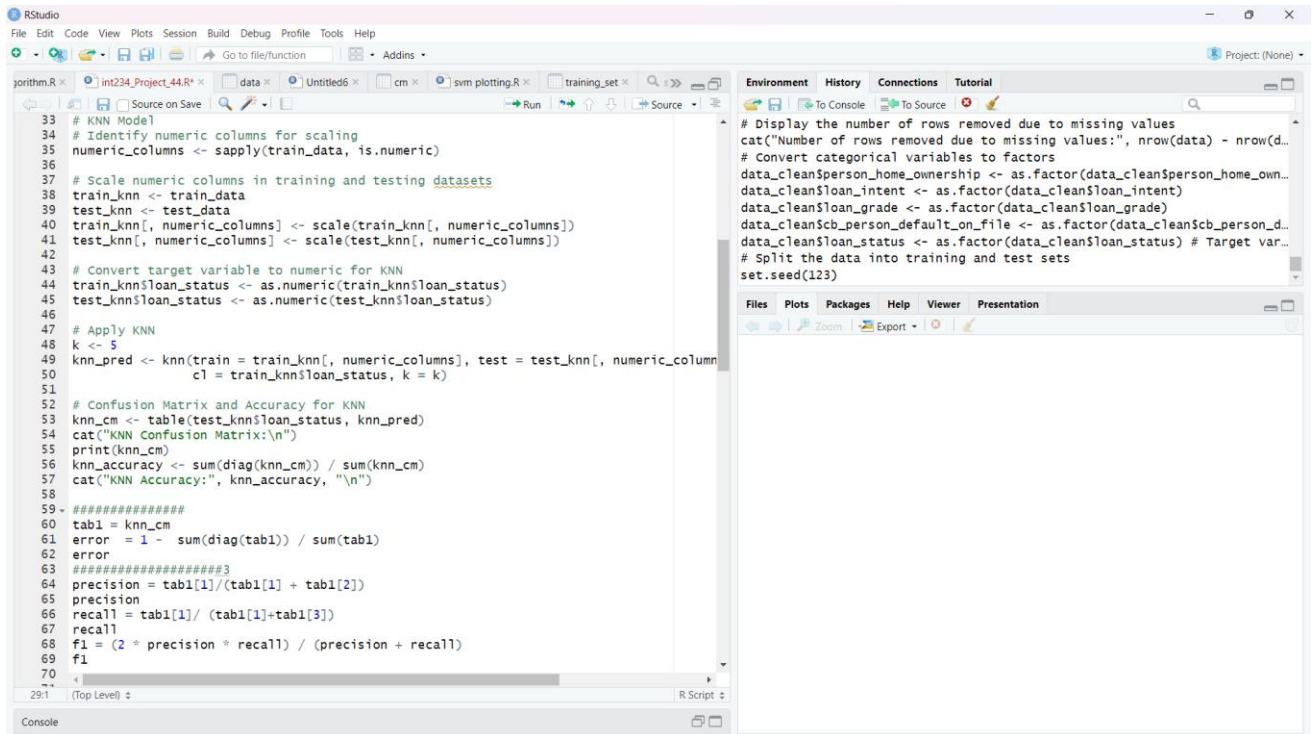
## **Summary of Model Performance:**

The table below summarizes the performance of each algorithm based on the metrics mentioned above:

<b>Algorithm</b>	<b>Accuracy</b>	<b>Error Rate</b>	<b>Precision</b>	<b>Recall</b>	<b>F1 Score</b>
<b>KNN</b>	0.84	0.16	0.86	0.94	0.9
<b>Naive Bayes</b>	0.84	0.16	0.86	0.88	0.87
<b>Decision Tree</b>	0.92	0.08	0.92	1	0.95
<b>SVM</b>	0.87	0.08	0.92	1	0.95

## About the Dataset:

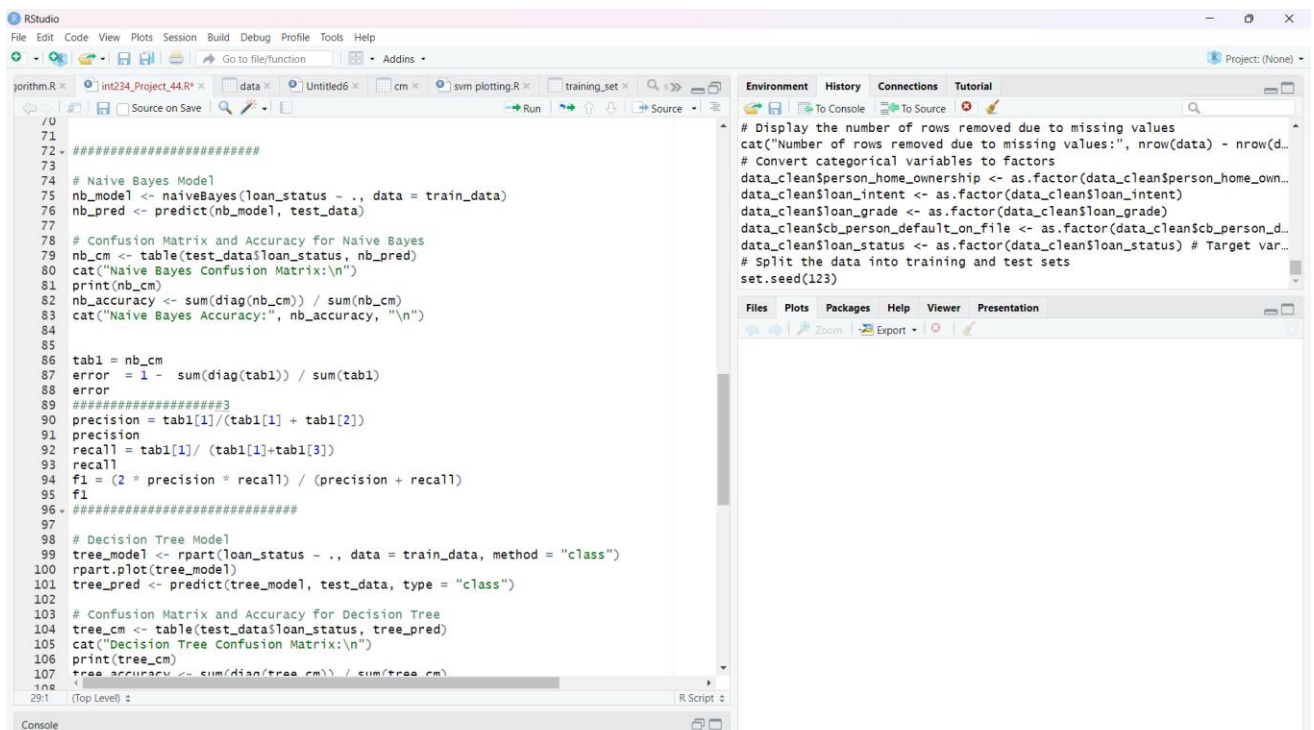
### 1) KNN



```
33 # KNN Model
34 # Identify numeric columns for scaling
35 numeric_columns <- apply(train_data, is.numeric)
36
37 # Scale numeric columns in training and testing datasets
38 train_knn <- train_data
39 test_knn <- test_data
40 train_knn[, numeric_columns] <- scale(train_knn[, numeric_columns])
41 test_knn[, numeric_columns] <- scale(test_knn[, numeric_columns])
42
43 # Convert target variable to numeric for KNN
44 train_knn$loan_status <- as.numeric(train_knn$loan_status)
45 test_knn$loan_status <- as.numeric(test_knn$loan_status)
46
47 # Apply KNN
48 k <- 5
49 knn_pred <- knn(train = train_knn[, numeric_columns], test = test_knn[, numeric_columns],
50               cl = train_knn$loan_status, k = k)
51
52 # Confusion Matrix and Accuracy for KNN
53 knn_cm <- table(test_knn$loan_status, knn_pred)
54 cat("KNN Confusion Matrix:\n")
55 print(knn_cm)
56 knn_accuracy <- sum(diag(knn_cm)) / sum(knn_cm)
57 cat("KNN Accuracy:", knn_accuracy, "\n")
58
59 #####
60 tab1 = knn_cm
61 error = 1 - sum(diag(tab1)) / sum(tab1)
62 error
63 #####
64 precision = tab1[1]/(tab1[1] + tab1[2])
65 precision
66 recall = tab1[1]/ (tab1[1]+tab1[3])
67 recall
68 f1 = (2 * precision * recall) / (precision + recall)
69 f1
70
71 (Top Level) z
```

```
# Display the number of rows removed due to missing values
cat("Number of rows removed due to missing values:", nrow(data) - nrow(d...
# Convert categorical variables to factors
data_clean$person_home_ownership <- as.factor(data_clean$person_home_own...
data_clean$loan_intent <- as.factor(data_clean$loan_intent)
data_clean$loan_grade <- as.factor(data_clean$loan_grade)
data_clean$cb_person_default_on_file <- as.factor(data_clean$cb_person_d...
data_clean$loan_status <- as.factor(data_clean$loan_status) # Target var...
# Split the data into training and test sets
set.seed(123)
```

### 2) Naïve Bayes

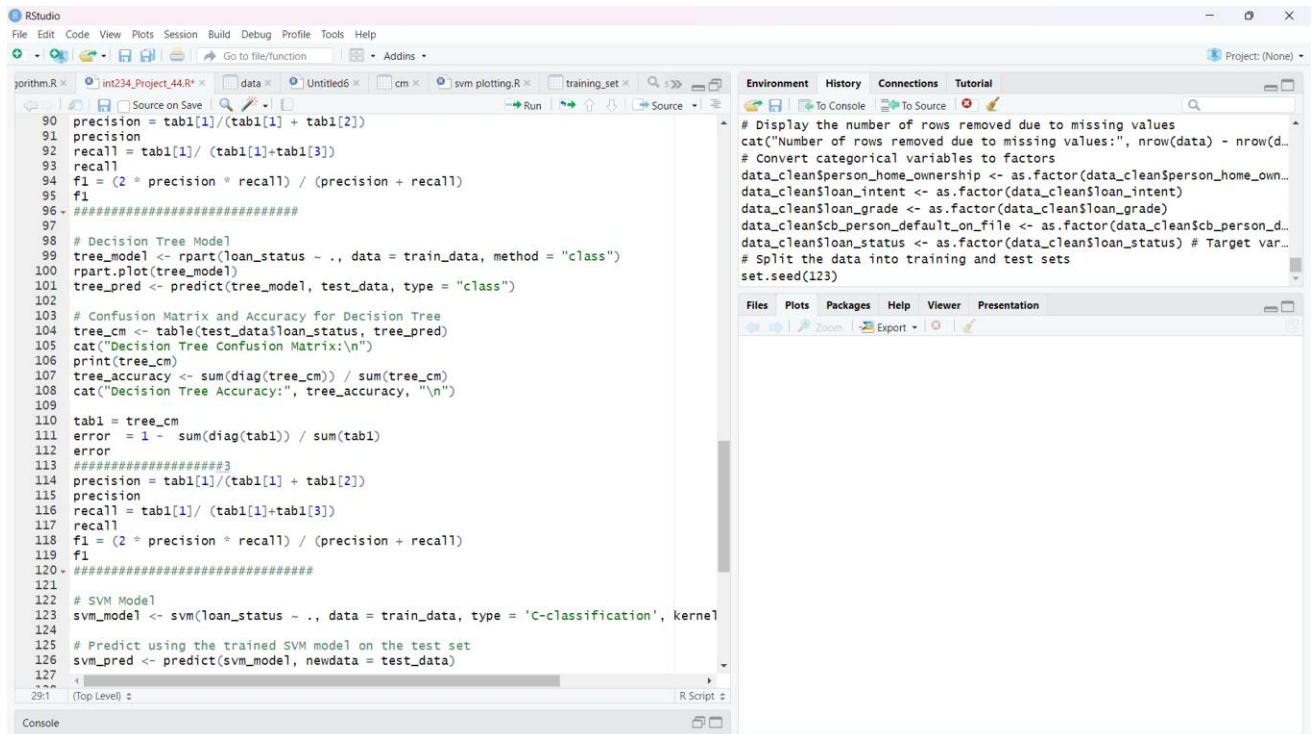


```
70
71 #####
72
73 # Naive Bayes Model
74 nb_model <- naiveBayes(loan_status ~ ., data = train_data)
75 nb_pred <- predict(nb_model, test_data)
76
77 # Confusion Matrix and Accuracy for Naive Bayes
78 nb_cm <- table(test_data$loan_status, nb_pred)
79 cat("Naive Bayes Confusion Matrix:\n")
80 print(nb_cm)
81 nb_accuracy <- sum(diag(nb_cm)) / sum(nb_cm)
82 cat("Naive Bayes Accuracy:", nb_accuracy, "\n")
83
84
85
86 tab1 = nb_cm
87 error = 1 - sum(diag(tab1)) / sum(tab1)
88 error
89 #####
90 precision = tab1[1]/(tab1[1] + tab1[2])
91 precision
92 recall = tab1[1]/ (tab1[1]+tab1[3])
93 recall
94 f1 = (2 * precision * recall) / (precision + recall)
95 f1
96 #####
97
98 # Decision Tree Model
99 tree_model <- rpart(loan_status ~ ., data = train_data, method = "class")
100 rpart.plot(tree_model)
101 tree_pred <- predict(tree_model, test_data, type = "class")
102
103 # Confusion Matrix and Accuracy for Decision Tree
104 tree_cm <- table(test_data$loan_status, tree_pred)
105 cat("Decision Tree Confusion Matrix:\n")
106 print(tree_cm)
107 tree_accuracy <- sum(diag(tree_cm)) / sum(tree_cm)
108
109 (Top Level) z
```

```
# Display the number of rows removed due to missing values
cat("Number of rows removed due to missing values:", nrow(data) - nrow(d...
# Convert categorical variables to factors
data_clean$person_home_ownership <- as.factor(data_clean$person_home_own...
data_clean$loan_intent <- as.factor(data_clean$loan_intent)
data_clean$loan_grade <- as.factor(data_clean$loan_grade)
data_clean$cb_person_default_on_file <- as.factor(data_clean$cb_person_d...
data_clean$loan_status <- as.factor(data_clean$loan_status) # Target var...
# Split the data into training and test sets
set.seed(123)
```



### 3) Decision Tree:



```
90 precision = tab1[1]/(tab1[1] + tab1[2])
91 precision
92 recall = tab1[1]/ (tab1[1]+tab1[3])
93 recall
94 f1 = (2 * precision * recall) / (precision + recall)
95 f1
96 #####
97
98 # Decision Tree Model
99 tree_model <- rpart(loan_status ~ ., data = train_data, method = "class")
100 rpart.plot(tree_model)
101 tree_pred <- predict(tree_model, test_data, type = "class")
102
103 # Confusion Matrix and Accuracy for Decision Tree
104 tree_cm <- table(test_data$loan_status, tree_pred)
105 cat("Decision Tree Confusion Matrix:\n")
106 print(tree_cm)
107 tree_accuracy <- sum(diag(tree_cm)) / sum(tree_cm)
108 cat("Decision Tree Accuracy:", tree_accuracy, "\n")
109
110 tab1 = tree_cm
111 error = 1 - sum(diag(tab1)) / sum(tab1)
112 error
113 #####3
114 precision = tab1[1]/(tab1[1] + tab1[2])
115 precision
116 recall = tab1[1]/ (tab1[1]+tab1[3])
117 recall
118 f1 = (2 * precision * recall) / (precision + recall)
119 f1
120 #####
121
122 # SVM Model
123 svm_model <- svm(loan_status ~ ., data = train_data, type = 'C-classification', kernel
124
125 # Predict using the trained SVM model on the test set
126 svm_pred <- predict(svm_model, newdata = test_data)
127
128 <
29:1 (Top Level) ± R Script ±
```

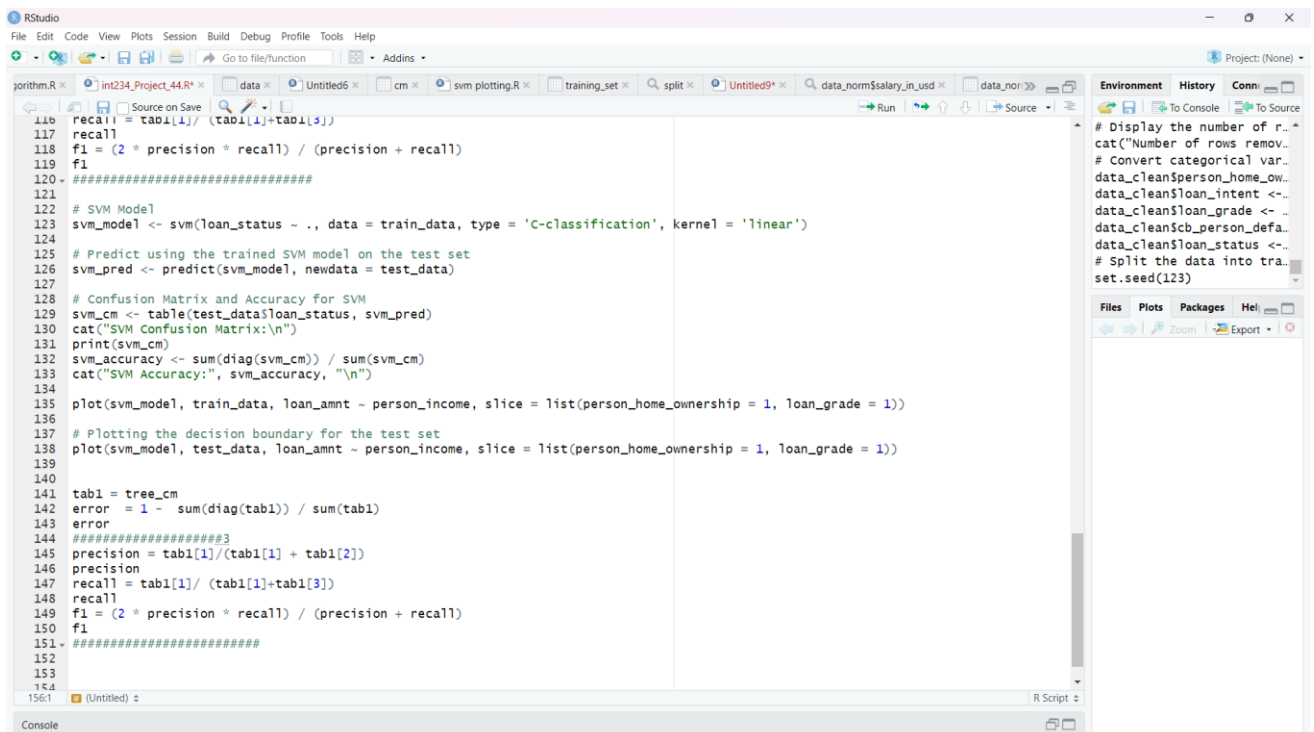
Environment History Connections Tutorial

```
# Display the number of rows removed due to missing values
cat("Number of rows removed due to missing values:", nrow(data) - nrow(d...
# Convert categorical variables to factors
data_clean$person_home_ownership <- as.factor(data_clean$person_home_own...
data_clean$loan_intent <- as.factor(data_clean$loan_intent)
data_clean$loan_grade <- as.factor(data_clean$loan_grade)
data_clean$cb_person_default_on_file <- as.factor(data_clean$cb_person_d...
data_clean$loan_status <- as.factor(data_clean$loan_status) # Target var...
# Split the data into training and test sets
set.seed(123)
```

Files Plots Packages Help Viewer Presentation

Zoom Export

### 4) SVM:

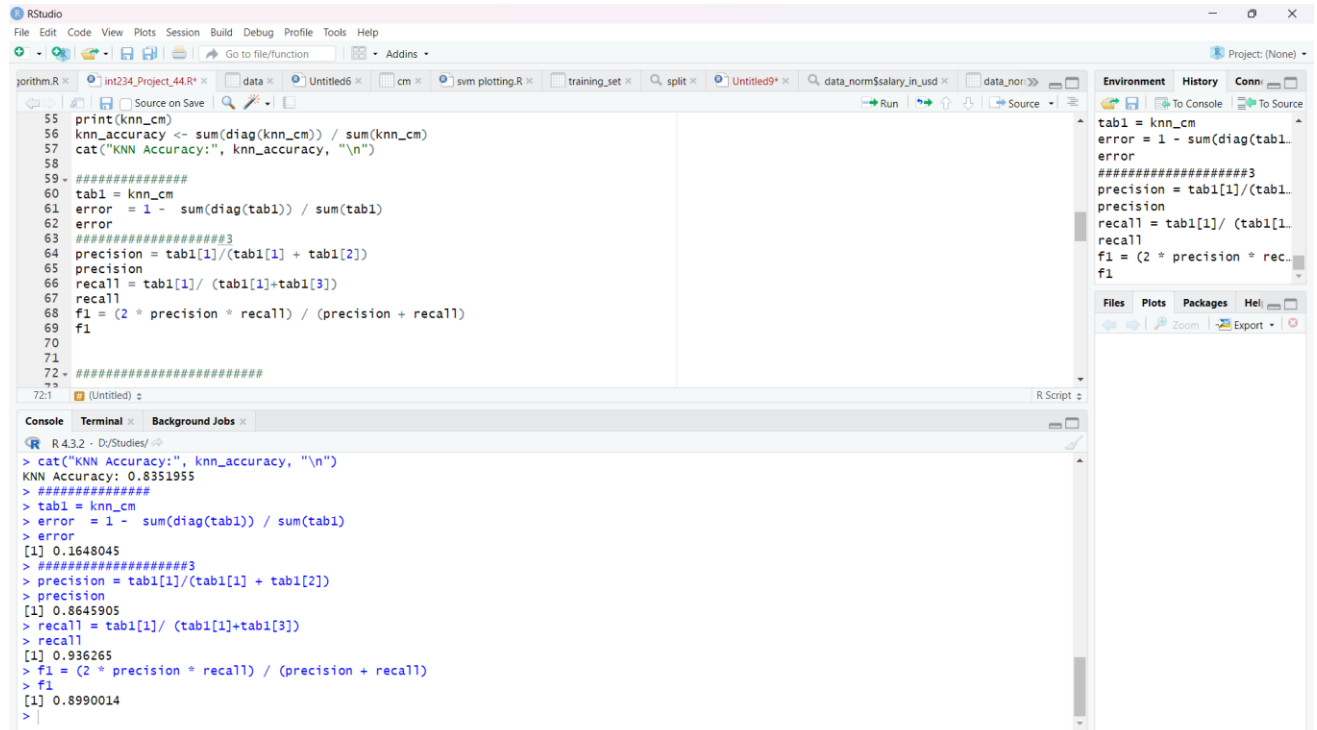


```
116 recall = tab1[1]/ (tab1[1]+tab1[3])
117 recall
118 f1 = (2 * precision * recall) / (precision + recall)
119 f1
120 #####
121
122 # SVM Model
123 svm_model <- svm(loan_status ~ ., data = train_data, type = 'C-classification', kernel = 'linear')
124
125 # Predict using the trained SVM model on the test set
126 svm_pred <- predict(svm_model, newdata = test_data)
127
128 # Confusion Matrix and Accuracy for SVM
129 svm_cm <- table(test_data$loan_status, svm_pred)
130 cat("SVM Confusion Matrix:\n")
131 print(svm_cm)
132 svm_accuracy <- sum(diag(svm_cm)) / sum(svm_cm)
133 cat("SVM Accuracy:", svm_accuracy, "\n")
134
135 plot(svm_model, train_data, loan_amnt ~ person_income, slice = list(person_home_ownership = 1, loan_grade = 1))
136
137 # Plotting the decision boundary for the test set
138 plot(svm_model, test_data, loan_amnt ~ person_income, slice = list(person_home_ownership = 1, loan_grade = 1))
139
140
141 tab1 = tree_cm
142 error = 1 - sum(diag(tab1)) / sum(tab1)
143 error
144 #####3
145 precision = tab1[1]/(tab1[1] + tab1[2])
146 precision
147 recall = tab1[1]/ (tab1[1]+tab1[3])
148 recall
149 f1 = (2 * precision * recall) / (precision + recall)
150 f1
151 #####
152
153
154
156:1 (Untitled) ± R Script ±
```

Environment History Conn: Files Plots Packages Hel: Zoom Export

## CODE OUTPUTS:

### K-Nearest Neighbors output:



The screenshot shows the RStudio interface with a script editor on the left, a console at the bottom, and an environment pane on the right. The script editor contains R code for K-Nearest Neighbors classification. The console shows the execution output, and the environment pane shows the objects created during the process.

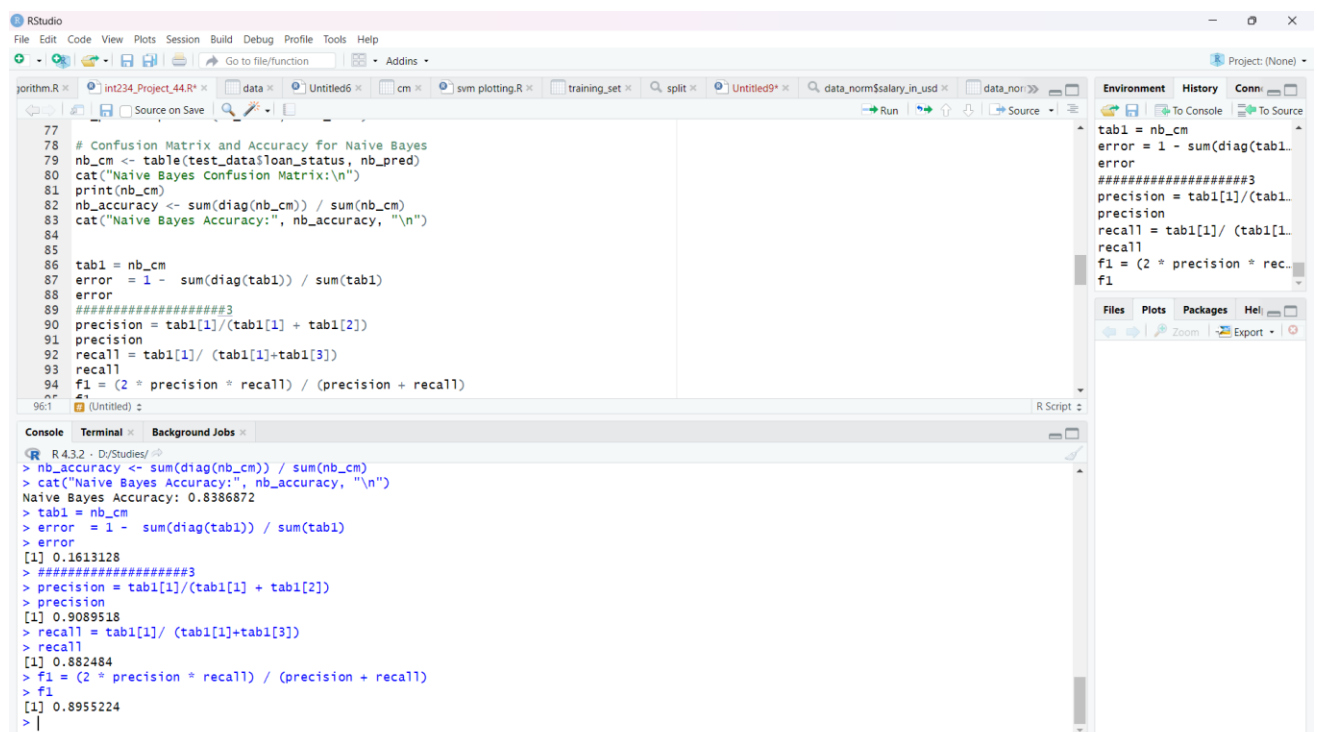
```
55 print(knn_cm)
56 knn_accuracy <- sum(diag(knn_cm)) / sum(knn_cm)
57 cat("KNN Accuracy:", knn_accuracy, "\n")
58
59 #####
60 tab1 = knn_cm
61 error = 1 - sum(diag(tab1)) / sum(tab1)
62 error
63 #####3
64 precision = tab1[1]/(tab1[1] + tab1[2])
65 precision
66 recall = tab1[1]/ (tab1[1]+tab1[3])
67 recall
68 f1 = (2 * precision * recall) / (precision + recall)
69 f1
70
71
72 - #####
```

```
> cat("KNN Accuracy:", knn_accuracy, "\n")
KNN Accuracy: 0.8351955
> #####
> tab1 = knn_cm
> error = 1 - sum(diag(tab1)) / sum(tab1)
> error
[1] 0.1648045
> #####3
> precision = tab1[1]/(tab1[1] + tab1[2])
> precision
[1] 0.8645905
> recall = tab1[1]/ (tab1[1]+tab1[3])
> recall
[1] 0.936265
> f1 = (2 * precision * recall) / (precision + recall)
> f1
[1] 0.8990014
> |
```

Environment pane:

```
tab1 = knn_cm
error = 1 - sum(diag(tab1)) / sum(tab1)
#####3
precision = tab1[1]/(tab1[1] + tab1[2])
recall = tab1[1]/ (tab1[1]+tab1[3])
f1 = (2 * precision * recall) / (precision + recall)
```

### Naive bayes output:



The screenshot shows the RStudio interface with a script editor on the left, a console at the bottom, and an environment pane on the right. The script editor contains R code for Naive Bayes classification. The console shows the execution output, and the environment pane shows the objects created during the process.

```
77
78 # Confusion Matrix and Accuracy for Naive Bayes
79 nb_cm <- table(test_data$loan_status, nb_pred)
80 cat("Naive Bayes Confusion Matrix:\n")
81 print(nb_cm)
82 nb_accuracy <- sum(diag(nb_cm)) / sum(nb_cm)
83 cat("Naive Bayes Accuracy:", nb_accuracy, "\n")
84
85
86 tab1 = nb_cm
87 error = 1 - sum(diag(tab1)) / sum(tab1)
88 error
89 #####3
90 precision = tab1[1]/(tab1[1] + tab1[2])
91 precision
92 recall = tab1[1]/ (tab1[1]+tab1[3])
93 recall
94 f1 = (2 * precision * recall) / (precision + recall)
95 f1
96 |
```

```
> nb_accuracy <- sum(diag(nb_cm)) / sum(nb_cm)
> cat("Naive Bayes Accuracy:", nb_accuracy, "\n")
Naive Bayes Accuracy: 0.8386872
> tab1 = nb_cm
> error = 1 - sum(diag(tab1)) / sum(tab1)
> error
[1] 0.1613128
> #####3
> precision = tab1[1]/(tab1[1] + tab1[2])
> precision
[1] 0.9089518
> recall = tab1[1]/ (tab1[1]+tab1[3])
> recall
[1] 0.882484
> f1 = (2 * precision * recall) / (precision + recall)
> f1
[1] 0.8955224
> |
```

Environment pane:

```
tab1 = nb_cm
error = 1 - sum(diag(tab1)) / sum(tab1)
#####3
precision = tab1[1]/(tab1[1] + tab1[2])
recall = tab1[1]/ (tab1[1]+tab1[3])
f1 = (2 * precision * recall) / (precision + recall)
```

## Decision Tree output:

This screenshot shows the RStudio interface with a script file containing R code for a Decision Tree model. The code calculates the confusion matrix and various performance metrics. The console shows the output of these calculations.

```
101 tree_pred <- predict(tree_model, test_data, type = "class")
102
103 # Confusion Matrix and Accuracy for Decision Tree
104 tree_cm <- table(test_data$loan_status, tree_pred)
105 cat("Decision Tree Confusion Matrix:\n")
106 print(tree_cm)
107 tree_accuracy <- sum(diag(tree_cm)) / sum(tree_cm)
108 cat("Decision Tree Accuracy:", tree_accuracy, "\n")
109
110 tabl = tree_cm
111 error = 1 - sum(diag(tabl)) / sum(tabl)
112 error
113 #####3
114 precision = tabl[1]/(tabl[1] + tabl[2])
115 precision
116 recall = tabl[1]/ (tabl[1]+tabl[3])
117 recall
118 f1 = (2 * precision * recall) / (precision + recall)
```

Console output:

```
R 4.3.2 - D:/Studies/ R/
[1] 0.8955224
> cat("Decision Tree Accuracy:", tree_accuracy, "\n")
Decision Tree Accuracy: 0.9239991
> tabl = tree_cm
> error = 1 - sum(diag(tabl)) / sum(tabl)
> error
[1] 0.07600093
> #####3
> precision = tabl[1]/(tabl[1] + tabl[2])
> precision
[1] 0.9152774
> recall = tabl[1]/ (tabl[1]+tabl[3])
> recall
[1] 0.9950973
> f1 = (2 * precision * recall) / (precision + recall)
> f1
[1] 0.9535198
>
```

This screenshot shows the RStudio interface with a script file containing R code for a Decision Tree model. The code calculates the confusion matrix and various performance metrics. The console shows the output of these calculations. The Environment pane shows the variables created. The Plots pane shows a decision tree plot.

```
97 #####3
98 # Decision Tree Model
99 tree_model <- rpart(loan_status ~ ., data = train_data, method = "class")
100 rpart.plot(tree_model)
101 tree_pred <- predict(tree_model, test_data, type = "class")
102
103 # Confusion Matrix and Accuracy for Decision Tree
104 tree_cm <- table(test_data$loan_status, tree_pred)
105 cat("Decision Tree Confusion Matrix:\n")
106 print(tree_cm)
107 tree_accuracy <- sum(diag(tree_cm)) / sum(tree_cm)
108 cat("Decision Tree Accuracy:", tree_accuracy, "\n")
109
110 tabl = tree_cm
111 error = 1 - sum(diag(tabl)) / sum(tabl)
112 error
113 #####3
114 precision = tabl[1]/(tabl[1] + tabl[2])
115 precision
116 recall = tabl[1]/ (tabl[1]+tabl[3])
117 recall
118 f1 = (2 * precision * recall) / (precision + recall)
119 f1
120 # Decision Tree Model
121 tree_model <- rpart(loan_status ~ ., data = train_data, method = "class")
122 rpart.plot(tree_model)
123 tree_pred <- predict(tree_model, test_data, type = "class")
124 # Confusion Matrix and Accuracy for Decision Tree
125 tree_cm <- table(test_data$loan_status, tree_pred)
```

Console output:

```
R 4.3.2 - D:/Studies/ R/
> #####3
[1] 0.8955224
> cat("Decision Tree Accuracy:", tree_accuracy, "\n")
Decision Tree Accuracy: 0.9239991
> tabl = tree_cm
> error = 1 - sum(diag(tabl)) / sum(tabl)
> error
[1] 0.07600093
> #####3
> precision = tabl[1]/(tabl[1] + tabl[2])
> precision
[1] 0.9152774
> recall = tabl[1]/ (tabl[1]+tabl[3])
> recall
[1] 0.9950973
> f1 = (2 * precision * recall) / (precision + recall)
> f1
[1] 0.9535198
> cat("SVM Accuracy:", svm_accuracy, "\n")
```

Environment pane:

```
recal1 = tabl[1]/ (tabl[1]+tabl[3])
recal1
f1 = (2 * precision * recall) / (precision + recall)
f1
# Decision Tree Model
tree_model <- rpart(loan_status ~ ., data = train_data, method = "class")
rpart.plot(tree_model)
tree_pred <- predict(tree_model, test_data, type = "class")
# Confusion Matrix and Accuracy for Decision Tree
tree_cm <- table(test_data$loan_status, tree_pred)
```

Plots pane:

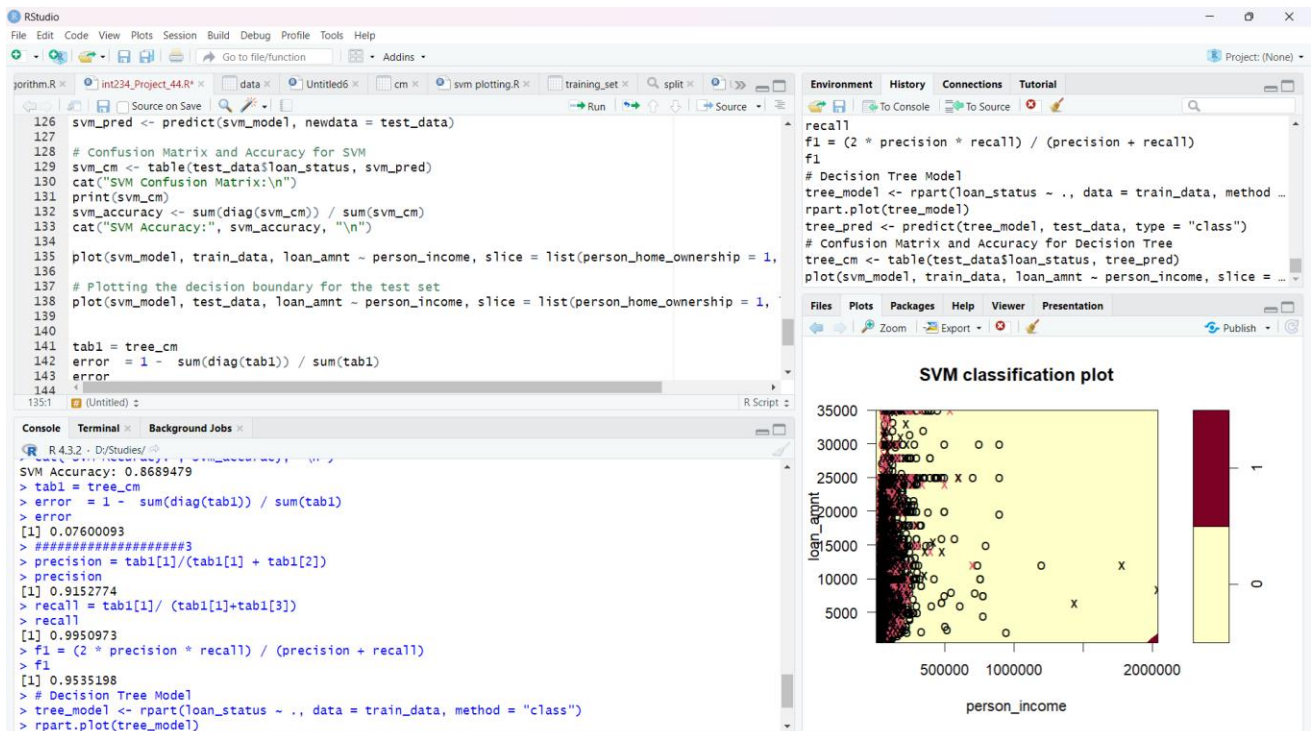
The decision tree plot shows the structure of the model. The root node splits on 'loan\_status'. The left branch splits on 'loan\_status', and the right branch splits on 'loan\_status'. The leaf nodes contain the predicted class for each branch.

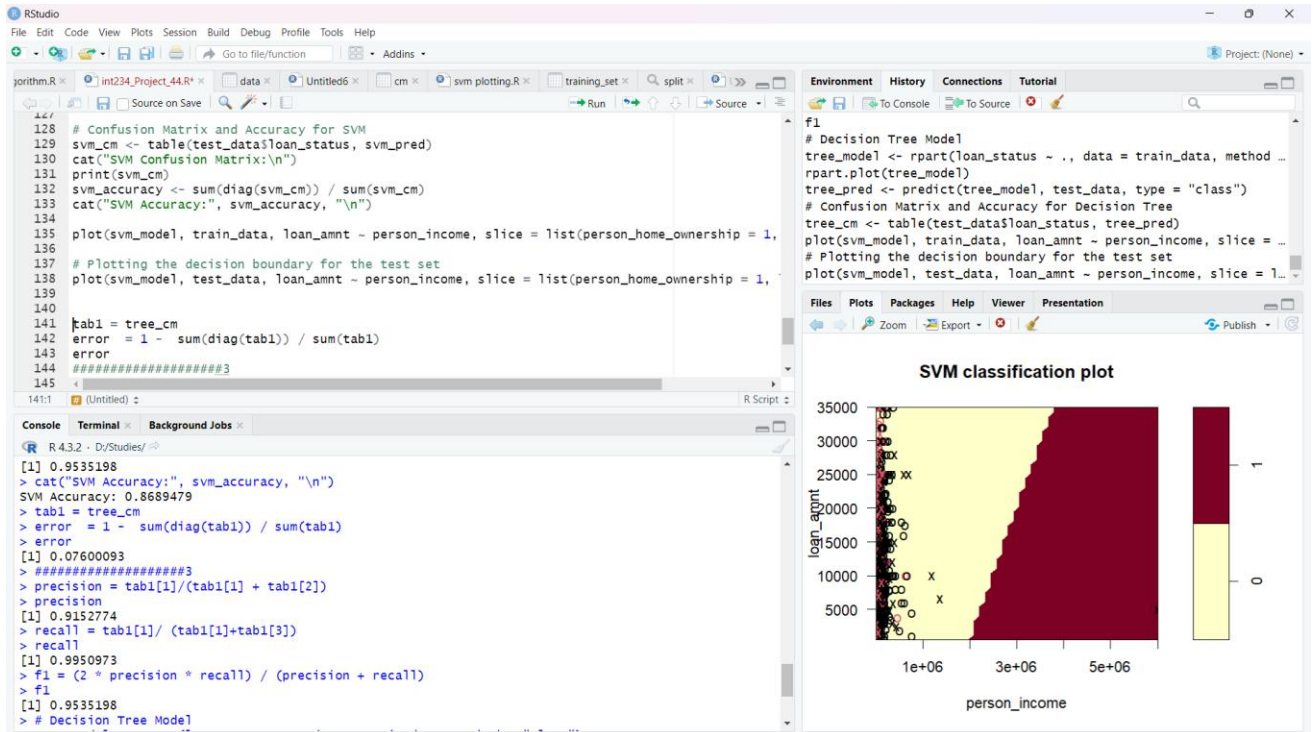
# SVM Output

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Project: (None)
Environment History Conn...
To Console To Source
Files Plots Packages Help
Zoom Export

137 # Plotting the decision boundary for the test set
138 plot(svm_model, test_data, loan_amnt ~ person_income, slice = list(person_home_ownership = 1, loan_grade = 1))
139
140 |
141 tab1 = tree_cm
142 error = 1 - sum(diag(tab1)) / sum(tab1)
143 error
144 #####3
145 precision = tab1[1]/(tab1[1] + tab1[2])
146 precision
147 recall = tab1[1]/ (tab1[1]+tab1[3])
148 recall
149 f1 = (2 * precision * recall) / (precision + recall)
150 f1
151 #####
152
153
154

Console Terminal Background Jobs
R 4.3.2 - D:/Studies/
[1] 0.9535198
> cat("SVM Accuracy:", svm_accuracy, "\n")
SVM Accuracy: 0.8689479
> tab1 = tree_cm
> error = 1 - sum(diag(tab1)) / sum(tab1)
> error
[1] 0.07600093
> #####3
> precision = tab1[1]/(tab1[1] + tab1[2])
> precision
[1] 0.9152774
> recall = tab1[1]/ (tab1[1]+tab1[3])
> recall
[1] 0.9950973
> f1 = (2 * precision * recall) / (precision + recall)
> f1
[1] 0.9535198
>
```







## Conclusion :

Based on the performance metrics of Accuracy, Error Rate, Precision, Recall, and F1 Score for each algorithm, we can draw the following conclusions:

### 1. **Decision Tree Model:**

With an accuracy of 0.92 and an error rate of 0.08, the Decision Tree performs the best overall in terms of accuracy.

It also has the highest recall (1.00) and F1 Score (0.95), indicating that it is very effective at correctly identifying positive cases with minimal false negatives.

This suggests that the Decision Tree model is well-suited for applications where capturing all positive instances is critical.

### 2. **SVM Model:**

SVM has a high accuracy of 0.87 and an error rate of 0.08, similar to the Decision Tree.

Its precision and recall are also strong (0.92 and 1.00), and its F1 Score of 0.95 indicates good balance, comparable to the Decision Tree.

SVM could be a reliable choice for balanced performance, especially when interpretability and generalization are also considerations.

### 3. **Naive Bayes Model:**

With an accuracy of 0.84 and an error rate of 0.16, Naive Bayes performs reasonably well but has slightly lower recall (0.88) and F1 Score (0.87) compared to Decision Tree and SVM.

It may work effectively in scenarios where computational efficiency is essential, even though it may miss some positive cases compared to the higher-performing models.

### 4. **K-Nearest Neighbors (KNN) Model:**

KNN also has an accuracy of 0.84 and an error rate of 0.16, with slightly higher precision (0.86) and recall (0.94) than Naive Bayes.

Its F1 Score of 0.90 indicates it balances precision and recall better than Naive Bayes, making it a potential alternative when interpretability or simpler assumptions are desirable.

## Overall Conclusion:

The **Decision Tree** and **SVM** models stand out as the top performers, with Decision Tree marginally leading in terms of accuracy and recall. For scenarios where accuracy, recall, and interpretability are crucial, **Decision Tree** is likely the best option. However, **SVM** is also a strong choice, particularly if a non-linear boundary might improve performance on further data exploration. For scenarios that prioritize simplicity or efficiency, **KNN** and **Naive Bayes** can be considered, though they are slightly less effective in capturing positive cases compared to Decision Tree and SVM.



