EXPERIMENT -2

AIM:- Create two vectors in R for Numeric Data.

PROCEDURE:-

#create two vectors for numeric data x
<- c(1,2,3,4,5)

use c() to combine elements into a numeric vector of
length 5 y <- c(6,7,8,9,10) x y class(x) class(y)

#extract the second element of x x[2]
#update the second element of y to 100 y[2]
<-100
y
#operations between two numeric vectors
print(paste("addition", x+y)) print(paste("division",y/5))
print(paste("subtraction",y-x))

#create two vectors for numeric data x <- c(1,2,3,4,5)

use c() to combine elements into a numeric vector of
length 5 y <- c(6,7,8,9,10) x y class(x) class(y)

#extract the second element of x x[2]

#update the second element of y to 100 y[2] <-
100 y

#operations between two numeric vectors
print(paste("addition", x+y))

INPUT/OUTPUT:-

```
> source("C:/Users/Hp/Desktop/2pr.r", echo=TRUE)

> #create two vectors for numeric data
> x <- c(1,2,3,4,5)

> # use c() to combine elements into a numeric vector of length 5
> y <- c(6,7,8,9,10)

> x
[1] 1 2 3 4 5

> y
[1]  6  7  8  9 10

> class(x)
[1] "numeric"

> class(y)
[1] "numeric"

> #extract the second element of x
> x[2]
[1] 2

> #update the second element of y to 100
> y[2] <-100

> y
[1]   6 100   8   9  10

> #operations between two numeric vectors
> print(paste("addition", x+y))
[1] "addition 7"   "addition 102" "addition 11"  "addition 13"  "addition 15"
```

RESULT:-

The code has been executed successfully.

EXPERIMENT-3

AIM: Create a list in a data structure that has components of mixed data types.

PROCEDURE:

#a vector having elements of different type is

called list.#Use the list() function to create a list.

x <- list(1,"first no.",22, 2, "true")

print(x) print(paste("type of x",

class(x)))y<-list("z"=1:10) print(y)

print(paste("length of list y",

length(y))) #Multiple lists can be

merged list

combined <- c(x, y)

#create an empty list of a prespecified length with the vector()

functionz<- vector("list", length=4)

print(z)

INPUT/OUTPUT:

```
Source
Console   Terminal   Background Jobs
R 4.3.2 · ~/
> source("~/.active-rstudio-document", echo=TRUE)

> #create a list in a data structure that has components of mixed data types.
> #a vector having elements of different type is called list.
> #use the .... [TRUNCATED]

> print(x)
[[1]]
[1] 1

[[2]]
[1] "first no."

[[3]]
[1] 22

[[4]]
[1] 2

[[5]]
[1] "true"


> print(paste("type of x", class(x)))
[1] "type of x list"

> y<-list("z"=1:10)

> print(y)
$z
 [1]  1  2  3  4  5  6  7  8  9 10


> print(paste("length of list y", length(y)))
[1] "length of list y 1"
```

```
> #Multiple lists can be merged
> list_combined <- c(x, y)

> #create an empty list of a prespecified length with the vector() function
> z<- vector("list", length=4 )

> print(z)
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

>
```

RESULT:

The Code Has Been Executed Successfully .

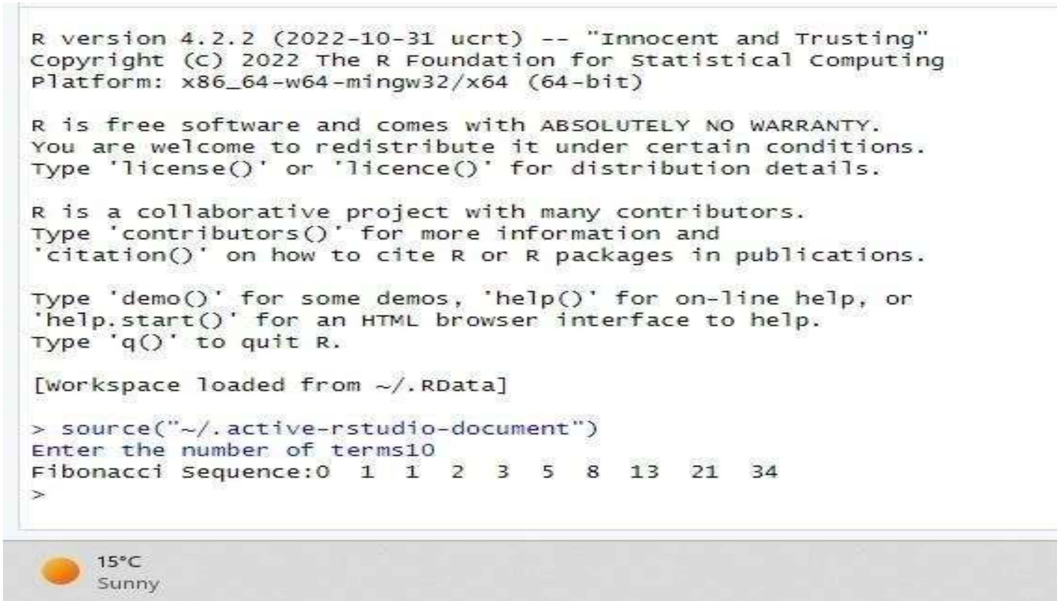
EXPERIMENT-4

AIM:-. Create a code to display Fibonacci series.

PROCEDURE:-

```
print_fibonacci <-
function(n) {a <- 0
b <- 1     cat("Fibonacci Sequence:")     for
  (i in 1:n) {cat(a," ")
next_num <- a + b      a
<- b
   b <- next_num
  }
}
number_of_terms<-readline("Enter the number of terms")
print_fibonacci(number_of_terms)
```

OUTPUT:-



```
R version 4.2.2 (2022-10-31 ucrt) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> source("~/.active-rstudio-document")
Enter the number of terms10
Fibonacci Sequence:0  1  1  2  3  5  8  13  21  34
>
```

15°C
Sunny

RESULT:-

The Code has been executed successfully.

EXPERIMENT-5

Aim: Implement decision tree on credit card issue dataset (import from kaggale).

Procedure:

Implementing a decision tree on a credit card issue dataset involves several steps, including data loading, preprocessing, model training, and evaluation. Below is a simplified example using R and the rpart library for decisiontree modeling. Note that in practice, you might need to adjust the code based on the specific structure and characteristics of your dataset.

First, make sure to install and load the required libraries: install.packages("rpart")library(rpart)

Now, let's assume you have a dataset named "credit_data.csv" with features and labels.

You can load the data and implement a decision tree as follows

```
# Install and load necessary libraries install.packages("rpart ")library(rpart)

# Load the dataset (replace "credit_data.csv" with your actual file path)

                credit_data <- read.csv("credit_data.csv")

# Explore the structure of the datasetstr(credit_data)

# Split the data into training and testing sets
                                set.seed(123)

# Set seed for reproducibility

sample_index <- sample(1:nrow(credit_data), 0.8 * nrow(credit_data))train_data

<- credit_data[sample_index, ] test_data <- credit_data[-
sample_index, ]# Build a decision tree model using rpart

decision_tree_model <- rpart(Class ~ ., data = train_data, method = "class")

# Plot the decision tree
```

plot(decision_tree_model) text(decision_tree_model, cex = 0.8)

Make predictions on the test set

predictions <- predict(decision_tree_model, test_data, type = "class")

Evaluate the model

conf_matrix <- table(predictions, test_data$Class)print("Confusion Matrix:")
print(conf_matrix)

accuracy<- sum(diag(conf_matrix)) / sum(conf_matrix)print(paste("Accuracy:",
round(accuracy, 4)))

OUTPUT:-

Console Terminal Background Jobs

R 4.3.2 · ~/

```
> # Display the first few rows of the dataset to understand its structure
> head(data)
  Time         V1          V2        V3         V4          V5          V6
1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
           V7          V8         V9         V10        V11         V12
1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
         V13        V14        V15        V16         V17         V18
1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
          V19         V20          V21          V22         V23         V24
1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
         V25        V26          V27         V28 Amount Class
1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
```

```
1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
6 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0

> # Identify features and target variable
> X <- data[, -c("target_variable")]  # Replace 'target_variable' with the actual target column name
```

Result:- Successfully Implemented.

EXPERIMENT-6

Aim : Implement the KNN algorithm on the Brest cancer dataset.

Procedure:

To implement the k-Nearest Neighbors (KNN) algorithm on the Breast Cancer dataset, you can use the caret and class packages in R. The Breast Cancer dataset is often available through the datasets package in R. Here's an example.

```
# Install and load necessary libraries
install.packages("caret")
install.packages("class") library(caret)
library(class)

# Load the Breast Cancer dataset
data("BreastCancer")

# Explore the structure of the dataset
str(BreastCancer)

# Split the data into training and testing sets
set.seed(123)
# Set seed for reproducibility
sample_index <- createDataPartition(BreastCancer$Class, p = 0.8, list = FALSE)
 train_data <- BreastCancer[sample_index, ]
test_data <- BreastCancer[-sample_index, ]

# Preprocess the data
# In this example, we'll scale the features
preprocess_params <- preProcess(train_data[, -1], method = c("center", "scale"))

train_data_scaled <- predict(preprocess_params, train_data[, -1]) test_data_scaled <-
predict(preprocess_params, test_data[, -1])

# Train the KNN model
knn_model <- knn(train_data_scaled, test_data_scaled, train_data$Class, k = 5)

# Evaluate the model
conf_matrix <- table(knn_model, test_data$Class) print("Confusion Matrix:")
print(conf_matrix)

accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix) print(paste("Accuracy:",
```

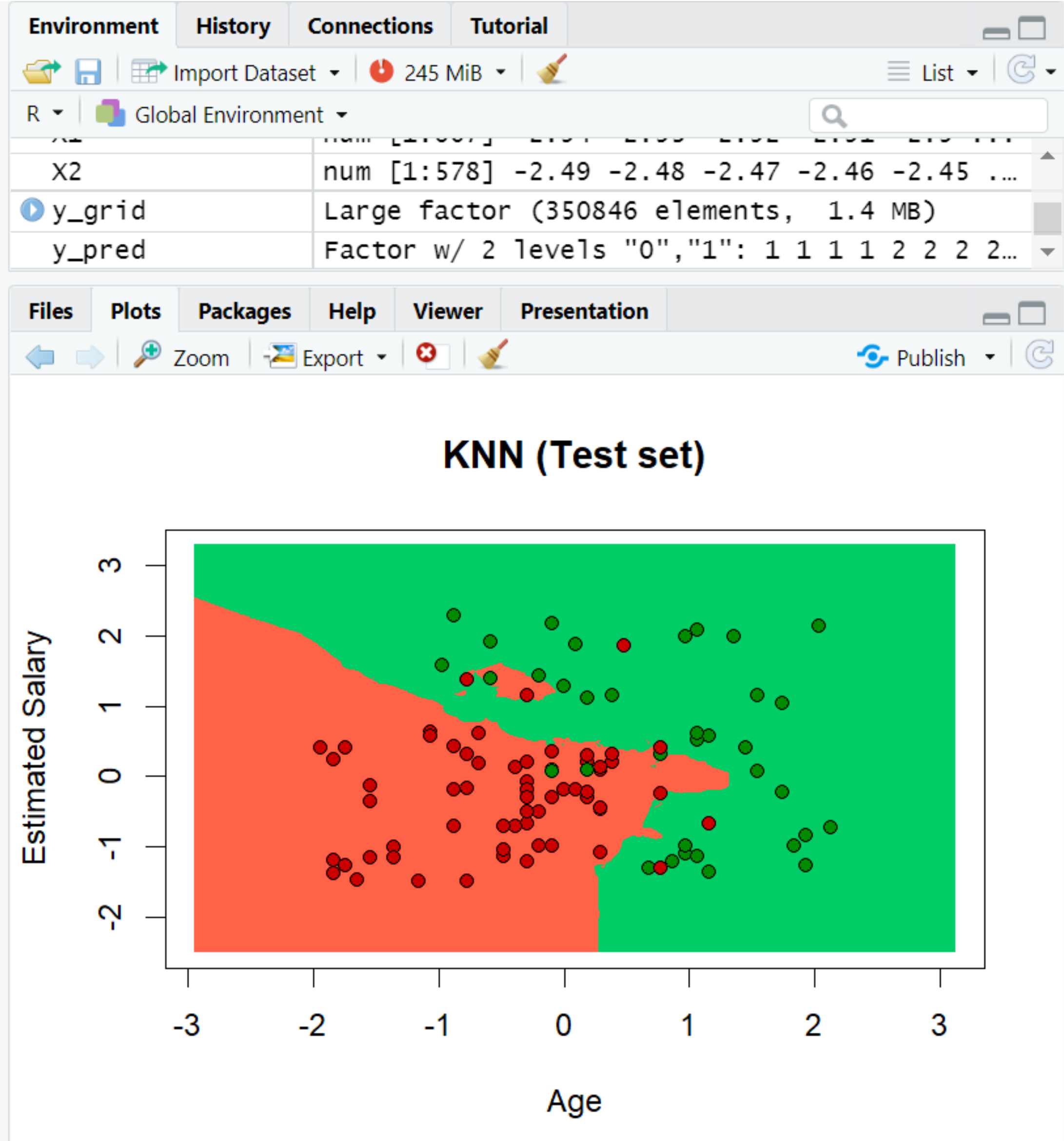
round(accuracy, 4)))

In this example, the Breast Cancer dataset is split into training and testing sets, and the featuresare scaled. The KNN model is then trained using the knn function from the class package, and the accuracy of the model is evaluated.

Remember to replace BreastCancer$Class with the actual column name representing the targetvariable in your dataset. Additionally, consider experimenting with different values of k to find the optimal number of neighbors for your specific dataset.

OUTPUT:-

```
> dataset = read.csv('social.csv')
> dataset = dataset[3:5]
>
> # Enconding the label
> dataset$Purchased = factor(dataset$Purchased, level = c(0, 1))
>
> # Splitting the dataset
> library(caTools)
> split = sample.split(dataset$Purchased, SplitRatio = 0.75)
> training_set = subset(dataset, split == TRUE)
> test_set = subset(dataset, split == FALSE)
>
> # Feature Scaling
> training_set[, 1:2] = scale(training_set[, 1:2])
> test_set[, 1:2] = scale(test_set[, 1:2])
>
> # Fitting and predicting the classifier
> library(class)
> y_pred = knn(train = training_set[, 1:2],
+                test = test_set[, 1:2],
+                cl = training_set[, 3],
+                k = 5)
>
> # Creating the confusion matrix
> cm = table(test_set[, 3], y_pred)
>
> set = training_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'Estimated Salary')
> y_grid = knn(train = training_set[, 1:2],
+               test = grid_set,
+               cl = training_set[, 3],
+               k = 5) # - means removing the column
> plot(set[, -3],
+      main = 'KNN (Training set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
>
> set = test_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'Estimated Salary')
> y_grid = knn(train = training_set[, 1:2],
+               test = grid_set,
+               cl = training_set[, 3],
+               k = 5) # - means removing the column
> plot(set[, -3],
+      main = 'KNN (Test set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```



Result:- The KNN Algorithm have been implemented successfully.

EXPERIMENT-7

Aim : Implement the Naïve Bayes algorithm on the iris dataset.

Procedure:

To implement the Naive Bayes algorithm on the Iris dataset, you can use the function naiveByes from the e1071 package in R

```
# Install and load necessary libraries

install.packages("e101") library(e1071)

# Load the Iris dataset data(iris)

# Explore the structure of the dataset str(iris)

# Split the data into training and testing sets
        set.seed(123)
# Set seed for reproducibility
sample_index <- sample(1:nrow(iris), 0.8 * nrow(iris)) train_data <- iris[sample_index, ]
test_data <- iris[-sample_index, ]

# Train the Naive Bayes model
      naive_bayes_model <- naiveBayes(Species ~ ., data = train_data)

# Make predictions on the test set
      predictions <- predict(naive_bayes_model, test_data)

# Evaluate the model
      conf_matrix <- table(predictions, test_data$Species)
      print("Confusion Matrix:")
      print(conf_matrix)

      accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
      print(paste("Accuracy:", round(accuracy, 4)))
```

OUTPUT:-

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Source

Console Terminal Background Jobs

R 4.3.2 · ~/

```
package ‘caret’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\PRIYANSHU\AppData\Local\Temp\RtmpmKB1b4\downloaded_packages

> # Loading package
> library(e1071)

> library(caTools)

> library(caret)
Loading required package: ggplot2
Loading required package: lattice

> # Splitting data into train
> # and test data
> split <- sample.split(iris, SplitRatio = 0.7)

> train_cl <- subset(iris, split == "TRUE")

> test_cl <- subset(iris, split == "FALSE")

> # Feature Scaling
> train_scale <- scale(train_cl[, 1:4])

> test_scale <- scale(test_cl[, 1:4])

> # Fitting Naive Bayes Model
> # to training dataset
> set.seed(120) # Setting Seed

> classifier_cl <- naiveBayes(Species ~ ., data = train_cl)
```

```
> classifier_cl

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    setosa versicolor  virginica 
 0.3333333  0.3333333  0.3333333 

Conditional probabilities:
            Sepal.Length
Y                [,1]      [,2]
  setosa     4.973333 0.3084257
  versicolor 5.966667 0.4929386
  virginica  6.520000 0.6764002

            Sepal.Width
Y                [,1]      [,2]
  setosa     3.426667 0.3561609
  versicolor 2.776667 0.2712466
  virginica  2.976667 0.3607304

            Petal.Length
Y                [,1]      [,2]
  setosa     1.453333 0.1775957
  versicolor 4.243333 0.4328600
  virginica  5.496667 0.5505379
```

```
> # Model Evaluation
> confusionMatrix(cm)
Confusion Matrix and Statistics

            y_pred
             setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         19         1
  virginica       0          1        19

Overall Statistics

               Accuracy : 0.9667
                 95% CI : (0.8847, 0.9959)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.95

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            0.9500           0.9500
Specificity                 1.0000            0.9750           0.9750
Pos Pred Value              1.0000            0.9500           0.9500
Neg Pred Value              1.0000            0.9750           0.9750
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.3167           0.3167
Detection Prevalence        0.3333            0.3333           0.3333
Balanced Accuracy           1.0000            0.9625           0.9625
>
```

Result:- Successfully Implemented.