

## **Project Title:** Diabetes prediction using KNN algorithm

### **About Dataset:**

#### **Introduction**

The Diabetes prediction dataset is a collection of medical and demographic data from patients, along with their diabetes status (positive or negative). The data includes features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. This dataset can be used to build machine learning models to predict diabetes in patients based on their medical history and demographic information. This can be useful for healthcare professionals in identifying patients who may be at risk of developing diabetes and in developing personalized treatment plans. Additionally, the dataset can be used by researchers to explore the relationships between various medical and demographic factors and the likelihood of developing diabetes.

#### **Key Features:**

Number of records: 100000

Number of columns: 9

Target Variable : Diabetes

Prediction Accuracy : 97%

**Code:**

```
#Importing Dataset
```

```
mydata<-read.csv("C:/diabetes.csv")
```

```
mydata
```

```
#importing necessary library
```

```
library(class)
```

```
library(ggplot2)
```

```
#Data Summarise
```

```
str(mydata)
```

```
summary(mydata)
```

```
attributes<- names(mydata)
```

```
attributes
```

```
dataType <- c(typeof(mydata$gender), typeof(mydata$age),  
typeof(mydata$hypertension),typeof(mydata$heart_disease),
```

```
typeof(mydata$bmi), typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
```

```
typeof(mydata$diabetes))
```

```
dataType
```

```
head(mydata)
```

```
colSums(is.na(mydata)) #how many instances are missing
```

```
#Data Normalization/scaling
```

```
data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
```

```
mydata[data_norm] <- scale(mydata[data_norm])
```

```
head(mydata)
head(data_norm)
data_norm
mydata[data_norm]
colSums(is.na(mydata[data_norm]))
```

```
# Setting predictor variables and the target variable
```

```
predictor_cols <- names(mydata[data_norm])[ncol(mydata[data_norm])]
target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]
```

```
set.seed(123)
train_indices <- sample(1:nrow(mydata[data_norm]), round(0.7 * nrow(mydata[data_norm])))
train_data <- mydata[data_norm][train_indices, predictor_cols]
train_labels <- mydata[data_norm][train_indices, target_col]
test_data <- mydata[data_norm][-train_indices, predictor_cols]
test_labels <- mydata[data_norm][-train_indices, target_col]
```

```
# Set the value of k (number of neighbors) and distance measures
```

```
knn_with_distance_measure <- function(train_data, test_data, train_labels, k,
                                       distance_measure) {
  predicted_labels <- knn(train = train_data, test = test_data, cl = train_labels, k = k, prob =
    TRUE, use.all = TRUE)
```

```
    return(predicted_labels)
}
```

```
# Set the values of k
```

```
k_values <- c(3, 5, 7)
```

```
# Initialize vectors to store accuracies
```

```
accuracies <- vector()
```

```
# Apply k-NN for each k value and distance measure
```

```
for (k in k_values) {
```

```
  # Apply k-NN with Euclidean distance
```

```
  euclidean_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "euclidean")
```

```
  # Apply k-NN with Manhattan distance
```

```
  manhattan_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "manhattan")
```

```
  # Apply k-NN with Maximum distance
```

```
  maximum_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "maximum")
```

```
  # Evaluate the accuracy of the predictions
```

```
  accuracy_euclidean <- sum(euclidean_predictions == test_labels) / length(test_labels)
```

```
  accuracy_manhattan <- sum(manhattan_predictions == test_labels) / length(test_labels)
```

```
  accuracy_maximum <- sum(maximum_predictions == test_labels) / length(test_labels)
```

```
  # Store the accuracy
```

```
  accuracies <- c(accuracies, accuracy_euclidean, accuracy_manhattan, accuracy_maximum)
```

```
  # Print the accuracy for the current k value
```

```
  cat("Accuracy for k =", k, "\n")
```

```
  cat("Euclidean Distance:", accuracy_euclidean, "\n")
```

```
cat("Manhattan Distance:", accuracy_manhattan, "\n")  
cat("Maximum Distance:", accuracy_maximum, "\n")  
cat("\n")  
}
```

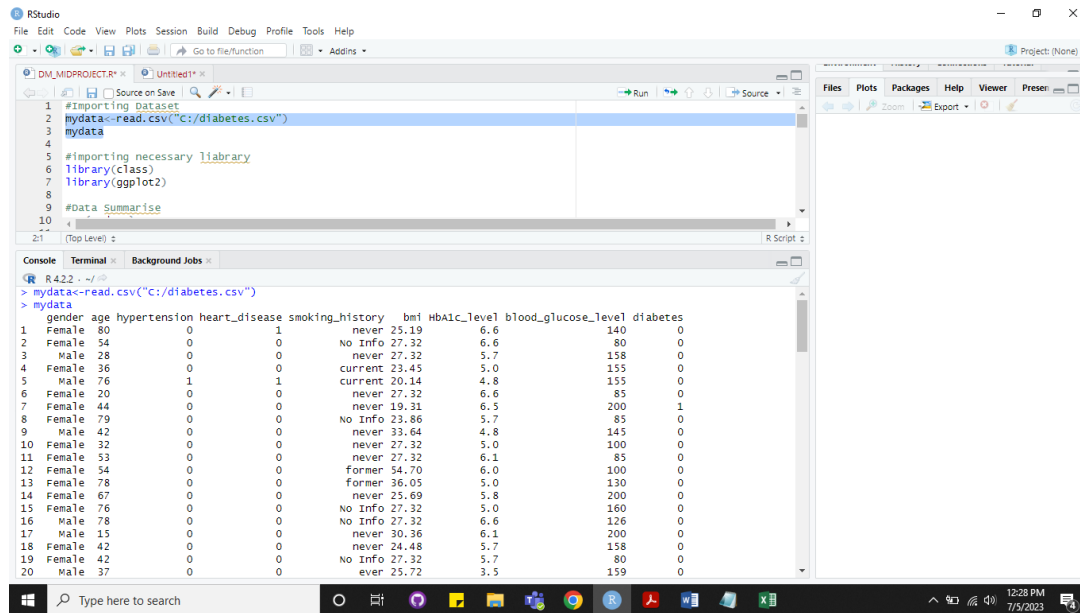
```
# Create a data frame for accuracies
```

```
accuracy_df <- data.frame(Distance = rep(c("Euclidean", "Manhattan", "Maximum"),  
                                     length(k_values)),  
                          K = rep(k_values, each = 3),  
                          Accuracy = accuracies)  
accuracy_df
```

```
# Plot the accuracies
```

```
ggplot(accuracy_df, aes(x = K, y = Accuracy, color = Distance, group = Distance)) +  
  geom_line() +  
  geom_point() +  
  labs(title = "Accuracy of k-NN with Different Distance Measures",  
        x = "k",  
        y = "Accuracy",  
        color = "Distance Measure") +  
  theme_minimal()
```

## 1. Importing Dataset



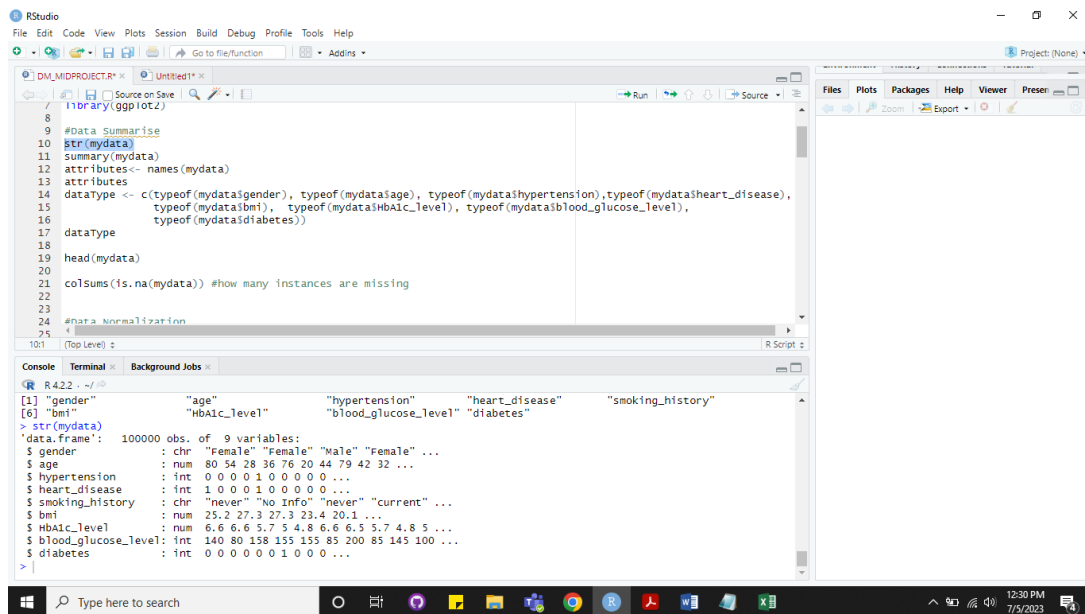
The screenshot shows the RStudio interface. The script editor contains the following code:

```
1 #Importing dataset
2 mydata<-read.csv("C:/diabetes.csv")
3 mydata
4
5 #importing necessary library
6 library(class)
7 library(ggplot2)
8
9 #Data Summarise
10
```

The console output shows the result of the `mydata` command, displaying the first 20 rows of the dataset:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
1	Female	80	0	1	never	25.19	6.6	140	0
2	Female	54	0	0	never	27.32	5.7	158	0
3	Male	28	0	0	current	23.45	5.0	155	0
4	Female	36	0	0	current	20.14	4.8	155	0
5	Male	76	1	1	never	27.32	6.6	85	0
6	Female	20	0	0	never	19.31	6.5	200	1
7	Female	44	0	0	No Info	23.86	5.7	85	0
8	Female	79	0	0	never	33.64	4.8	145	0
9	Male	42	0	0	never	27.32	5.0	100	0
10	Female	32	0	0	never	27.32	6.1	85	0
11	Female	53	0	0	former	54.70	6.0	100	0
12	Female	54	0	0	former	36.05	5.0	130	0
13	Female	78	0	0	never	25.69	5.8	200	0
14	Female	67	0	0	No Info	27.32	5.0	160	0
15	Female	76	0	0	No Info	27.32	6.6	126	0
16	Male	78	0	0	never	30.36	6.1	200	0
17	Male	15	0	0	never	24.48	5.7	158	0
18	Female	42	0	0	No Info	27.32	5.7	80	0
19	Female	42	0	0	ever	25.72	3.5	159	0
20	Male	37	0	0					

## 2. Summary



The screenshot shows the RStudio interface. The script editor contains the following code:

```
1 library(ggplot2)
2
3 #Data Summarise
4 str(mydata)
5 summary(mydata)
6 attributes<- names(mydata)
7 attributes
8
9 datatype <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension), typeof(mydata$heart_disease),
10               typeof(mydata$bmi), typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
11               typeof(mydata$diabetes))
12
13 datatype
14
15 head(mydata)
16
17 colSums(is.na(mydata)) #how many instances are missing
18
19 #Data Normalization
20
```

The console output shows the result of the `str(mydata)` command, displaying the structure of the dataset:

```
[1] "gender"      "age"          "hypertension" "heart_disease" "smoking_history"
[6] "bmi"         "HbA1c_level"  "blood_glucose_level" "diabetes"
```

The output of `summary(mydata)` is also shown:

```
> str(mydata)
'data.frame':   100000 obs. of  9 variables:
 $ gender      : chr  "female" "female" "male" "female" ...
 $ age         : num  80 54 28 36 76 20 44 79 42 32 ...
 $ hypertension: int   0 0 0 1 0 0 0 0 0 ...
 $ heart_disease: int   1 0 0 1 0 0 0 0 0 ...
 $ smoking_history: chr  "never" "No Info" "never" "current" ...
 $ bmi         : num  25.2 27.3 27.3 23.4 20.1 ...
 $ HbA1c_level : num  6.6 6.6 5.7 5.4 8.6 6.6 6.5 5.7 4.8 5 ...
 $ blood_glucose_level: int  140 80 158 155 155 85 200 85 145 100 ...
 $ diabetes    : int   0 0 0 0 0 1 0 0 0 ...
```

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

DM\_MIDPROJECT.R\* x Untitled1\* x

```

7 library(ggplot2)
8
9 #Data Summarise
10 str(mydata)
11 summary(mydata)
12 attributes<- names(mydata)
13 attributes
14 dataType <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension),typeof(mydata$heart_disease),
15               typeof(mydata$bmi), typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
16               typeof(mydata$diabetes))
17 dataType
18
19 head(mydata)
20
21 colSums(is.na(mydata)) #how many instances are missing
22
23
24 #data Normalization
25
11:1 (Top Level)

```

Console Terminal Background Jobs

R 4.2.2 ~ /

```

$ diabetes      : int  0 0 0 0 0 0 1 0 0 0 ...
> summary(mydata)
  gender      age      hypertension      heart_disease      smoking_history      bmi
Length:100000 Min. : 0.08 Min. :0.00000 Min. :0.00000 Length:100000 Min. :10.01
Class :character 1st Qu.:24.00 1st Qu.:0.00000 1st Qu.:0.00000 Class :character 1st Qu.:23.63
Mode :character  Median :43.00 Median :0.00000 Median :0.00000 Mode :character  Median :27.32
                Mean  :41.89 Mean  :0.07485 Mean  :0.03942                Mean  :27.32
                3rd Qu.:60.00 3rd Qu.:0.00000 3rd Qu.:0.00000                3rd Qu.:29.58
                Max.  :80.00 Max.  :1.00000 Max.  :1.00000                Max.  :95.69

HbA1c_level      blood_glucose_level      diabetes
Min. :3.500 Min. : 80.0 Min. :0.000
1st Qu.:4.800 1st Qu.:100.0 1st Qu.:0.000
Median :5.800 Median :140.0 Median :0.000
Mean  :5.528 Mean  :138.1 Mean  :0.085
3rd Qu.:6.200 3rd Qu.:159.0 3rd Qu.:0.000
Max.  :9.000 Max.  :300.0 Max.  :1.000

```

Type here to search

12:30 PM 7/5/2023

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

DM\_MIDPROJECT.R\* x Untitled1\* x

```

7 library(ggplot2)
8
9 #Data Summarise
10 str(mydata)
11 summary(mydata)
12 attributes<- names(mydata)
13 attributes
14 dataType <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension),typeof(mydata$heart_disease),
15               typeof(mydata$bmi), typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
16               typeof(mydata$diabetes))
17 dataType
18
19 head(mydata)
20
21 colSums(is.na(mydata)) #how many instances are missing
22
23
24 #data Normalization
25
12:1 (Top Level)

```

Console Terminal Background Jobs

R 4.2.2 ~ /

```

      3rd Qu.:60.00 3rd Qu.:0.00000 3rd Qu.:0.00000      3rd Qu.:29.58
      Max. :80.00 Max. :1.00000 Max. :1.00000      Max. :95.69

HbA1c_level      blood_glucose_level      diabetes
Min. :3.500 Min. : 80.0 Min. :0.000
1st Qu.:4.800 1st Qu.:100.0 1st Qu.:0.000
Median :5.800 Median :140.0 Median :0.000
Mean  :5.528 Mean  :138.1 Mean  :0.085
3rd Qu.:6.200 3rd Qu.:159.0 3rd Qu.:0.000
Max.  :9.000 Max.  :300.0 Max.  :1.000
> attributes<- names(mydata)
> attributes
[1] "gender"      "age"      "hypertension" "heart_disease" "smoking_history"
[6] "bmi"      "HbA1c_level" "blood_glucose_level" "diabetes"
>

```

Type here to search

12:30 PM 7/5/2023

The screenshot shows the RStudio interface with a script editor, console, and terminal. The script defines a data frame, summarizes it, and assigns data types to each column. The console shows the output of the summary function.

```
library(ggplot2)

#Data Summary
str(mydata)
summary(mydata)
attributes<- names(mydata)
attributes
datatype <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension),typeof(mydata$heart_disease),
              typeof(mydata$bmi),   typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
              typeof(mydata$diabetes))
datatype
head(mydata)
colSums(is.na(mydata)) #how many instances are missing

#Data Normalization
```

Console output:

```
R 4.2.2 ~ ./
Median :5.800   Median :140.0   Median :0.000
Mean    :5.528   Mean    :138.1   Mean    :0.085
3rd Qu.:6.200   3rd Qu.:139.0   3rd Qu.:0.000
Max.    :9.000   Max.    :300.0   Max.    :1.000
> attributes<- names(mydata)
> attributes
[1] "gender"      "age"          "hypertension" "heart_disease" "smoking_history"
[6] "bmi"         "HbA1c_level"  "blood_glucose_level" "diabetes"
> datatype <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension),typeof(mydata$heart_disease),
+               typeof(mydata$bmi),   typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
+               typeof(mydata$diabetes))
> datatype
[1] "character" "double"    "integer"   "integer"   "double"    "double"    "integer"   "integer"
```

### 3. Looking for missing values

The screenshot shows the RStudio interface with a script editor, console, and terminal. The script normalizes the data and checks for missing values. The console shows the output of the normalization steps.

```
attributes<- names(mydata)
attributes
datatype <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension),typeof(mydata$heart_disease),
              typeof(mydata$bmi),   typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
              typeof(mydata$diabetes))
datatype
colSums(is.na(mydata)) #how many instances are missing

#Data Normalization
data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
mydata[data_norm] <- scale(mydata[data_norm])
head(mydata)
```

Console output:

```
R 4.2.2 ~ ./
> datatype <- c(typeof(mydata$gender), typeof(mydata$age), typeof(mydata$hypertension),typeof(mydata$heart_disease),
+               typeof(mydata$bmi),   typeof(mydata$HbA1c_level), typeof(mydata$blood_glucose_level),
+               typeof(mydata$diabetes))
> datatype
[1] "character" "double"    "integer"   "integer"   "double"    "double"    "integer"   "integer"
> head(mydata)
  gender age hypertension heart_disease smoking_history  bmi HbA1c_level blood_glucose_level diabetes
1 Female  80           0             1         never 25.19      6.6          140.0             0
2 Female  54           0             0         No info 27.32      6.6           80.0             0
3 Male   28           0             0         never 27.32      5.7          158.0             0
4 Female  36           0             0         current 23.45      5.0          155.0             0
5 Male   76           1             1         current 20.14      4.8          155.0             0
6 Female  20           0             0         never 27.32      6.6           85.0             0
```



## 4. Data Normalization

```
20
21 colSums(is.na(mydata)) #how many instances are missing
22
23
24 #Data Normalization
25 data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
26 mydata[data_norm] <- scale(mydata[data_norm])
27
28 head(mydata)
29 head(data_norm)
30 data_norm
31 mydata[data_norm]
32 colSums(is.na(mydata[data_norm]))
33
34
35
36 # Setting predictor variables and the target variable
37 predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]
38
39
40
```

Console Terminal Background Jobs

```
R 4.2.2 ~ /
1 Female 80 0 1 never 25.19 6.6 140 0
2 Female 54 0 0 No Info 27.32 6.6 80 0
3 Male 28 0 0 never 27.32 5.7 158 0
4 Female 36 0 0 current 23.45 5.0 155 0
5 Male 76 1 1 current 20.14 4.8 155 0
6 Female 20 0 0 never 27.32 6.6 85 0
> colSums(is.na(mydata))
gender age hypertension heart_disease smoking_history
0 0 0 0 0
bml HbA1c_level blood_glucose_level diabetes
0 0 0 0
> data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
> mydata[data_norm] <- scale(mydata[data_norm])
>
|
```

```
24 #Data Normalization
25 data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
26 mydata[data_norm] <- scale(mydata[data_norm])
27
28 head(mydata)
29 head(data_norm)
30 data_norm
31 mydata[data_norm]
32 colSums(is.na(mydata[data_norm]))
33
34
35
36 # Setting predictor variables and the target variable
37 predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]
38 target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]
39
40 set.seed(123)
41
42
```

Console Terminal Background Jobs

```
R 4.2.2 ~ /
4 Female -0.2613980 -0.284438 -0.2025766 current -0.5832293834 -0.4926877 0.41618069
5 Male 1.5150503 3.515669 4.9363539 current -1.0819649594 -0.6794863 0.41618069
6 Female -0.9719772 -0.284438 -0.2025766 never -0.0001155831 1.0017007 -1.30337729
diabetes
1 -0.3047872
2 -0.3047872
3 -0.3047872
4 -0.3047872
5 -0.3047872
6 -0.3047872
> head(data_norm)
[1] "age" "hypertension" "heart_disease" "bml" "HbA1c_level"
[6] "blood_glucose_level"
>
|
```

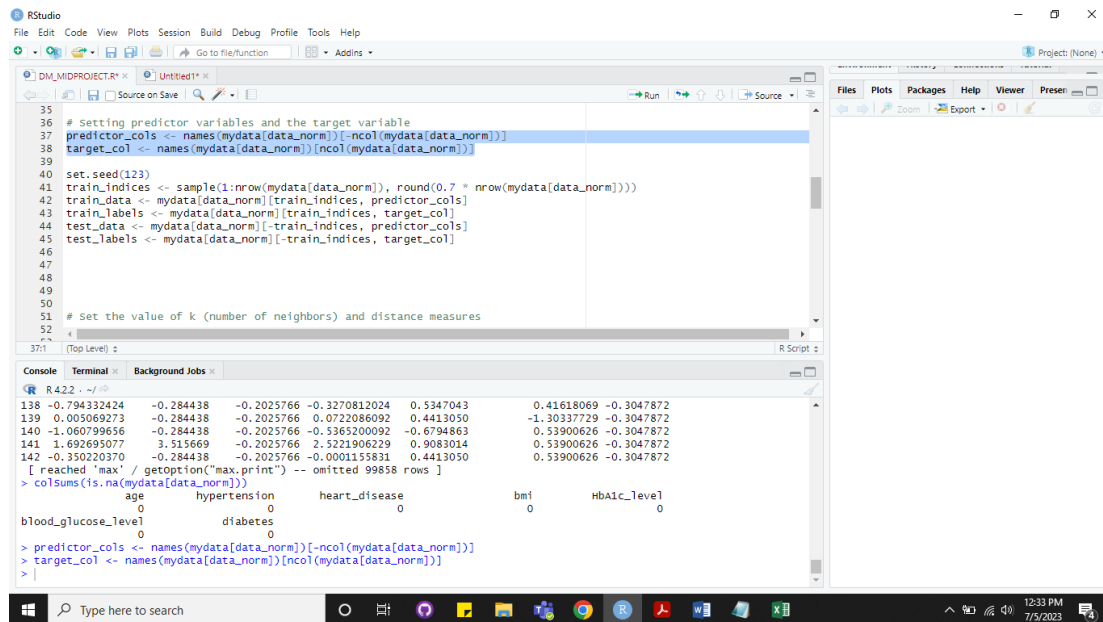
```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Project: (None)
Files Plots Packages Help Viewer Presenter
Zoom Export
24 #Data Normalization
25 data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
26 mydata[data_norm] <- scale(mydata[data_norm])
27
28 head(mydata)
29 head(data_norm)
30 data_norm
31 mydata[data_norm]
32 colSums(is.na(mydata[data_norm]))
33
34
35
36 # Setting predictor variables and the target variable
37 predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]
38 target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]
39
40 set.seed(123)
41
```

```
R 4.2.2 ~ /
> mydata[data_norm]
      age hypertension heart_disease      bmi HbA1c_level blood_glucose_level diabetes
1  1.692695077      -0.284438      -0.2025766 -0.3210541864      1.0017007      0.04770398 -0.3047872
2  0.538003737      -0.284438      -0.2025766 -0.0001155831      1.0017007      -1.42620286 -0.3047872
3  -0.616687603      -0.284438      -0.2025766 -0.0001155831      0.1611072      0.48987603 -0.3047872
4  -0.261397960      -0.284438      -0.2025766 -0.5832293834      -0.4926877      0.41618069 -0.3047872
5  1.513050255      3.515669      4.9363539 -1.0819649594      -0.6794863      0.41618069 -0.3047872
6  -0.971977246      -0.284438      -0.2025766 -0.0001155831      1.0017007      -1.30337729 -0.3047872
7  0.093891683      -0.284438      -0.2025766 -1.2070255419      0.9083014      1.52161081 3.2809447
8  1.648283872      -0.284438      -0.2025766 -0.5214524692      0.1611072      -1.30337729 -0.3047872
9  0.005069273      -0.284438      -0.2025766 0.9521529487      -0.6794863      0.17052953 -0.3047872
10 -0.439042781      -0.284438      -0.2025766 -0.0001155831      -0.4926877      -0.93490058 -0.3047872
11 0.493592532      -0.284438      -0.2025766 -0.0001155831      0.5347043      -1.30337729 -0.3047872
12 0.538003737      -0.284438      -0.2025766 4.1253768853      0.4413050      -0.93490058 -0.3047872
```

```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins
Project: (None)
Files Plots Packages Help Viewer Presenter
Zoom Export
24 #Data Normalization
25 data_norm <- setdiff(names(mydata), c("gender", "smoking_history"))
26 mydata[data_norm] <- scale(mydata[data_norm])
27
28 head(mydata)
29 head(data_norm)
30 data_norm
31 mydata[data_norm]
32 colSums(is.na(mydata[data_norm]))
33
34
35
36 # Setting predictor variables and the target variable
37 predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]
38 target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]
39
40 set.seed(123)
41
```

```
R 4.2.2 ~ /
136 1.692695077      -0.284438      -0.2025766 -0.7926681903      3.2432834      0.41618069 3.2809447
137 0.227125300      -0.284438      -0.2025766 1.3815778404      1.8422942      0.41618069 3.2809447
138 -0.794332424      -0.284438      -0.2025766 -0.3270812024      0.5347043      0.41618069 -0.3047872
139 0.005069273      -0.284438      -0.2025766 0.0722086092      0.4413050      -1.30337729 -0.3047872
140 -1.060799656      -0.284438      -0.2025766 -0.5365200092      -0.6794863      0.53900626 -0.3047872
141 1.692695077      3.515669      -0.2025766 2.5221906229      0.9083014      0.53900626 -0.3047872
142 -0.350220370      -0.284438      -0.2025766 -0.0001155831      0.4413050      0.53900626 -0.3047872
[ reached 'max' / getoption("max.print") -- omitted 99858 rows ]
> colSums(is.na(mydata[data_norm]))
      age hypertension heart_disease      bmi HbA1c_level
0      0      0      0      0
blood_glucose_level diabetes
0      0
```

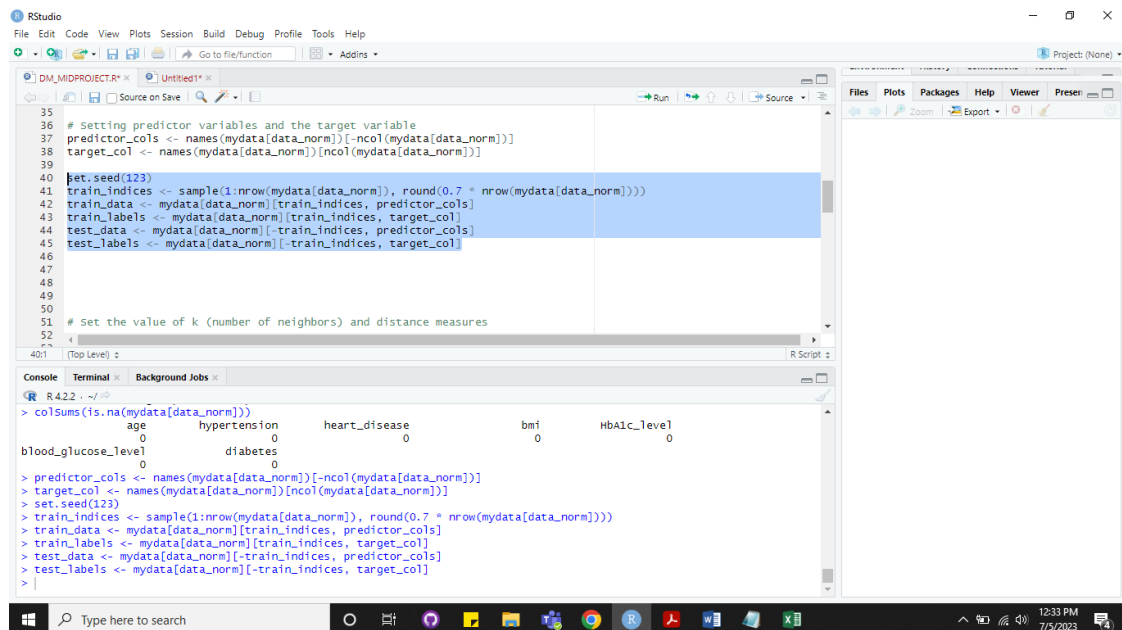
## 5. Setting predictor variables and target variable



The screenshot shows the RStudio interface with a script editor and a console. The script editor contains R code for setting predictor and target variables. The console shows the output of the code, including a summary of the data and the values of the predictor and target variables.

```
35  
36 # Setting predictor variables and the target variable  
37 predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]  
38 target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]  
39  
40 set.seed(123)  
41 train_indices <- sample(1:nrow(mydata[data_norm]), round(0.7 * nrow(mydata[data_norm])))  
42 train_data <- mydata[data_norm][train_indices, predictor_cols]  
43 train_labels <- mydata[data_norm][train_indices, target_col]  
44 test_data <- mydata[data_norm][-train_indices, predictor_cols]  
45 test_labels <- mydata[data_norm][-train_indices, target_col]  
46  
47  
48  
49  
50  
51 # Set the value of k (number of neighbors) and distance measures  
52  
37:1 (Top Level) ± R Script ±  
Console Terminal Background Jobs  
R 4.2.2 . ~/ /  
138 -0.794332424 -0.284438 -0.2025766 -0.3270812024 0.5347043 0.41618069 -0.3047872  
139 0.005069273 -0.284438 -0.2025766 0.0722086092 0.4413050 -1.30337729 -0.3047872  
140 -1.060799656 -0.284438 -0.2025766 -0.5365200092 -0.6794863 0.53900626 -0.3047872  
141 1.692695077 3.515669 -0.2025766 2.5221906229 0.9083014 0.53900626 -0.3047872  
142 -0.350220370 -0.284438 -0.2025766 -0.000115831 0.4413050 0.53900626 -0.3047872  
[ reached 'max' / getoption("max.print") -- omitted 99858 rows ]  
> colSums(is.na(mydata[data_norm]))  
age hypertension heart_disease 0 bmi HbA1c_level  
blood_glucose_level diabetes  
0 0  
> predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]  
> target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]  
> |
```

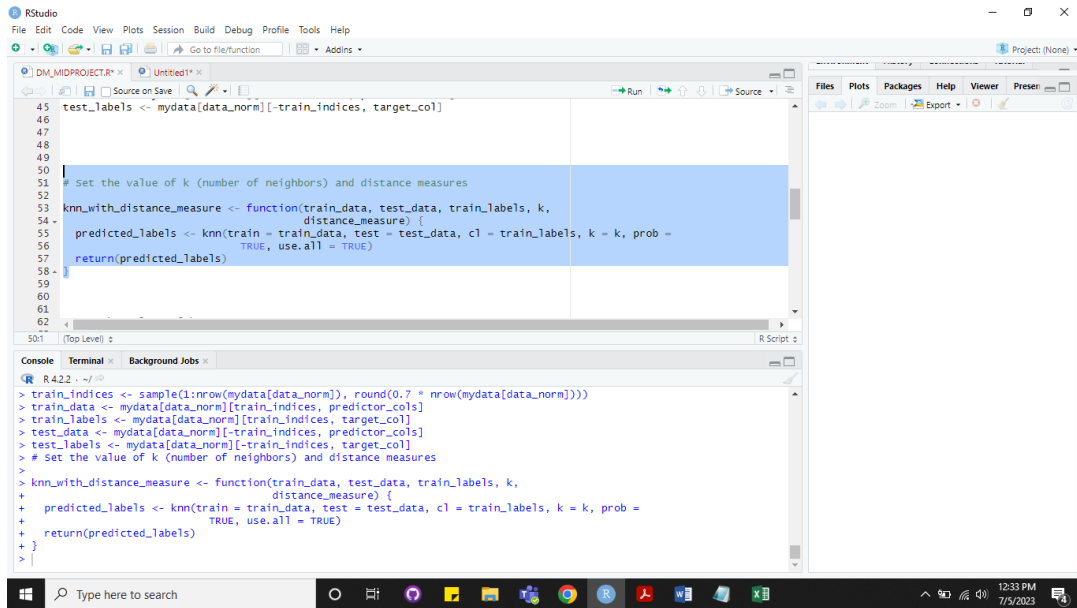
## 6. Splitting into Training and Test data



The screenshot shows the RStudio interface with a script editor and a console. The script editor contains R code for splitting the data into training and test sets. The console shows the output of the code, including a summary of the data and the values of the predictor and target variables.

```
35  
36 # Setting predictor variables and the target variable  
37 predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]  
38 target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]  
39  
40 set.seed(123)  
41 train_indices <- sample(1:nrow(mydata[data_norm]), round(0.7 * nrow(mydata[data_norm])))  
42 train_data <- mydata[data_norm][train_indices, predictor_cols]  
43 train_labels <- mydata[data_norm][train_indices, target_col]  
44 test_data <- mydata[data_norm][-train_indices, predictor_cols]  
45 test_labels <- mydata[data_norm][-train_indices, target_col]  
46  
47  
48  
49  
50  
51 # Set the value of k (number of neighbors) and distance measures  
52  
40:1 (Top Level) ± R Script ±  
Console Terminal Background Jobs  
R 4.2.2 . ~/ /  
> colSums(is.na(mydata[data_norm]))  
age hypertension heart_disease 0 bmi HbA1c_level  
blood_glucose_level diabetes  
0 0  
> predictor_cols <- names(mydata[data_norm])[1:ncol(mydata[data_norm])]  
> target_col <- names(mydata[data_norm])[ncol(mydata[data_norm])]  
> set.seed(123)  
> train_indices <- sample(1:nrow(mydata[data_norm]), round(0.7 * nrow(mydata[data_norm])))  
> train_data <- mydata[data_norm][train_indices, predictor_cols]  
> train_labels <- mydata[data_norm][train_indices, target_col]  
> test_data <- mydata[data_norm][-train_indices, predictor_cols]  
> test_labels <- mydata[data_norm][-train_indices, target_col]  
> |
```

## 7. Implementing KNN algorithm

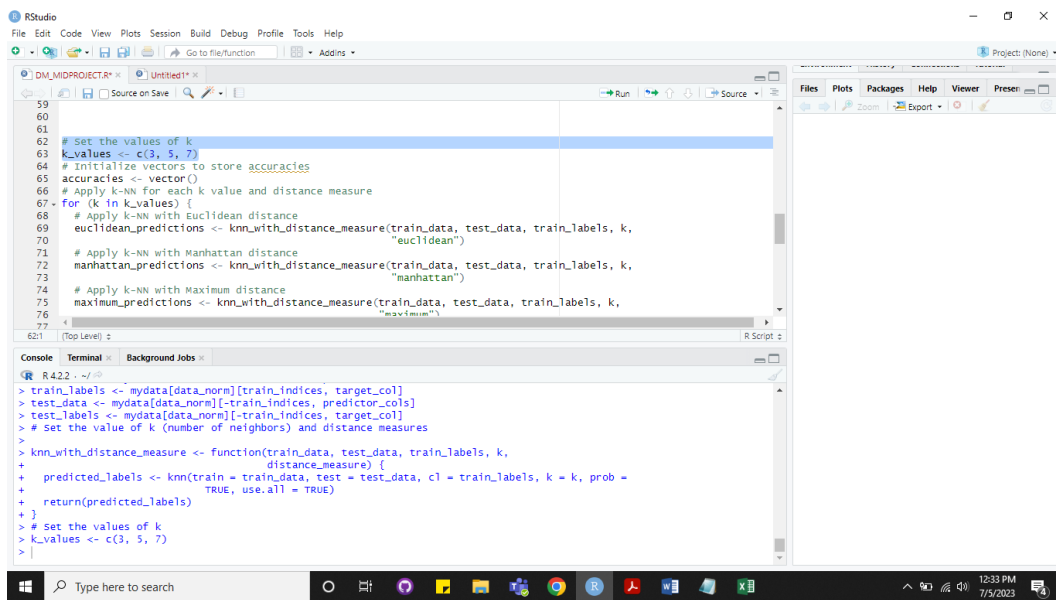


```
test_labels <- mydata[data_norm][~train_indices, target_col]

# Set the value of k (number of neighbors) and distance measures
knn_with_distance_measure <- function(train_data, test_data, train_labels, k,
distance_measure) {
  predicted_labels <- knn(train = train_data, test = test_data, cl = train_labels, k = k, prob =
  TRUE, use.all = TRUE)
  return(predicted_labels)
}

train_indices <- sample(1:nrow(mydata[data_norm]), round(0.7 * nrow(mydata[data_norm])))
train_data <- mydata[data_norm][train_indices, predictor_cols]
train_labels <- mydata[data_norm][train_indices, target_col]
test_data <- mydata[data_norm][~train_indices, predictor_cols]
test_labels <- mydata[data_norm][~train_indices, target_col]
# Set the value of k (number of neighbors) and distance measures
knn_with_distance_measure <- function(train_data, test_data, train_labels, k,
distance_measure) {
  predicted_labels <- knn(train = train_data, test = test_data, cl = train_labels, k = k, prob =
  TRUE, use.all = TRUE)
  return(predicted_labels)
}
```

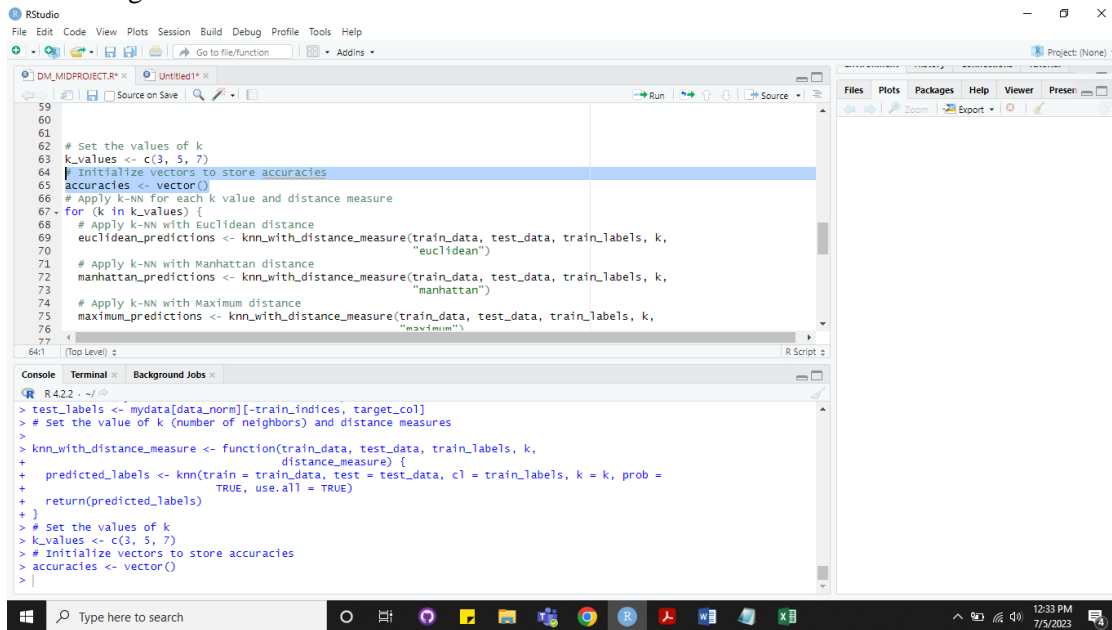
## 8. Initialize K values



```
# set the values of k
k_values <- c(3, 5, 7)
# Initialize vectors to store accuracies
accuracies <- vector()
# Apply k-NN for each k value and distance measure
for (k in k_values) {
  # Apply k-NN with Euclidean distance
  euclidean_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
"euclidean")
  # Apply k-NN with Manhattan distance
  manhattan_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
"manhattan")
  # Apply k-NN with Maximum distance
  maximum_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
"maximum")
}

train_labels <- mydata[data_norm][train_indices, target_col]
test_data <- mydata[data_norm][~train_indices, predictor_cols]
test_labels <- mydata[data_norm][~train_indices, target_col]
# Set the value of k (number of neighbors) and distance measures
knn_with_distance_measure <- function(train_data, test_data, train_labels, k,
distance_measure) {
  predicted_labels <- knn(train = train_data, test = test_data, cl = train_labels, k = k, prob =
  TRUE, use.all = TRUE)
  return(predicted_labels)
}
# Set the values of k
k_values <- c(3, 5, 7)
```

## 9. Initializing vectors to store accuracies

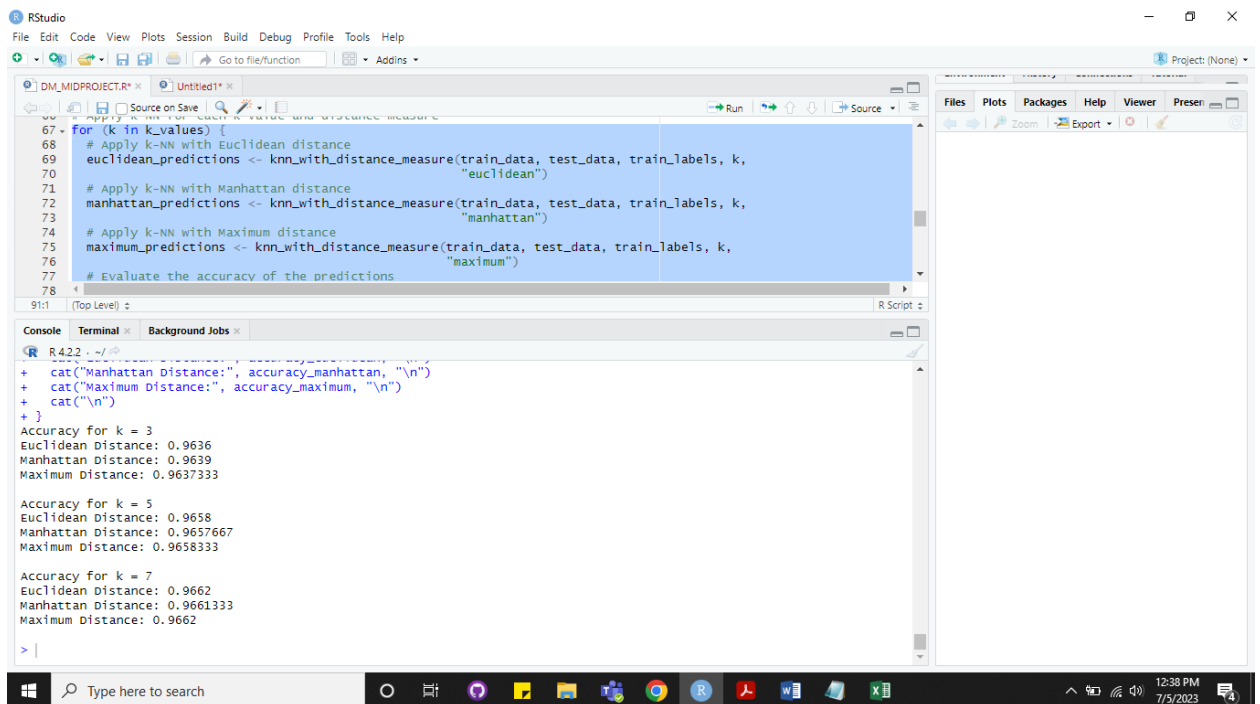


The screenshot shows the RStudio interface with a script editor and a console. The script editor contains R code for initializing vectors and applying k-NN with different distance measures. The console shows the execution of the code.

```
# Set the values of k
k_values <- c(3, 5, 7)
# Initialize vectors to store accuracies
accuracies <- vector()
# Apply k-NN for each k value and distance measure
for (k in k_values) {
  # Apply k-NN with Euclidean distance
  euclidean_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "euclidean")
  # Apply k-NN with Manhattan distance
  manhattan_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "manhattan")
  # Apply k-NN with Maximum distance
  maximum_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "maximum")
}
```

```
> test_labels <- mydata[data_norm][,-train_indices, target_col]
> # Set the value of k (number of neighbors) and distance measures
> knn_with_distance_measure <- function(train_data, test_data, train_labels, k,
+   distance_measure) {
+   predicted_labels <- knn(train = train_data, test = test_data, cl = train_labels, k = k, prob =
+   TRUE, use.all = TRUE)
+   return(predicted_labels)
+ }
> # Set the values of k
> k_values <- c(3, 5, 7)
> # Initialize vectors to store accuracies
> accuracies <- vector()
>
```

## 10. Apply k-NN for each k value and distance measure



The screenshot shows the RStudio interface with a script editor and a console. The script editor contains R code for applying k-NN for each k value and distance measure. The console shows the execution of the code and the resulting accuracies.

```
for (k in k_values) {
  # Apply k-NN with Euclidean distance
  euclidean_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "euclidean")
  # Apply k-NN with Manhattan distance
  manhattan_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "manhattan")
  # Apply k-NN with Maximum distance
  maximum_predictions <- knn_with_distance_measure(train_data, test_data, train_labels, k,
                                                    "maximum")
  # Evaluate the accuracy of the predictions
}
```

```
+ cat("Manhattan Distance:", accuracy_manhattan, "\n")
+ cat("Maximum Distance:", accuracy_maximum, "\n")
+ cat("\n")
+ }
Accuracy for k = 3
Euclidean Distance: 0.9636
Manhattan Distance: 0.9639
Maximum Distance: 0.9637333

Accuracy for k = 5
Euclidean Distance: 0.9658
Manhattan Distance: 0.9657667
Maximum Distance: 0.9658333

Accuracy for k = 7
Euclidean Distance: 0.9662
Manhattan Distance: 0.9661333
Maximum Distance: 0.9662
>
```

## 11. Create Data Frame for accuracy

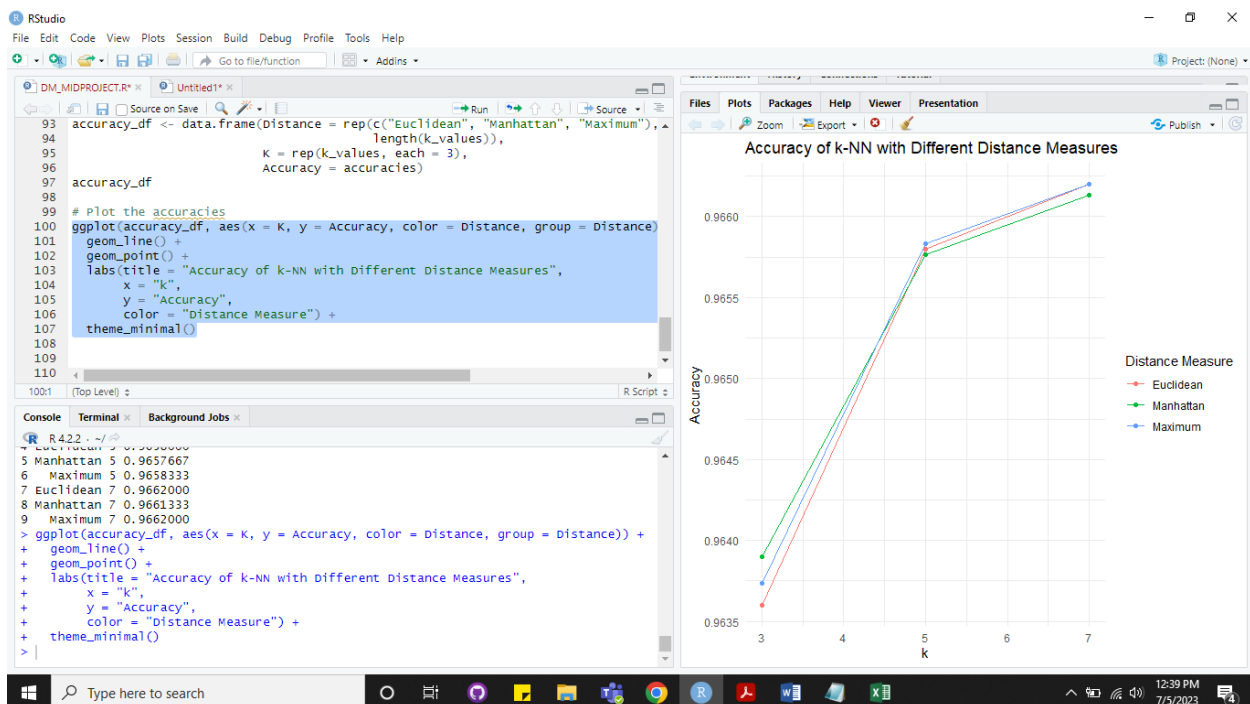
```
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

DM_MIDPROJECT.R* x1 Untitled1* x2
Source on Save Run Go to file/function Addins

92 # Create a data frame for accuracies
93 accuracy_df <- data.frame(Distance = rep(c("Euclidean", "Manhattan", "Maximum"),
94                               length(k_values)),
95                           K = rep(k_values, each = 3),
96                           Accuracy = accuracies)
97 accuracy_df
98
99 # Plot the accuracies
100 ggplot(accuracy_df, aes(x = K, y = Accuracy, color = Distance, group = Distance)) +
101   geom_line() +
102   geom_point() +
103   labs(title = "Accuracy of k-NN with Different Distance Measures",
104         y = "k")
105
93:1 (Top Level) R Script
```

```
R 4.2.2 ~ ./
+                               length(k_values)),
+                               K = rep(k_values, each = 3),
+                               Accuracy = accuracies)
> accuracy_df
  Distance K Accuracy
1 Euclidean 3 0.9636000
2 Manhattan 3 0.9639000
3 Maximum 3 0.9637333
4 Euclidean 5 0.9658000
5 Manhattan 5 0.9657667
6 Maximum 5 0.9658333
7 Euclidean 7 0.9662000
8 Manhattan 7 0.9661333
9 Maximum 7 0.9662000
> |
```

## 12. Plotting the accuracies



The graph above shows how the accuracy of the knn algorithm varies with different values of k on using the 3 distance measuring methods (Euclidean, Manhattan, and Maximum Dimension).

It can be seen that the accuracy was the highest when the value of  $k$  was 7. In this instance, Euclidean distance had the highest accuracy, followed by Maximum Dimension and then Manhattan.

When the value of  $k$  was increased to 5, the accuracies of all the distance method dropped, but the Maximum dimension was the highest, followed by Euclidean and then Manhattan.

Further, when the value was set to 3, the accuracy dropped. Also noticeable is that the Manhattan took the lead making the Euclidean last and Maximum the second last.

Among the three distance measures, Performance of Euclidean and Manhattan distance was giving the best in terms of accuracy. This indicates that for this particular dataset, this two distance measure was more suitable for distinguishing between the different attributes.