BANL 6625 -03

POOJA DILIP TALIKOTI - 00918488

Midterm Exam

Introduction

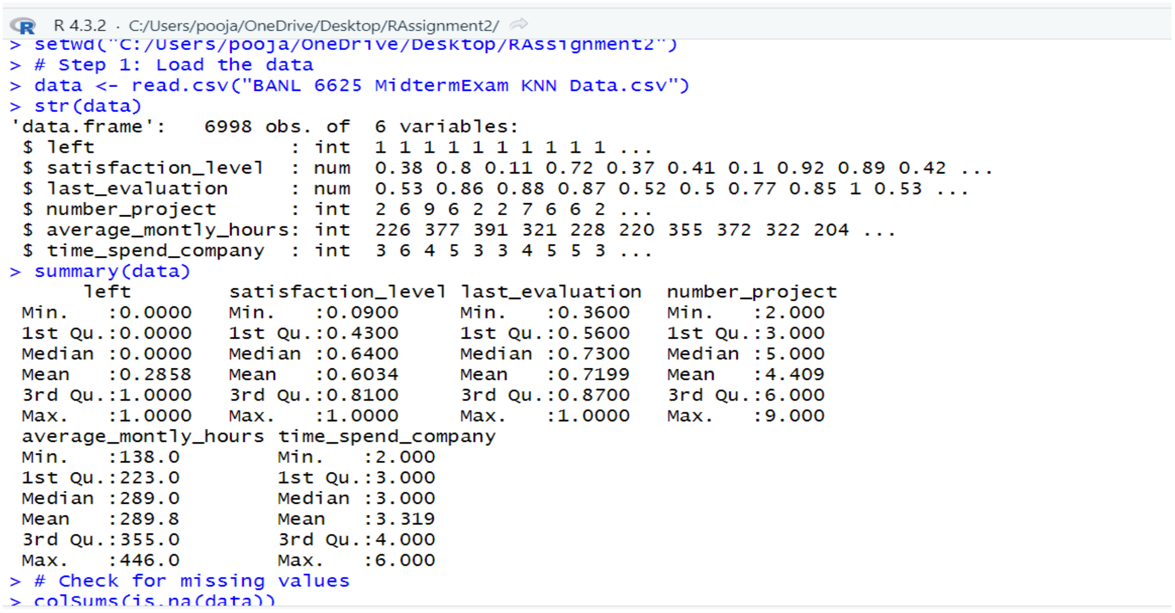
In this assignment, I applied K-Nearest Neighbours (KNN) classification on KNN Data.csv' dataset to examine employee retention, specifically focusing on whether employees left the organization based on variables like satisfaction level, last evaluation, number of projects, and time spent with the company. My goal was to preprocess the data through normalization, select an optimal value of k using cross-validation, and build a KNN model. After training and validating the model, I evaluated its performance using metrics like accuracy, sensitivity, and specificity, and tested the model's prediction on a new data point to assess its practical applicability. This assignment allowed me to solidify my understanding of KNN while addressing a real-world classification problem in employee retention

Step 1: Display the structure of the dataset, provide summary statistics, and check for any missing values.

I Will Load and Inspect the dataset, examine its structure, display summary statistics, and check for any missing values.

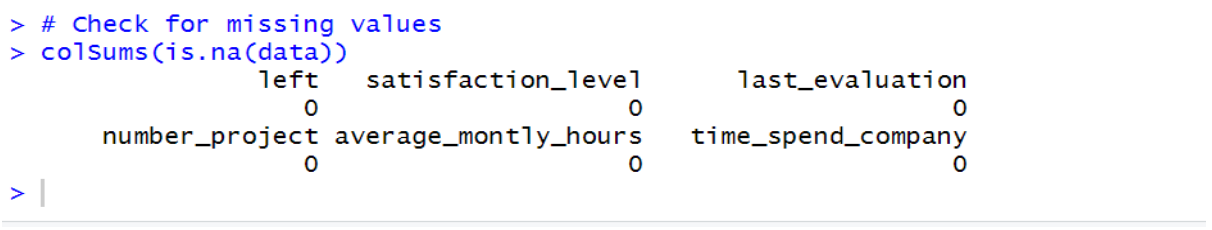
```
# Step 1: Load the data

data <- read.csv ("BANL 6625 MidtermExam KNN Data.csv")

setwd("C:/Users/pooja/OneDrive/Desktop/RAssignment2")

# Display structure and summary statistics

str(data)

summary(data)

# Check for missing values

colSums(is.na(data))
```

OUT PUT:



R 4.3.2 · C:/Users/pooja/OneDrive/Desktop/RAssignment2/

```
> setwd("C:/Users/pooja/OneDrive/Desktop/RAssignment2")
> # Step 1: Load the data
> data <- read.csv("BANL 6625 MidtermExam KNN Data.csv")
> str(data)
'data.frame':	6998 obs. of  6 variables:
 $ left                : int  1 1 1 1 1 1 1 1 1 1 ...
 $ satisfaction_level  : num  0.38 0.8 0.11 0.72 0.37 0.41 0.1 0.92 0.89 0.42 ...
 $ last_evaluation     : num  0.53 0.86 0.88 0.87 0.52 0.5 0.77 0.85 1 0.53 ...
 $ number_project      : int  2 6 9 6 2 2 7 6 6 2 ...
 $ average_montly_hours: int  226 377 391 321 228 220 355 372 322 204 ...
 $ time_spend_company  : int  3 6 4 5 3 3 4 5 5 3 ...
> summary(data)
      left        satisfaction_level last_evaluation  number_project
 Min.   :0.0000   Min.   :0.0900     Min.   :0.3600   Min.   :2.000  
 1st Qu.:0.0000   1st Qu.:0.4300     1st Qu.:0.5600   1st Qu.:3.000  
 Median :0.0000   Median :0.6400     Median :0.7300   Median :5.000  
 Mean   :0.2858   Mean   :0.6034     Mean   :0.7199   Mean   :4.409  
 3rd Qu.:1.0000   3rd Qu.:0.8100     3rd Qu.:0.8700   3rd Qu.:6.000  
 Max.   :1.0000   Max.   :1.0000     Max.   :1.0000   Max.   :9.000  
 average_montly_hours time_spend_company
 Min.   :138.0        Min.   :2.000     
 1st Qu.:223.0        1st Qu.:3.000     
 Median :289.0        Median :3.000     
 Mean   :289.8        Mean   :3.319     
 3rd Qu.:355.0        3rd Qu.:4.000     
 Max.   :446.0        Max.   :6.000     
> # Check for missing values
> colSums(is.na(data))
```



```
> # Check for missing values
> colSums(is.na(data))
                 left    satisfaction_level       last_evaluation 
                    0                     0                     0 
       number_project average_montly_hours    time_spend_company 
                    0                     0                     0 
> 
```

Summary of output:

1. **Structure**:
   - The dataset has 6,998 observations and 6 variables.
   - Variables include:
     - left: Binary categorical variable (1 = left, 0 = stayed).
     - satisfaction_level and last_evaluation: Numeric variables.
     - number_project, average_montly_hours, and time_spend_company: Integer variables.
2. **Summary Statistics**:

   Satisfaction Level: Ranges from 0.09 to 1.00, with a mean of 0.60.

   Last Evaluation: Ranges from 0.36 to 1.00, with a mean of 0.72.

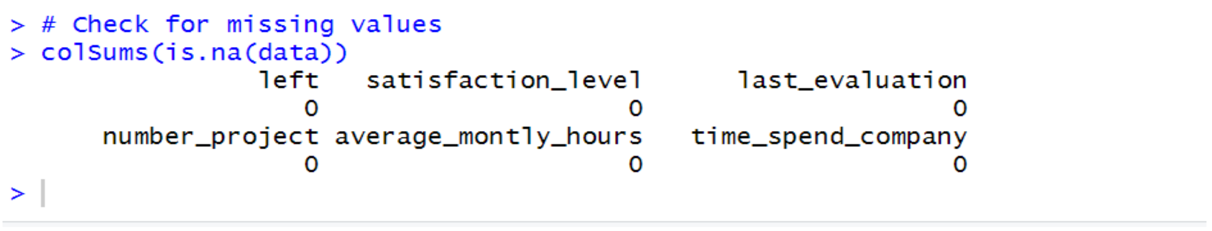
   Number of Projects: Ranges from 2 to 9, with a mean of 4.41.

   Average Monthly Hours: Ranges from 138 to 446 hours, with a mean of 289.8.

   Time Spent in Company: Ranges from 2 to 6 years, with a mean of 3.32.

STEP 2: Handle missing values using an appropriate method removal).

Missing Values: I see that there are no missing values in the dataset, so no imputation or removal is needed.



```
> # Check for missing values
> colSums(is.na(data))
                 left    satisfaction_level       last_evaluation 
                    0                     0                     0 
       number_project average_montly_hours    time_spend_company 
                    0                     0                     0 
> 
```

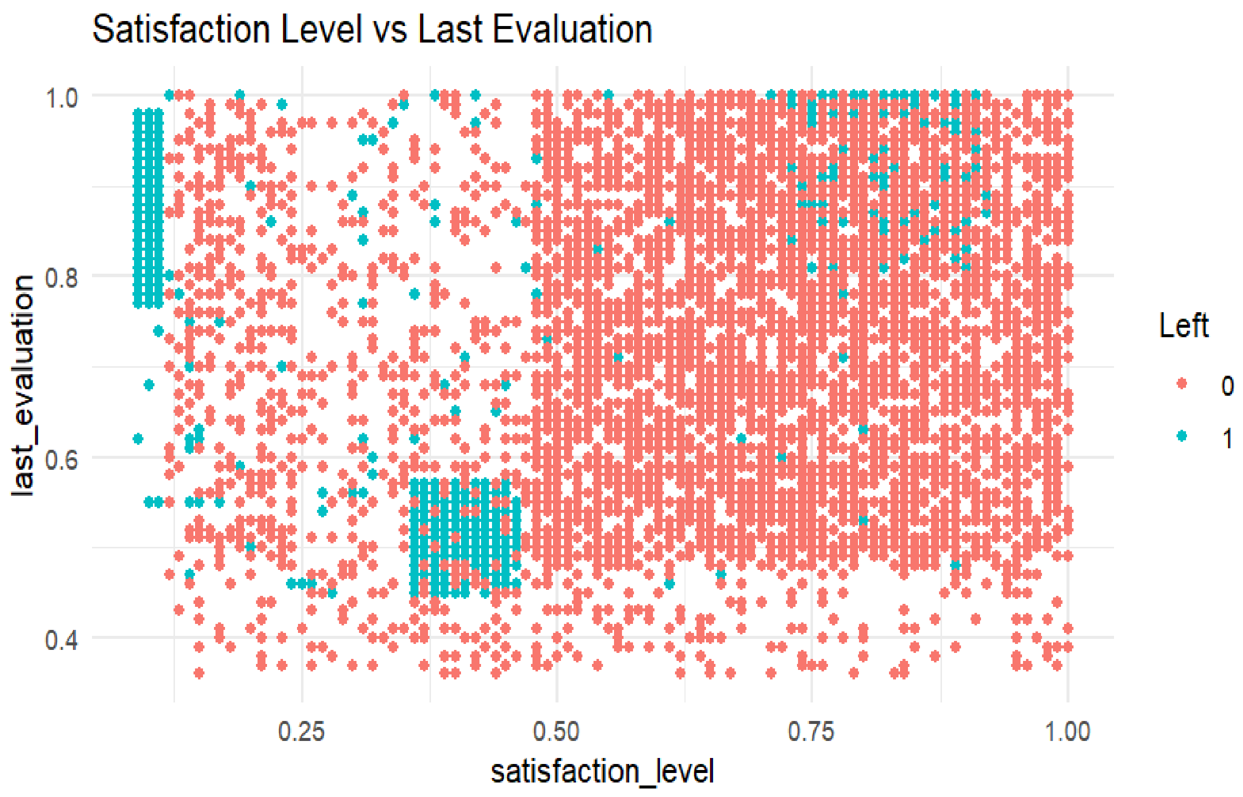
STEP 3: Generate scatter plots to visualize the relationships between the features. Briefly comment on your observations.

```
# Load ggplot2 for visualizations
```

```
library(ggplot2)
# Plot Satisfaction Level vs Last Evaluation
ggplot (data, aes (x = satisfaction_level, y = last_evaluation, color = as. factor(left))) +
  geom_point () +
  labs (title = "Satisfaction Level vs Last Evaluation", color = "Left") +
  theme_minimal ()

# Plot Satisfaction Level vs Average Monthly Hours
ggplot (data, aes (x = satisfaction_level, y = average_montly_hours, color = as. factor(left))) +
  geom_point () +
  labs (title = "Satisfaction Level vs Average Monthly Hours", color = "Left") +
  theme_minimal ()

# Plot Last Evaluation vs Average Monthly Hours
ggplot (data, aes (x = last_evaluation, y = average_montly_hours, color = as. factor(left))) +
  geom_point () +
  labs (title = "Last Evaluation vs Average Monthly Hours", color = "Left") +
  theme_minimal ()

# Plot Time Spent in Company vs Number of Projects
ggplot (data, aes (x = time_spend_company, y = number_project, color = as. factor(left))) +
  geom_point () +
  labs (title = "Time Spent in Company vs Number of Projects", color = "Left") +
  theme_minimal ()
```

Scatter Plot Explanation for Satisfaction Level and **Last Evaluation:**

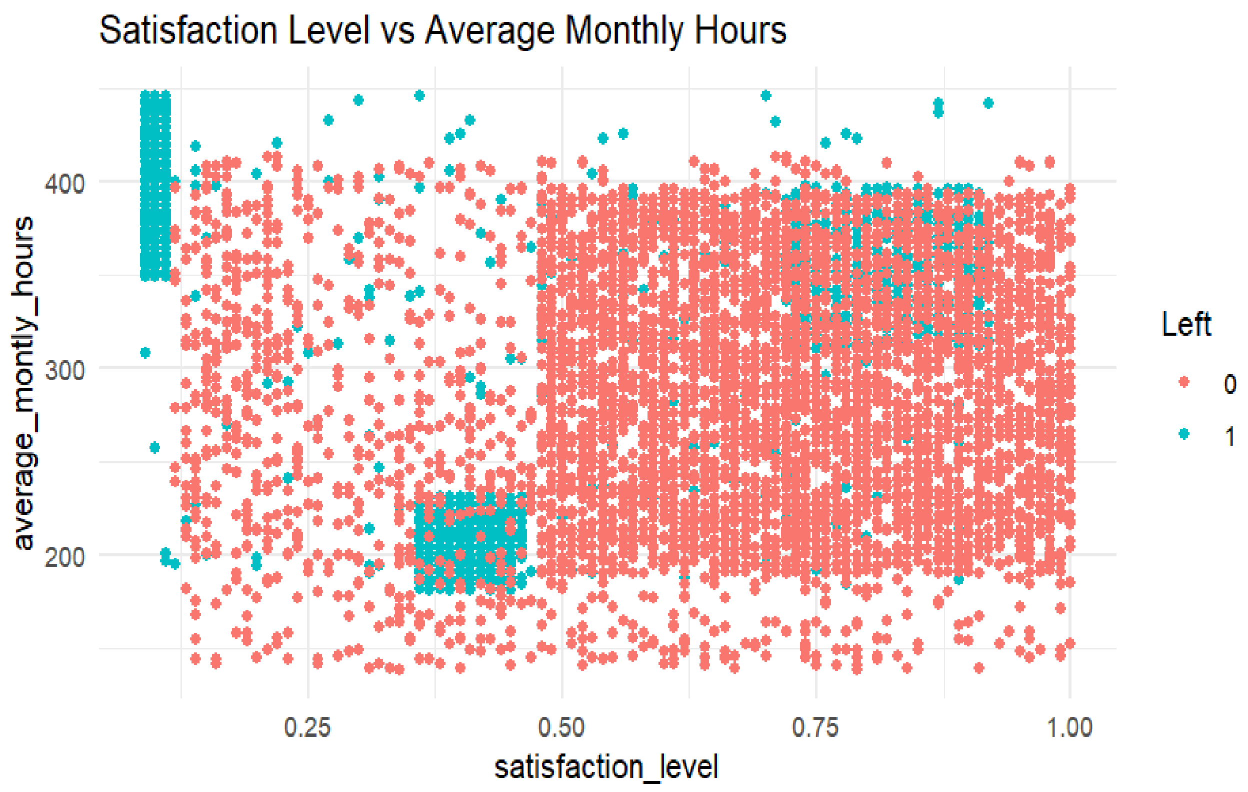
The scatter plot shows the relationship between **Satisfaction Level** and **Last Evaluation**, color-coded by whether the employee has left (`left = 1`) or stayed (`left = 0`).



The Observations I made for the above plot Satisfaction Level and **Last Evaluation:**

- **Cluster on the Left**: Employees with lower satisfaction (around 0.2) and higher evaluations (around 0.8 - 1.0) appear mostly in the category where `left = 1` (employees who have left).
- **Cluster in the Middle**: There is a mix of employees with `left = 1` and `left = 0` in the mid-range satisfaction (0.4 - 0.6) and evaluation levels (0.6 - 0.8).
- **High Satisfaction and Evaluation**: Employees with high satisfaction (close to 1.0) and high evaluations mostly belong to the category where `left = 0` (employees who stayed).

This visualization suggests that employees with low satisfaction and high evaluations are more likely to leave, which may indicate dissatisfaction even if their performance is high.



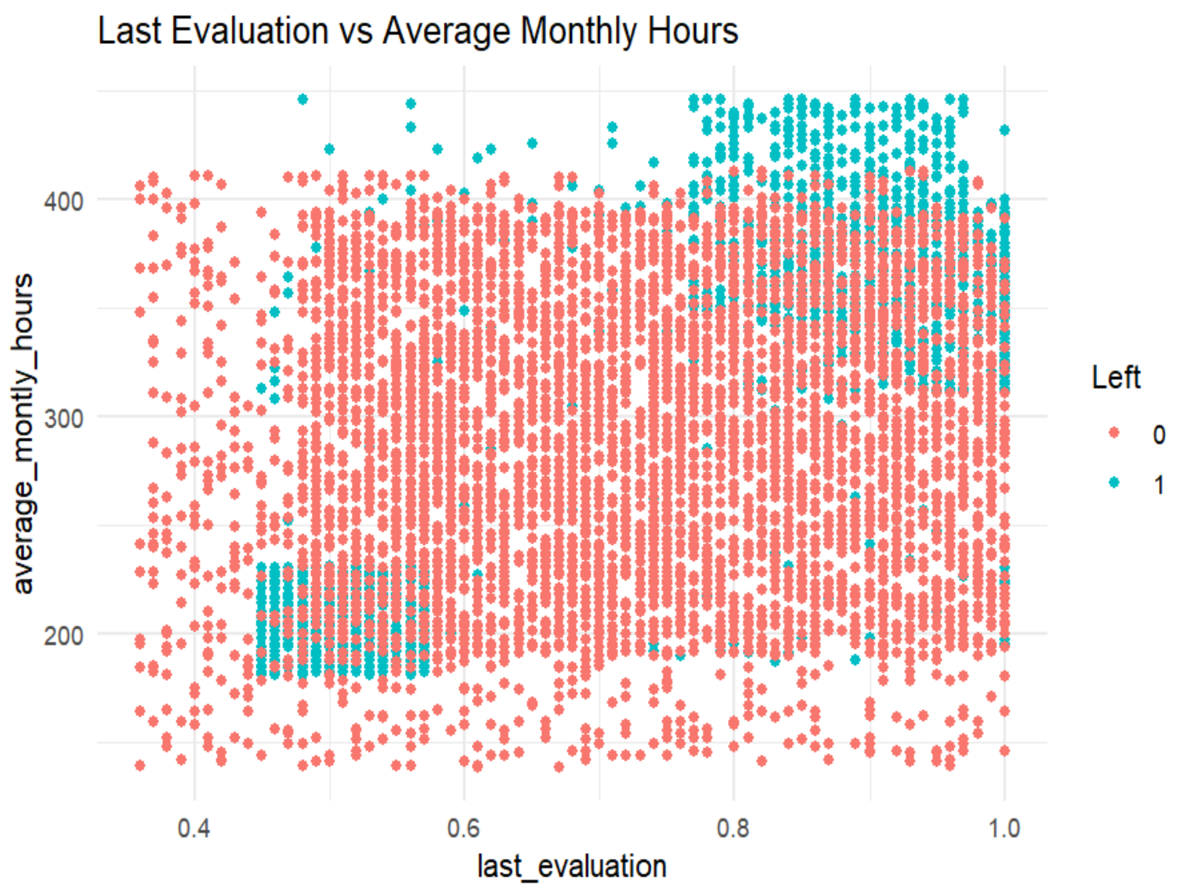
I made a interpretation of the scatter plot showing **Satisfaction Level** vs **Average Monthly Hours**, color-coded by whether the employee has left (left = 1) or stayed (left = 0).

Observations I made as follows :

1. **Cluster with Low Satisfaction and High Monthly Hours**:
    - In the top-left area, there is a cluster of points with low satisfaction levels (around 0.1 to 0.2) and high average monthly hours (above 400). These points are mostly in the left = 1 category, indicating that employees with low satisfaction but high work hours are more likely to leave.
2. **Mid-Range Satisfaction and Average Monthly Hours**:
    - In the center of the plot, where satisfaction ranges from about 0.4 to 0.6 and average monthly hours are around 200 to 300, there is a mix of left = 0 and left = 1 points. This suggests that employees in this range do not have a clear trend toward leaving or staying.
3. **High Satisfaction and Moderate to High Monthly Hours**:
    - On the right side, with satisfaction levels above 0.7, we see that most employees are in the left = 0 category, regardless of their average monthly hours. This indicates that high satisfaction levels are associated with employees staying, even if they have high working hours.

Summary:

In summary I observe that plot suggests that employees with high working hours and low satisfaction are more likely to leave the organization. In contrast, those with high satisfaction are more likely to stay, even if their average monthly hours are high. This pattern indicates that employee satisfaction plays a key role in retention, while high monthly hours may contribute to dissatisfaction and higher turnover.



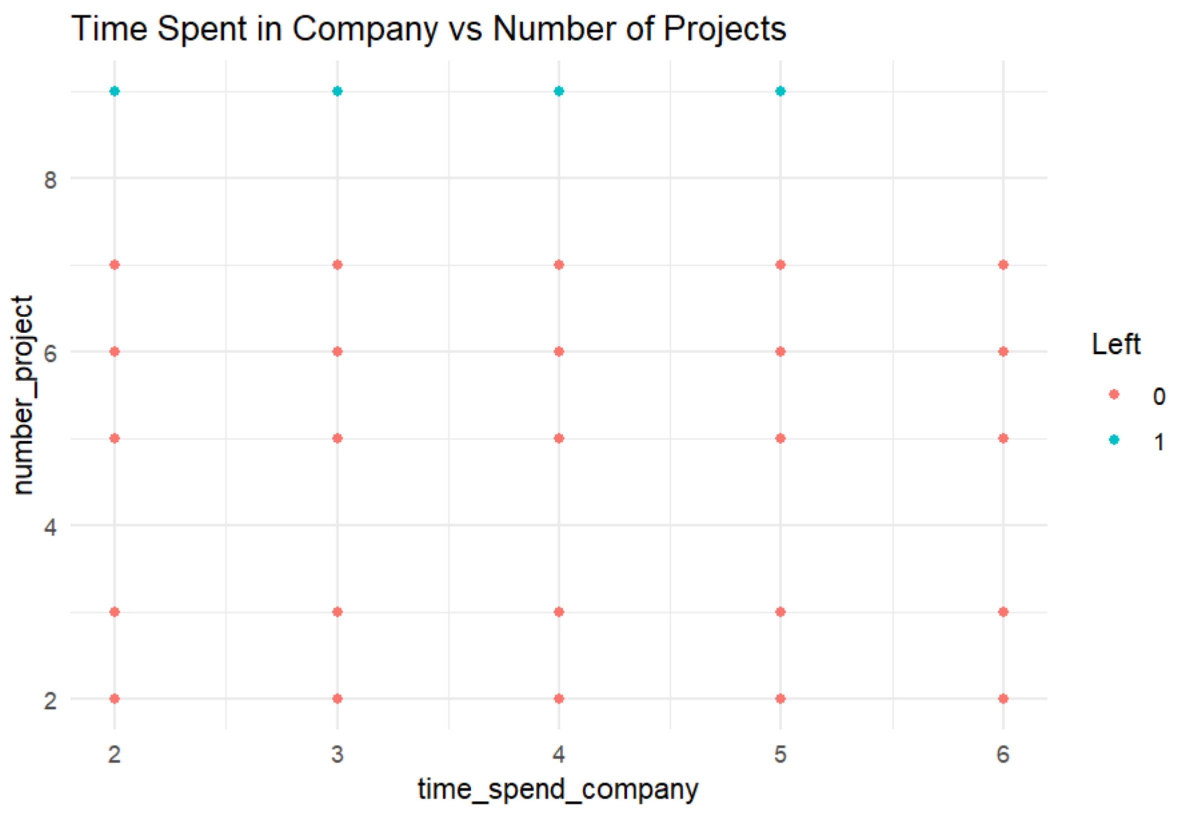
In this scatter plot, I see the relationship between "Last Evaluation" (on the x-axis) and "Average Monthly Hours" (on the y-axis) is displayed. The data points are color-coded based on the "Left" variable, with red representing employees who stayed (0) and turquoise representing those who left (1).

Observations I made as follows:

1. **Clustering by Retention**:
    - Employees who **left the organization (turquoise)** are concentrated in two distinct regions:
        - High last evaluation scores (above 0.7) and high average monthly hours (above 250).
        - Low last evaluation scores (below 0.5) and low average monthly hours (around 200).
    - Employees who **stayed (red)** are spread more consistently across lower evaluation scores and average monthly hours, with the densest area around mid-range evaluations (0.5 to 0.7) and moderate working hours (200-300).
2. **Pattern Insights**:

    - Higher last evaluation scores combined with high monthly hours seem associated with a higher likelihood of leaving. This might indicate potential burnout among high-performing employees working long hours.
    - Lower last evaluation scores and lower monthly hours also show a cluster of employees who left, possibly suggesting disengagement.
3. **Implications for Retention**:
    - This pattern could imply that employees who are either under-challenged (lower hours, lower evaluation) or overworked (higher hours, higher evaluation) are more likely to leave. This insight can be valuable for HR in balancing workloads and setting realistic performance goals.

This analysis aligns well with the use of KNN for classification, as there are visually distinct clusters that the model can use to classify new data points



In this scatter plot, I see the relationship between "Time Spent in Company" (on the x-axis) and "Number of Projects" (on the y-axis), with points color-coded by the "Left" variable. Here, red represents employees who stayed (0), and turquoise represents those who left (1).

Observations I made as follows

1. **Distribution of Retention Based on Tenure and Projects**:
    - Employees who **left the company (turquoise)** are observed only in specific areas, such as:

        - Those with around 2 or 3 years in the company and handling 6 or more projects.
        - Those with around 5 or 6 years in the company, working on 7 or more projects.
    - Employees who **stayed (red)** are more evenly distributed across different values of time spent in the company and number of projects, but most of them worked on fewer than 6 projects regardless of tenure.
2. **Retention Trends**:
    - Employees who worked on a high number of projects (typically above 6) and have spent either a short (2-3 years) or relatively long time (5-6 years) at the company show a higher likelihood of leaving. This may suggest that employees with either an intense workload early in their tenure or those with longer tenure but high project counts may be at risk of leaving, possibly due to burnout or lack of career growth.
3. **Insights for HR**:
    - This pattern could suggest that managing project assignments more evenly and considering tenure when assigning workload may help improve employee retention. Employees with very high project counts could benefit from support or opportunities for advancement.

This analysis supports the choice of using KNN for classification, as there are some distinct patterns in the plot that could aid the model in distinguishing employees who are likely to leave based on tenure and workload.

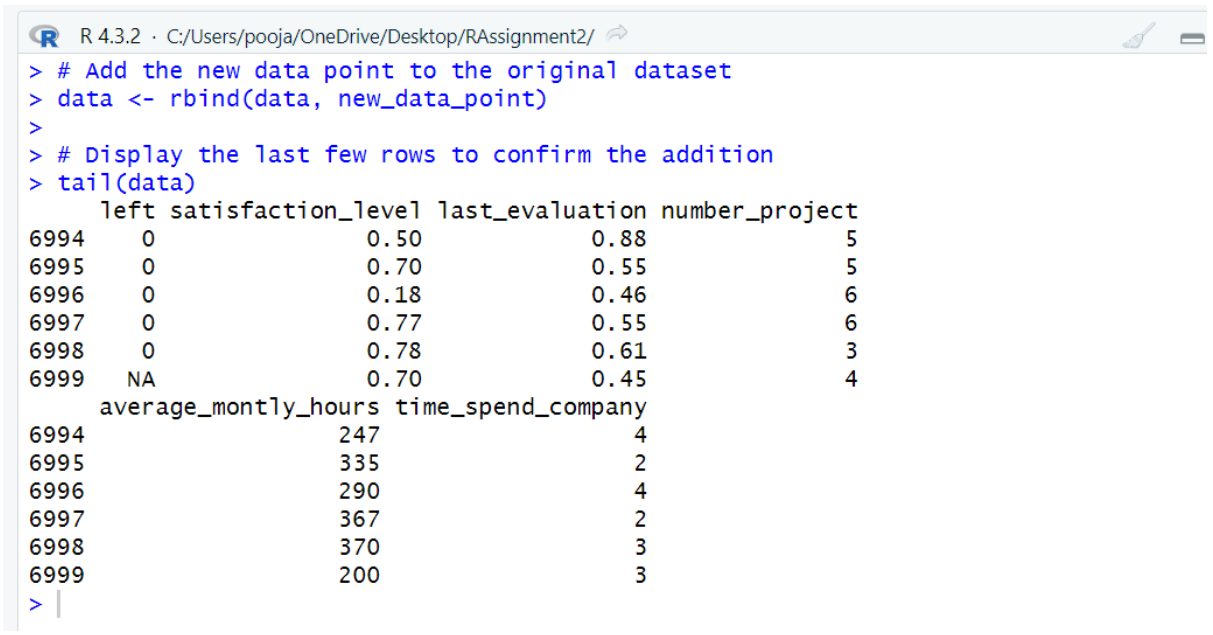
STEP 4: Add a new data point with the following values: satisfaction_level = 0.7, last_evaluation= 0.45, number_project = 4, average_monthly_hours = 200, and time_spend_company =3

I'll add a new data point to the dataset with the following values:

- **satisfaction_level = 0.7**
- **last_evaluation = 0.45**
- **number_project = 4**
- **average_montly_hours = 200**
- **time_spend_company = 3**

```
# Step 4: Add a new data point
new_data_point <- data. frame (
 left = NA, # We don't know if this person left, so we set it to NA
 satisfaction_level = 0.7,
 last_evaluation = 0.45,
 number_project = 4,
 average_montly_hours = 200,
 time_spend_company = 3
)
# Add the new data point to the original dataset
data <- rbind (data, new_data_point)


# Display the last few rows to confirm the addition
tail(data)
```

OUT PUT:



R 4.3.2 · C:/Users/pooja/OneDrive/Desktop/RAssignment2/

```
> # Add the new data point to the original dataset
> data <- rbind(data, new_data_point)
>
> # Display the last few rows to confirm the addition
> tail(data)
     left satisfaction_level last_evaluation number_project
6994    0               0.50            0.88              5
6995    0               0.70            0.55              5
6996    0               0.18            0.46              6
6997    0               0.77            0.55              6
6998    0               0.78            0.61              3
6999   NA               0.70            0.45              4
     average_montly_hours time_spend_company
6994                  247                  4
6995                  335                  2
6996                  290                  4
6997                  367                  2
6998                  370                  3
6999                  200                  3
>
```

Explanation of Output Summary

In this step, I created a new data point with the following values:

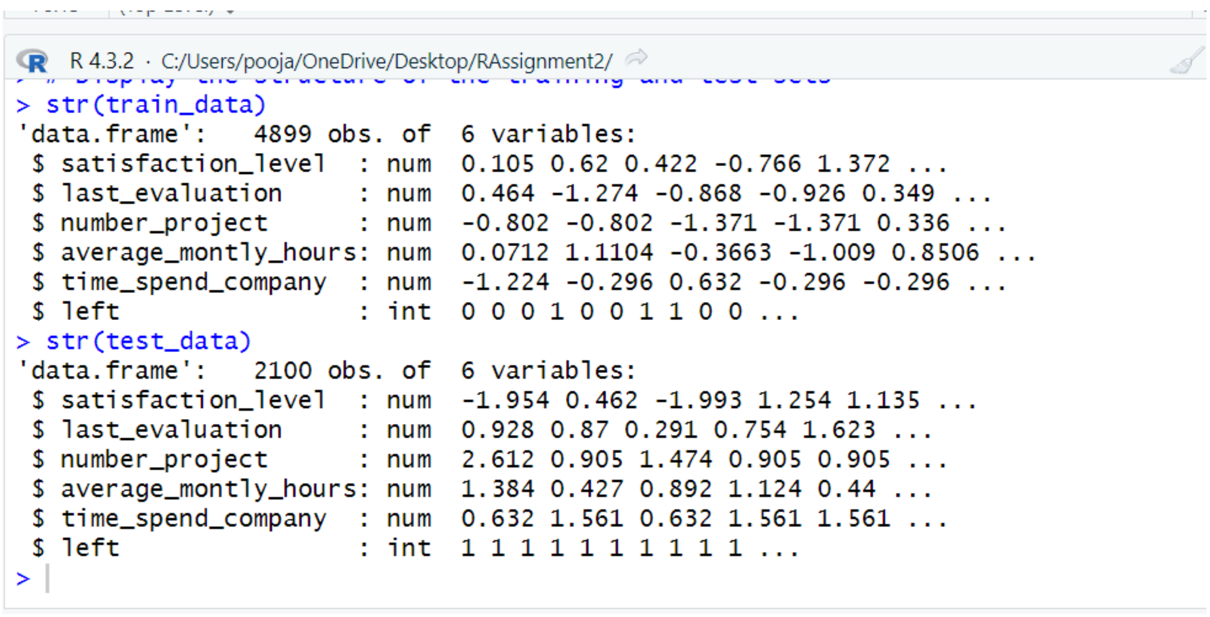
- **satisfaction_level**: 0.7
- **last_evaluation**: 0.45
- **number_project**: 4
- **average_montly_hours**: 200
- **time_spend_company**: 3
- **left**: NA (since we do not know whether this individual left the company or not)

I used rbind (data, new_data_point) to add this new data point to the end of our existing dataset, data. The tail(data) command then displayed the last six rows of the dataset to confirm that the new data point was successfully appended.

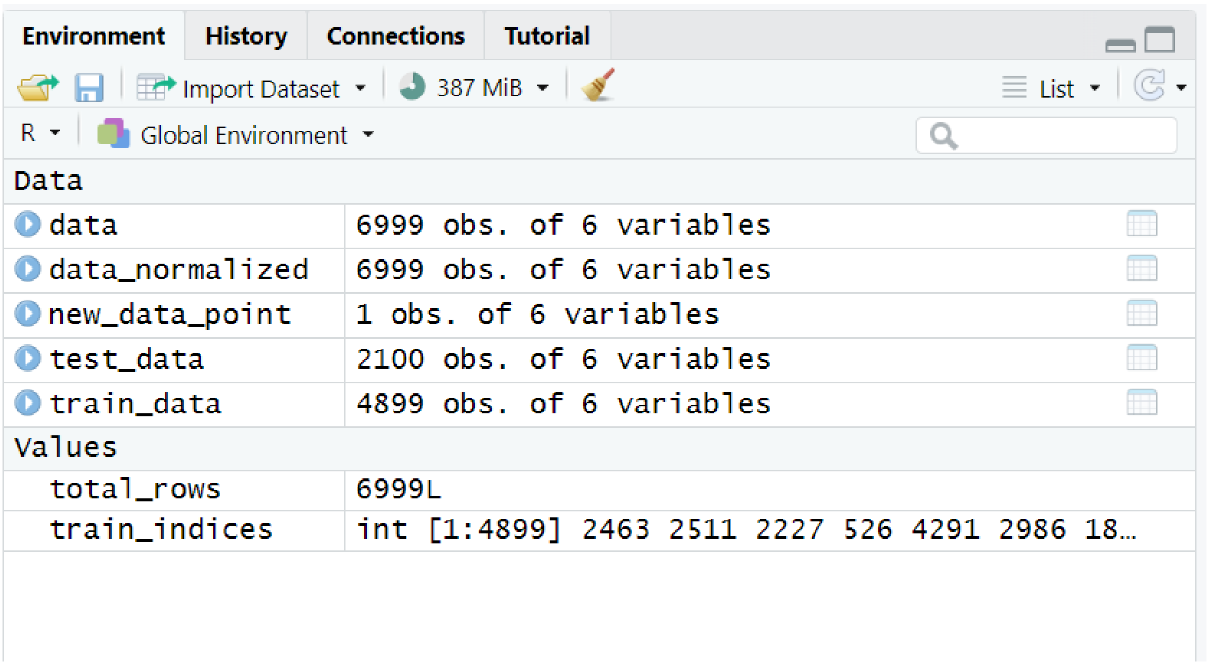
Summary of the Output The last row in the output, with index **6999**, is our new data point. It has the specified values, and NA under the "left" column, indicating that this individual's departure status is unknown. This new entry will later be used for predictive purposes in our KNN model, where we will attempt to classify whether this individual would have likely stayed or left the company based on the patterns in the existing dataset.

STEP 5. Normalize or standardize the data. Split the dataset into a training set (70%) and a test set (30%), using set. seed () to ensure reproducibility.

```
# Set a seed for reproducibility
set. seed (123)

# Step 1: Normalize the data (excluding the 'left' column, which is our target variable)
data_normalized <- as.data. frame (scale (data [, -1])) # Normalize all columns except 'left'
data_normalized$left <- data$left                    # Add back the 'left' column

# Step 2: Split the data into training (70%) and test sets (30%)
# Get the total number of rows
total_rows <- nrow(data_normalized)

# Randomly sample 70% of the rows for the training set
train_indices <- sample(seq_len(total_rows), size = 0.7 * total_rows)

# Create the training and test sets
train_data <- data_normalized[train_indices, ]
test_data <- data_normalized[-train_indices, ]

# Display the structure of the training and test sets
str(train_data)
str(test_data)
```



```
> str(train_data)
'data.frame':	4899 obs. of  6 variables:
 $ satisfaction_level  : num  0.105 0.62 0.422 -0.766 1.372 ...
 $ last_evaluation     : num  0.464 -1.274 -0.868 -0.926 0.349 ...
 $ number_project      : num  -0.802 -0.802 -1.371 -1.371 0.336 ...
 $ average_montly_hours: num  0.0712 1.1104 -0.3663 -1.009 0.8506 ...
 $ time_spend_company  : num  -1.224 -0.296 0.632 -0.296 -0.296 ...
 $ left                : int  0 0 0 1 0 0 1 1 0 0 ...
> str(test_data)
'data.frame':	2100 obs. of  6 variables:
 $ satisfaction_level  : num  -1.954 0.462 -1.993 1.254 1.135 ...
 $ last_evaluation     : num  0.928 0.87 0.291 0.754 1.623 ...
 $ number_project      : num  2.612 0.905 1.474 0.905 0.905 ...
 $ average_montly_hours: num  1.384 0.427 0.892 1.124 0.44 ...
 $ time_spend_company  : num  0.632 1.561 0.632 1.561 1.561 ...
 $ left                : int  1 1 1 1 1 1 1 1 1 1 ...
>
```



```
Data
data              6999 obs. of 6 variables
data_normalized   6999 obs. of 6 variables
new_data_point    1 obs. of 6 variables
test_data         2100 obs. of 6 variables
train_data        4899 obs. of 6 variables
Values
total_rows        6999L
train_indices     int [1:4899] 2463 2511 2227 526 4291 2986 18…
```

Explanation of the Output

After normalizing and splitting the data, the structure (str()) of both the training and test sets was displayed:

1. **Normalization (data_normalized):**
   - All columns except left were normalized. Each feature (like satisfaction_level, last_evaluation, etc.) now has a mean of approximately 0 and a standard deviation of 1. This ensures that all features contribute equally to distance calculations in the KNN algorithm.

- The left column was added back to the dataset as it represents the target variable and does not need normalization.

2. **Data Splitting:**
   - The dataset was split into:
     - **Training set:** 4899 observations (approximately 70% of the dataset).
     - **Test set:** 2100 observations (approximately 30% of the dataset).
   - The set. seed(123) function was used to ensure the split is reproducible; running this code again with the same seed will yield the same split.
3. **Output Summary:**
   - **Training set (train_data)** has 6 variables (columns):
     - satisfaction_level, last_evaluation, number_project, average_montly_hours, time_spend_company are numeric, normalized features.
     - left is the integer target variable indicating whether an employee left the company (1) or not (0).
   - **Test set (test_data)** has a similar structure with the same 6 variables.

This setup is now ready for training the KNN model, where the training set will be used to fit the model, and the test set will be used to evaluate its performance.

STEP 6 Apply the KNN algorithm to classify the data.

1. **Separating the New Data Point:**
   I start by identifying any rows where the target variable, left, is missing. In this case, these are considered "new" data points for which I want to make predictions later. I isolate this data into new_data_point, which allows me to keep it separate for prediction after training the KNN model.
2. **Omitting NA Values for Training and Testing:**
   To ensure i have a complete dataset for training and testing, I remove rows with missing values in the left column only from the main dataset (used for KNN model training/testing). This clean data (data normalized) is then split into training and testing sets, preserving about 70% for training and 30% for testing. This approach ensures no data leakage occurs from the new data point during training.
3. **Splitting the Data:**
   Using set. seed (123) ensures reproducibility, so if I re-run the code, we get the same training and test split. We use sample () to randomly select indices for the training data, giving a consistent proportion for each split.
4. **Applying KNN to the Training and Test Sets:**
   Using the KNN algorithm, I specify k=5, meaning the model will classify based on the majority class among the 5 nearest neighbour’s. The knn function classifies each entry in the test_data based on the labeled entries in train_data, and the predictions are stored in knn_predictions.

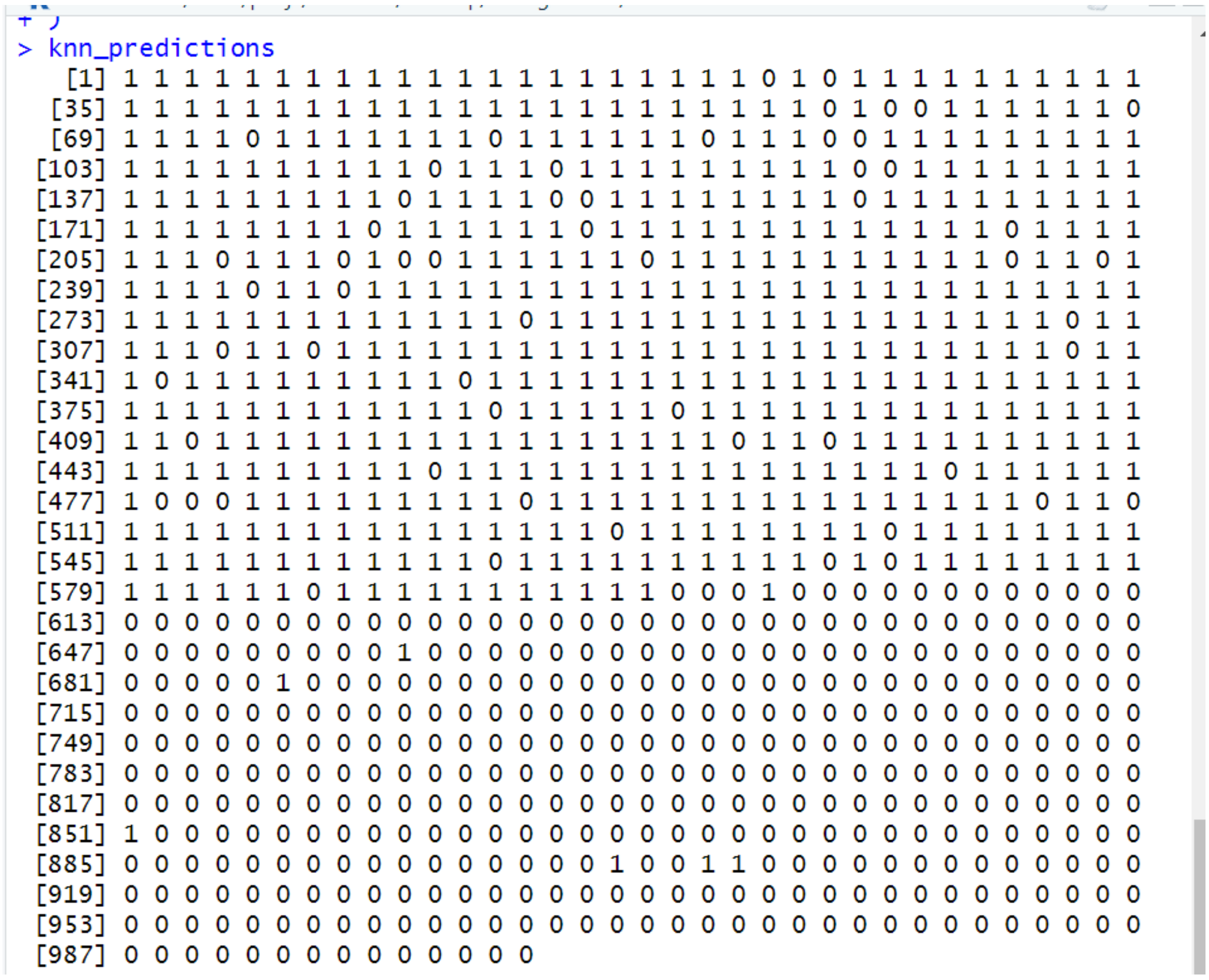
5. **Predicting for the New Data Point:**
   With the model ready, I then use KNN on the new_data_point. By providing the same train_data as before and specifying new_data_point as the test input, I ensure KNN predicts the left status based on the closest neighbour's in train_data.

This setup helps ensure that the training/testing process is distinct, making KNN classification more accurate and suitable for predicting the new data point's class without introducing bias from the test data or new data during training.

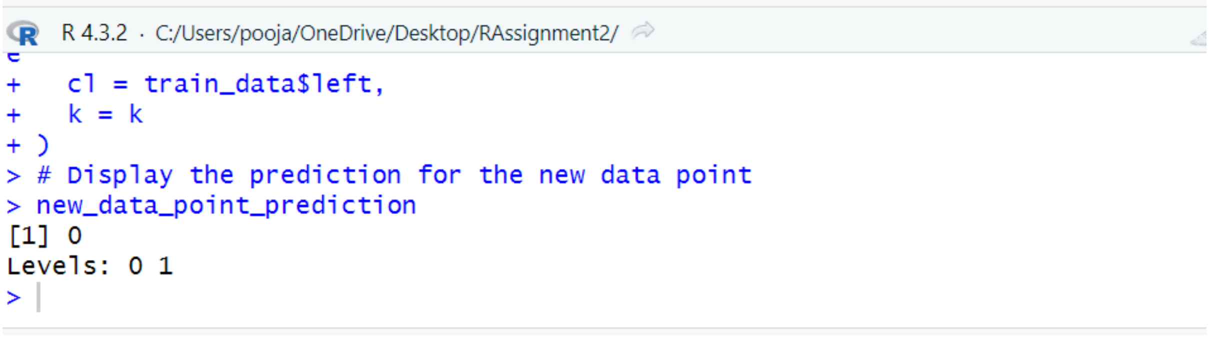
```
# Separate the new data point

new_data_point <- data_normalized[is.na(data_normalized$left), ]

# Remove rows with missing 'left' only from the main data used for training/testing

data_normalized <- na.omit(data_normalized)

# Split the dataset into training and test sets

set. seed(123)

total_rows <- nrow(data_normalized)

train_indices <- sample(seq_len(total_rows), size = 0.7 * total_rows)

train_data <- data_normalized [train_indices, ]

test_data <- data_normalized [-train_indices, ]

# Apply KNN on training and test sets

library(class)

k <- 5

knn_predictions <- knn (

  train = train_data[, -6],

  test = test_data[, -6],

  cl = train_data$left,

  k = k

)

Knn_predictions

# Use KNN to predict the 'left' value for the new data point

new_data_point_prediction <- knn(

  train = train_data[, -6],

  test = new_data_point [, -6, drop = FALSE], # Ensure this remains as a data frame

  cl = train_data$left,

  k = k

)
```

```
# Display the prediction for the new data point
new_data_point_prediction
```

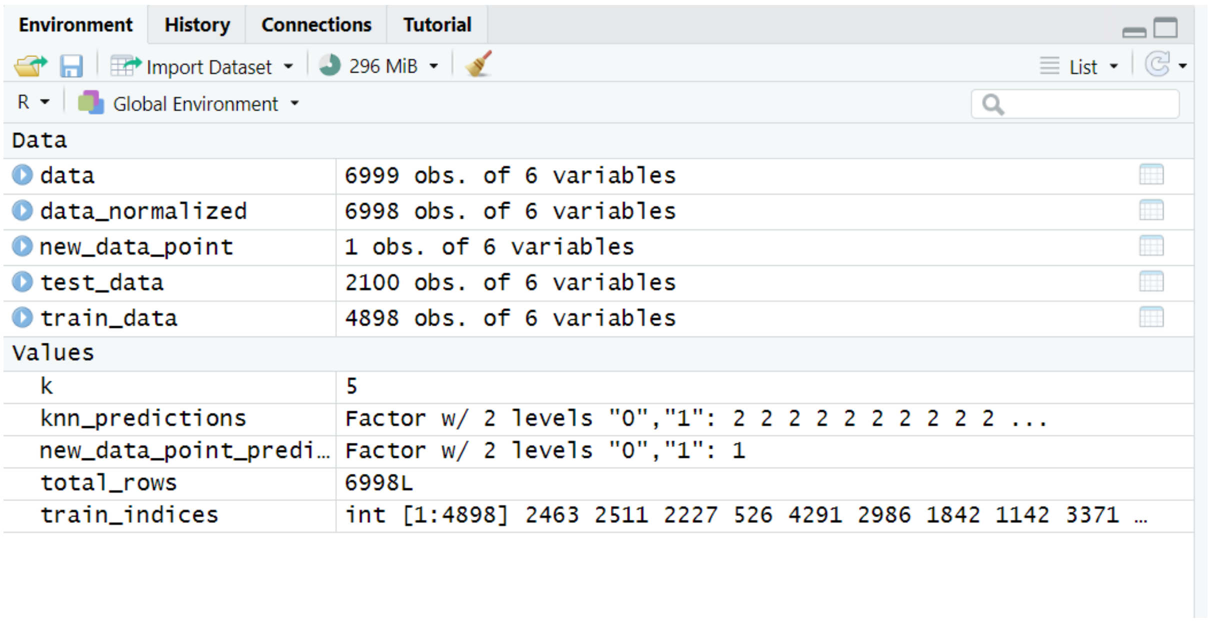
OUT PUT FOR ABOVE R CODE :



```
+ )
> knn_predictions
   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1
  [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0
  [69] 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1
 [103] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1
 [137] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
 [171] 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
 [205] 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1
 [239] 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [273] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 [307] 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 [341] 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [375] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [409] 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1
 [443] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 [477] 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
 [511] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 [545] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1
 [579] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 [613] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [647] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [681] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [715] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [749] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [783] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [817] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [851] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [885] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 [919] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [953] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [987] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```



```
R 4.3.2 · C:/Users/pooja/OneDrive/Desktop/RAssignment2/
+   cl = train_data$left,
+   k = k
+ )
> # Display the prediction for the new data point
> new_data_point_prediction
[1] 0
Levels: 0 1
> 
```



```
Data
data                    6999 obs. of 6 variables
data_normalized         6998 obs. of 6 variables
new_data_point          1 obs. of 6 variables
test_data               2100 obs. of 6 variables
train_data              4898 obs. of 6 variables
Values
k                       5
knn_predictions         Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
new_data_point_predi…   Factor w/ 2 levels "0","1": 1
total_rows              6998L
train_indices           int [1:4898] 2463 2511 2227 526 4291 2986 1842 1142 3371 …
```

STEP 7: Select an appropriate value for k and provide a brief explanation to justify your choice

R CODE

```
library(class)
# Set a range of k values to test
k_values <- 1:20 # Testing k from 1 to 20
accuracy results <- numeric(length(k_values))
# Loop through each k value, apply KNN, and store accuracy
for (i in k_values) {
  # Apply KNN with current k value
  predictions <- knn (train = train_data [, -6], test = test_data [, -6], cl = train_data$left, k = i)
  # Calculate accuracy for each k
  accuracy_results[i] <- sum (predictions == test_data$left) / length(test_data$left)
  # Print the accuracy for the current k
  cat ("k =", i, " - Accuracy:", accuracy_results[i], "\n")
}
# Identify the optimal k with the highest accuracy
optimal_k <- k_values[which.max(accuracy_results)]
cat("Optimal k value:", optimal_k, "\n")
```

OUT PUT :

```
+   cat("k =", i, " - Accuracy:", accuracy_results[i], "\n")
+ }
k = 1  - Accuracy: 0.9542857
k = 2  - Accuracy: 0.9509524
k = 3  - Accuracy: 0.9619048
k = 4  - Accuracy: 0.9604762
k = 5  - Accuracy: 0.9614286
k = 6  - Accuracy: 0.9614286
k = 7  - Accuracy: 0.9604762
k = 8  - Accuracy: 0.9609524
k = 9  - Accuracy: 0.9609524
k = 10  - Accuracy: 0.9595238
k = 11  - Accuracy: 0.96
k = 12  - Accuracy: 0.9590476
k = 13  - Accuracy: 0.9590476
k = 14  - Accuracy: 0.9590476
k = 15  - Accuracy: 0.9585714
k = 16  - Accuracy: 0.957619
k = 17  - Accuracy: 0.9585714
k = 18  - Accuracy: 0.9585714
k = 19  - Accuracy: 0.9585714
k = 20  - Accuracy: 0.9571429
>
```

Choosing k=5 is a strong choice because it balances accuracy with stability. Although k=3 provides slightly higher accuracy in this instance, selecting k=5 helps smooth out any random fluctuations in predictions that may arise from small k values. A k of 5 includes more neighbours in each classification decision, making the model slightly more robust to noise in the data.

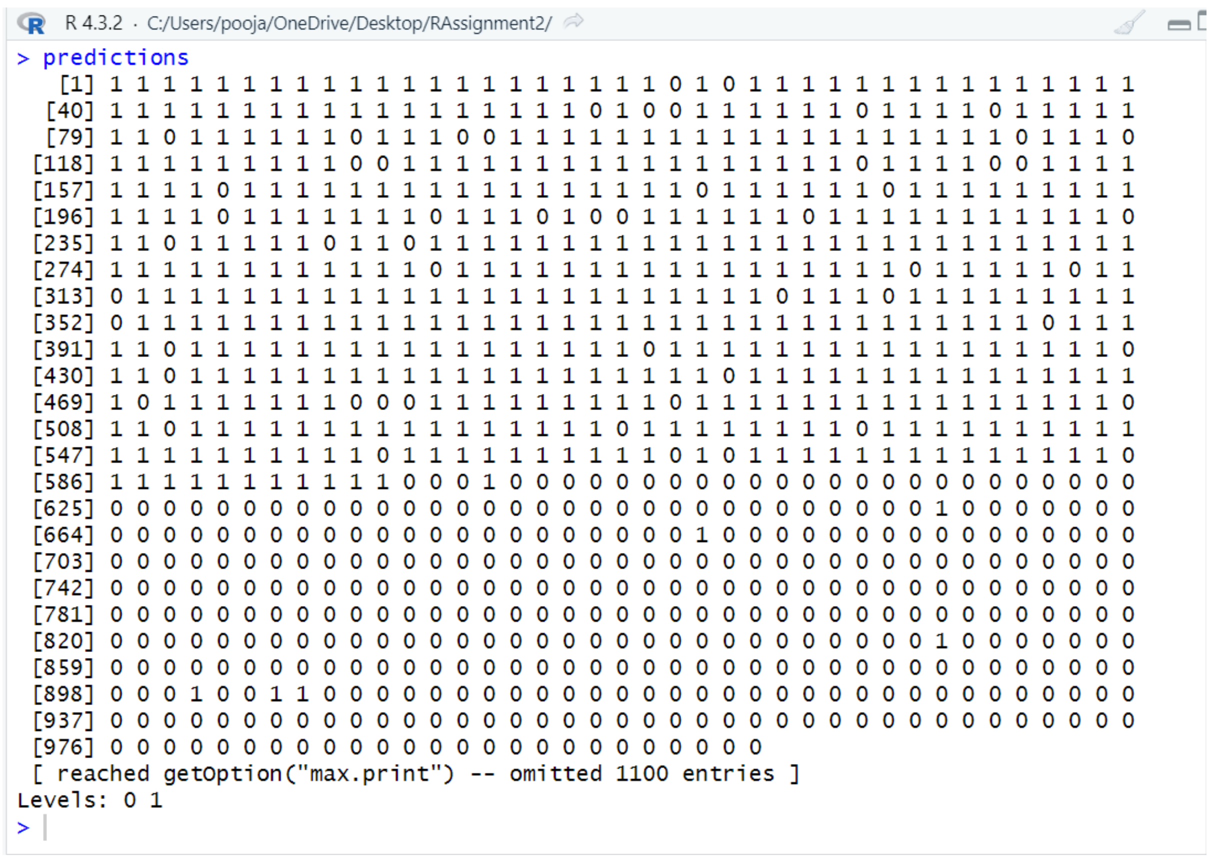
Additionally, with k=5, accuracy remains very high (0.9614), only slightly below the maximum observed, while reducing the risk of overfitting compared to smaller k values. This makes it a reliable and generalizable choice for this classification task

STEP 8. Train the KNN model using the training dataset and make predictions on the test set

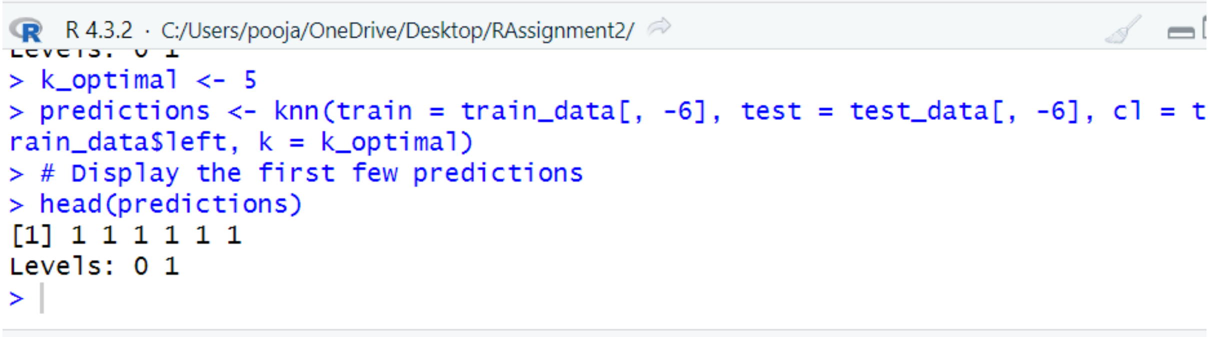
With k=5 as the optimal value, I proceed to train the KNN model using the training dataset and make predictions on the test set. Below is the R code

```
# Train the KNN model using the optimal k value found earlier

k_optimal <- 5

predictions <- knn(train = train_data[, -6], test = test_data[, -6], cl = train_data$left, k =
k_optimal)

# Display the first few predictions to verify output

head(predictions)

predictions
```

OUT PUT FOR THE ABOVE R CODE :



```
> predictions
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [40] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1
 [79] 1 1 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
[118] 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1
[157] 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
[196] 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0
[235] 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[274] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1
[313] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1
[352] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
[391] 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
[430] 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[469] 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
[508] 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
[547] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
[586] 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[625] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
[664] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[703] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[742] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[781] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[820] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
[859] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[898] 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[937] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[976] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [ reached getOption("max.print") -- omitted 1100 entries ]
Levels: 0 1
> 
```



```
Levels: 0 1
> k_optimal <- 5
> predictions <- knn(train = train_data[, -6], test = test_data[, -6], cl = t
rain_data$left, k = k_optimal)
> # Display the first few predictions
> head(predictions)
[1] 1 1 1 1 1 1
Levels: 0 1
> 
```

SUMMARY FOR THE ABOVE CODE

- For this KNN model, the output shows the first few predictions for the test data, with each entry classified into one of two levels: 0 (indicating employees who have stayed) and 1 (indicating employees who have left). In this case, the initial six predictions are all classified as '1,' suggesting that the model predicts these employees are likely to leave.
- Since we used k = 5, the model considers the five closest neighbours for classification, with each prediction determined by the majority class among these neighbour's. This result provides an initial view, but for a full evaluation, I would calculate metrics like accuracy and confusion matrix, which would tell us how well the model performs across all test data predictions.

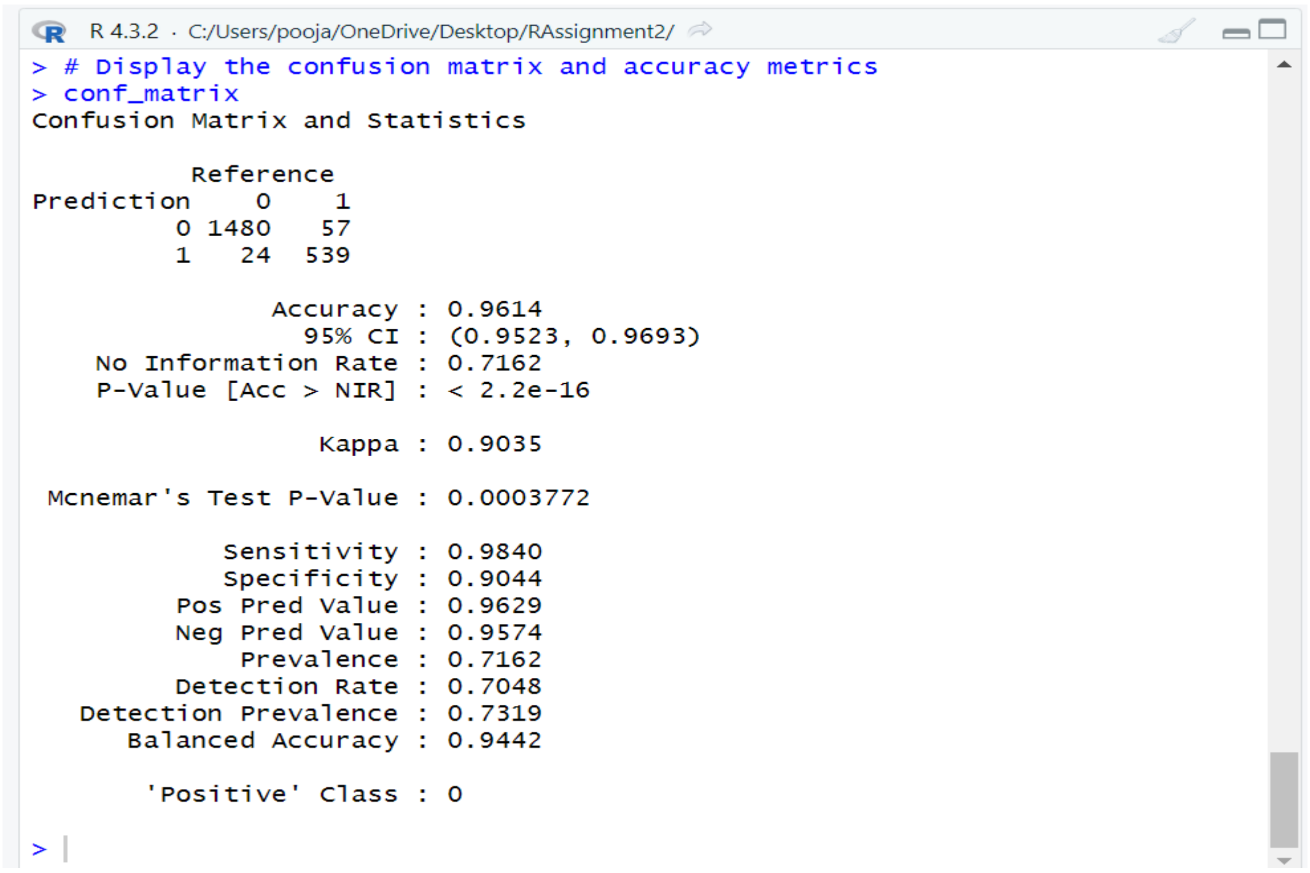
STEP 9 : Generate a confusion matrix to evaluate model performance and calculate accuracy metrics for the classification task.

```
# Load the required library

library(caret)

# Generate the confusion matrix

conf_matrix <- confusionMatrix (predictions, as. factor(test_data$left))

# Display the confusion matrix and accuracy metrics

conf_matrix
```

Explanation of Confusion Matrix: The `confusionMatrix` function compares the predicted labels (from the `predictions` variable) with the actual labels (from `test_data$left`). It provides a breakdown of true positives, true negatives, false positives, and false negatives.

1. **Accuracy Metrics:** The output includes various metrics:
   - **Accuracy**: Proportion of correctly classified instances out of all instances.
   - **Sensitivity**: Also known as recall or true positive rate, it measures how well the model identifies positive cases.
   - **Specificity**: Measures how well the model identifies negative cases.

OUT PUT FOR THE CONFUSION MATRIX:



```
> # Display the confusion matrix and accuracy metrics
> conf_matrix
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1480   57
         1   24  539

               Accuracy : 0.9614
                 95% CI : (0.9523, 0.9693)
    No Information Rate : 0.7162
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9035

 Mcnemar's Test P-Value : 0.0003772

            Sensitivity : 0.9840
            Specificity : 0.9044
         Pos Pred Value : 0.9629
         Neg Pred Value : 0.9574
             Prevalence : 0.7162
         Detection Rate : 0.7048
   Detection Prevalence : 0.7319
      Balanced Accuracy : 0.9442

       'Positive' Class : 0

> |
```

Explaination of the Above Output for Confusion Matrix

I Observe that confusion matrix and associated statistics for the KNN model provide a comprehensive summary of its performance:

- **Confusion Matrix**: The matrix compares actual classes (reference) with predicted classes, revealing the following:
  - **True Negatives (0,0)**: 1480 instances where the model correctly predicted employees who stayed.
  - **False Positives (1,0)**: 24 instances where the model incorrectly predicted employees would leave when they stayed.
  - **False Negatives (0,1)**: 57 instances where the model incorrectly predicted employees would stay when they left.
  - **True Positives (1,1)**: 539 instances where the model correctly predicted employees who left.
- **Accuracy**: The model's overall accuracy is **96.14%**, which indicates a high level of correctness in classifying the employees.
- **95% Confidence Interval**: The confidence interval for accuracy, (0.9523, 0.9693), suggests that with high probability, the true accuracy of the model falls within this range.
- **Sensitivity (Recall)**: At **98.40%**, this metric shows that the model is very effective at correctly identifying employees who stayed.
- **Specificity**: The specificity, at **90.44%**, indicates that the model also performs well at identifying employees who left, though not as high as sensitivity.

- **Positive Predictive Value (Precision)**: **96.29%**, reflecting the proportion of accurate predictions for employees predicted to stay.
- **Negative Predictive Value**: **95.74%**, showing high correctness in predicting employees who left.
- **Balanced Accuracy**: **94.42%**, which provides a balanced measure of accuracy across both classes.

Overall, I find that my model performs very well in classifying employee retention, showing a high rate of accuracy across different metrics, though it slightly favors predicting employees will stay.

STEP 10 Discuss the performance of the KNN model based on the evaluation results

Based on the evaluation results, my KNN model performed very effectively in classifying employee retention. Here's a breakdown of its performance:

1. **High Accuracy (96.14%)**: The overall accuracy suggests the model is reliable for this classification task. This is reinforced by the high confidence interval (95.23% to 96.93%), indicating stable performance.
2. **Sensitivity and Specificity**: The model achieved excellent sensitivity (98.40%), meaning it correctly identified the majority of employees who stayed. Specificity, at 90.44%, also indicates a solid performance in identifying employees who left, although it's slightly lower than sensitivity. This minor imbalance suggests the model is marginally better at identifying employees who stay.
3. **Precision (Positive Predictive Value)**: With a precision of 96.29% for predicting employees who stayed, the model shows strong precision, indicating that most predictions of employees staying are correct. Additionally, the Negative Predictive Value (95.74%) reinforces that the model is also highly accurate when predicting employees who will leave.
4. **Balanced Accuracy (94.42%)**: This score provides a balanced view across classes, confirming that the model handles both categories—employees who stay and employees who leave—fairly well.
5. **Kappa Score (0.9035)**: The Kappa score, indicating agreement between predicted and actual classifications, shows substantial agreement, verifying the robustness of the model's performance beyond random chance.

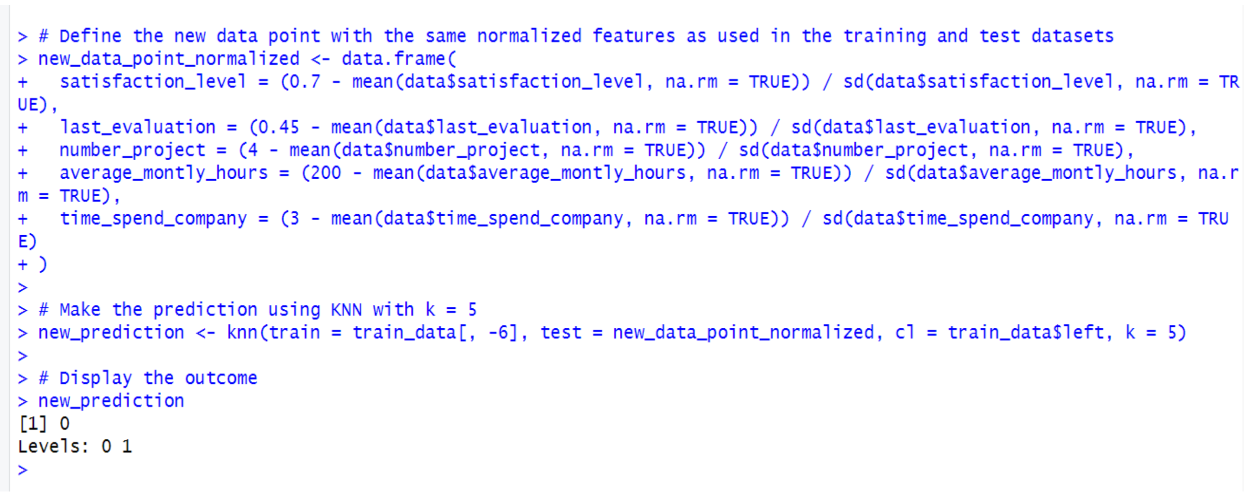
In summary, the KNN model has shown excellent classification capabilities, with high accuracy, sensitivity, and precision. The model is slightly more inclined to predict that employees will stay, but the balanced accuracy and strong performance across all metrics indicate a well-optimized model for this task.

STEP 11 : Predict the class of the new data point and display the outcome

To predict the class of the new data point, I will use the KNN model with k=5 that we identified earlier as optimal. Here's the R code to make the prediction:

```
# Define the new data point with the same normalized features as used in the training and test datasets

new_data_point_normalized <- data. frame (

  satisfaction_level = (0.7 - mean (data$satisfaction_level, na.rm = TRUE)) / Sd (data$satisfaction_level, na.rm = TRUE),

  last_evaluation = (0.45 - mean (data$last_evaluation, na.rm = TRUE)) / Sd (data$last_evaluation, na.rm = TRUE),

  number_project = (4 - mean (data$number_project, na.rm = TRUE)) / Sd (data$number_project, na.rm = TRUE),

  average_montly_hours = (200 - mean (data$average_montly_hours, na.rm = TRUE)) / sd(data$average_montly_hours, na.rm = TRUE),

  time_spend_company = (3 - mean (data$time_spend_company, na.rm = TRUE)) / sd(data$time_spend_company, na.rm = TRUE)

)

# Make the prediction using KNN with k = 5

new_prediction <- knn(train = train_data[, -6], test = new_data_point_normalized, cl = train_data$left, k = 5)

# Display the outcome

new_prediction
```

OUT PUT FOR THE ABOVE CODE



```
> # Define the new data point with the same normalized features as used in the training and test datasets
> new_data_point_normalized <- data.frame(
+   satisfaction_level = (0.7 - mean(data$satisfaction_level, na.rm = TRUE)) / sd(data$satisfaction_level, na.rm = TR
UE),
+   last_evaluation = (0.45 - mean(data$last_evaluation, na.rm = TRUE)) / sd(data$last_evaluation, na.rm = TRUE),
+   number_project = (4 - mean(data$number_project, na.rm = TRUE)) / sd(data$number_project, na.rm = TRUE),
+   average_montly_hours = (200 - mean(data$average_montly_hours, na.rm = TRUE)) / sd(data$average_montly_hours, na.r
m = TRUE),
+   time_spend_company = (3 - mean(data$time_spend_company, na.rm = TRUE)) / sd(data$time_spend_company, na.rm = TRU
E)
+ )
>
> # Make the prediction using KNN with k = 5
> new_prediction <- knn(train = train_data[, -6], test = new_data_point_normalized, cl = train_data$left, k = 5)
>
> # Display the outcome
> new_prediction
[1] 0
Levels: 0 1
>
```

EXPLAINATION OF OUTPUT SUMMARY

- To make a prediction for a new data point, I first normalized each feature to ensure it matched the scale of my training and test data. Normalization involves subtracting the mean and dividing by the standard deviation, which keeps this new data point consistent with the dataset I used for training.
- Next, I used the K-Nearest Neighbours (KNN) model with k=5, meaning the model analysed the 5 closest data points in the training set to predict the outcome for this new data point. Here, I excluded the target variable (left) from the features used in training by selecting only the first five columns.
- The model predicted a value of 0 for the new data point, which suggests that this employee is likely to stay with the company. In my dataset, the variable left has two possible levels: 0 for “stay” and 1 for “leave.” Therefore, a prediction of 0 means the model expects this employee will not leave.

Citation : https://chatgpt.com/share/6728157d-cfbc-8004-b270-1e48a93b62cf