1.

```
name<-c("baskar","kannan")
age<-c(20,42)
df<-data.frame(name,age)
print(df)
```

2.

```
name<-c("akash","sri","baskar","kannan")
age<-c(23,45,67,89)
ls<-c(name)
print(ls)
ls<-c(age)
print(ls)
```

4.

```
set.seed(10)
random_vector<-sample(-50:50,10,replace=TRUE)
print(random_vector)
```

3.

```
sequence_20_to_50 <- 20:50
mean_20_to_60 <- mean(20:60)
sum_51_to_91 <- sum(51:91)
```

```
print(sequence_20_to_50)
print(mean_20_to_60)
print(sum_51_to_91)
```

5.

```
fibonacci<-numeric(10)
fibonacci[1]<-0
fibonacci[2]<-1
for(i in 3:10)fibonacci[i]<-fibonacci[i-2]+fibonacci[i-1]
print(fibonacci)
```

6.

```
is_prime <- function(n) {
  if (n <= 1) {
    return(FALSE)
  }

  for (i in 2:sqrt(n)) {
    if (n %% i == 0) {
      return(FALSE)
    }
  }
```

```
  return(TRUE)
}
get_primes_up_to <- function(limit) {
  primes <- c()

  for (num in 2:limit) {
    if (is_prime(num)) {
      primes <- append(primes, num)
    }
  }

  return(primes)
}
limit <- 50
prime_numbers <- get_primes_up_to(limit)
cat("Prime numbers up to", limit, "are:")
print(prime_numbers)
```

7.

```
first_10_lower<-letters[1:10]
```

```
last_10_upper<-LETTERS[17:26]
letters_22_to_24_uppercase<-LETTERS[22:24]
print(first_10_lower)
print(last_10_upper)
letters_22_to_24_uppercase
```

8.

```
print("First 10 letters in lower case:")
t = (letters[1:10])
print(t)
print("Last 10 letters in upper case:")
t = tail(LETTERS, 10)
print(t)
print("Letters between 22nd to 24th letters in upper case:")
e = tail(LETTERS[22:24])
print(e)
```

9.

```
num<-as.integer(readline(prompt="Enter any nu3mber"))
for(i in 1:100)
{
```

```
  if(num%%i==0){
    print(i)
  }
}
```

10.

```
x<-c(10,20,30,31)
a<-max(x)
b<-min(x)
cat("MAXIMUM",a)
cat("MINIMUM",b)
```

11.

```
get_unique_chars<-function(input_string)
  {
  unique_chars<-unique(strsplit(input_string,'')[[1]])
  return(unique_chars)
}
get_unique_numbers<-function(input_vector)
{
  unique_numbers<-unique(input_vector)
  return(unique_numbers)
```

```
}
input_string<-"helllow"
input_vector<-c(1,2,3,4,3,21,2,3,4,5)
unique_chars<-get_unique_chars(input_string)
print(unique_chars)
unique_numbers<-get_unique_numbers(input_vector)
print(unique_numbers)
```

12.

```
a <- c(1, 2, 3)
b <- c(4, 5, 6)
c <- c(7, 8, 9)
combined_matrix <- matrix(c(a, b, c), ncol=3, byrow  = TRUE)
print(combined_matrix)
```

13.

```
set.seed(123)
num_samples <- 10
mean_value <- 0
sd_value <- 1
random_numbers <- rnorm(num_samples, mean = mean_value, sd = sd_value)
```

```
value_counts <- table(random_numbers)
cat("Generated random numbers:\n")
print(random_numbers)

cat("\nOccurrences of each value:\n")
print(value_counts)
```

14.

```
data= read.csv("20th R program..csv")
print(data)
```

15.

```
numeric_vector <- c(1.5, 2.7, 3.2, 4.9)
character_vector <- c("apple", "banana", "cherry",
"date")
logical_vector <- c(TRUE, FALSE, TRUE, FALSE)
cat("Numeric Vector:\n")
print(numeric_vector)
cat("Character Vector:\n")
print(character_vector)
cat("Logical Vector:\n")
print(logical_vector)
```

16.

```
matrix_5x4 <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE)

rownames(matrix_5x4) <- c("Row1", "Row2", "Row3", "Row4", "Row5")

colnames(matrix_5x4) <- c("Col1", "Col2", "Col3", "Col4")


matrix_3x3 <- matrix(11:19, nrow = 3, ncol = 3)

rownames(matrix_3x3) <- c("R1", "R2", "R3")

colnames(matrix_3x3) <- c("C1", "C2", "C3")


matrix_2x2 <- matrix(101:104, nrow = 2, ncol = 2, byrow = FALSE)

rownames(matrix_2x2) <- c("Row_A", "Row_B")

colnames(matrix_2x2) <- c("Col_X", "Col_Y")


cat("5x4 Matrix (filled by rows):\n")

print(matrix_5x4)


cat("\n3x3 Matrix (with labels):\n")
```

```
print(matrix_3x3)


cat("\n2x2 Matrix (filled by column):\n")
print(matrix_2x2)
```

17.

```
values <- 1:24
dimensions <- c(3, 4, 3)
dim_names <- list(
  c("Row1", "Row2", "Row3"),
  c("Col1", "Col2", "Col3", "Col4"),
  c("Depth1","Depth2","Depth3")
)
my_array <- array(values, dim = dimensions, dimnames = dim_names)
print(my_array)
```

18.

```
vector1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vector2 <- c(10, 11, 12, 13, 14, 15, 16, 17, 18)
combined_matrix <- cbind(vector1, vector2)
my_array <- array(combined_matrix, dim = c(3, 3, 2))
```

```
print(my_array)
```

19.

```
my_list <- list(
  numeric_vector = c(1.5, 2.7, 3.2),
  character_matrix = matrix(c("apple", "banana", "cherry"), nrow = 1),
  logical_matrix = matrix(c(TRUE, FALSE, TRUE), nrow = 1),
  custom_function = function(x) x^2
)
print("Content of the list:")
print(my_list)
```

20.

```
x=c(0,10)
y=c(5,20)
plot(x,y)
```

21.

```
# Create two vectors
vector1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vector2 <- c(10, 11, 12, 13, 14, 15, 16, 17, 18)
```

```
# Reshape vectors into two 3x3 matrices
matrix1 <- matrix(vector1, nrow = 3, byrow = TRUE)
matrix2 <- matrix(vector2, nrow = 3, byrow = TRUE)

# Create an array of the two matrices
matrix_array <- array(c(matrix1, matrix2), dim = c(3, 3, 2))

# Print the second row of the second matrix
cat("Second row of the second matrix:\n")
print(matrix_array[2, , 2])

# Print the element in the 3rd row and 3rd column of the 1st matrix
cat("Element in the 3rd row and 3rd column of the 1st matrix:", matrix_array[3, 3, 1], "\n")
```

22.

```
# Create three example arrays
array1 <- array(1:9, dim = c(3, 3))
array2 <- array(10:18, dim = c(3, 3))
array3 <- array(19:27, dim = c(3, 3))
```

```
# Combine the arrays
combined_array <- array(c(array1[1, ], array2[1, ], array3[1, ]), dim = c(3, 3))

# Print the combined array
print(combined_array)
```

23.

```
# Create the data for the array
column1 <- c(1, 4, 7)
column2 <- c(2, 5, 8)
column3 <- c(3, 6, 9)
column4 <- c(10, 13, 16)
column5 <- c(11, 14, 17)
column6 <- c(12, 15, 18)

# Combine the columns into two tables
table1 <- cbind(column1, column2, column3)
table2 <- cbind(column4, column5, column6)
```

```
# Create a 3-dimensional array using the tables
array_data <- array(c(table1, table2), dim = c(3, 3, 2))

# Display the content of the array
print(array_data)
```

24.

```
# Generate a sequence of even integers greater than 50
even_integers <- seq(from = 52, by = 2, length.out = 5 * 3)

# Reshape the sequence into a 5x3 array
array_5x3 <- matrix(even_integers, nrow = 5, ncol = 3, byrow = TRUE)

# Print the array
print(array_5x3)
```

25.

```
# Create the initial exam_data data frame
exam_data <- data.frame(
```

```
  name = c('Anastasia', 'Dima', 'Katherine', 'James',
'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'),
  score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19),
  attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1),
  qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no',
'no', 'yes')
)

# a. Extract 3rd and 5th rows with 1st and 3rd columns
extracted_data <- exam_data[c(3, 5), c(1, 3)]
print("Extracted data:")
print(extracted_data)

# b. Add a new column named country
Country <- c("USA", "USA", "USA", "USA", "UK", "USA",
"USA", "India", "USA", "USA")
exam_data$country <- Country

# c. Add new row(s) to the existing data frame
```

```
new_exam_data <- data.frame(name = c('Robert', 'Sophia'), score = c(10.5, 9), attempts = c(1, 3), qualify = c('yes', 'no'))

updated_exam_data <- rbind(exam_data, new_exam_data)


# d. Sort the data frame by name and score

sorted_exam_data <- exam_data[order(exam_data$name, exam_data$score), ]


# e. Save the information of the data frame in a file

write.csv(exam_data, "exam_data.csv")


# Display the information of the file

file_contents <- read.csv("exam_data.csv")

print("Contents of the file:")

print(file_contents)
```

26.

```
# Load the built-in dataset airquality

data("airquality")
```

```
# Check if airquality is a data frame
if (is.data.frame(airquality)) {
  print("airquality is a data frame.")
} else {
  print("airquality is not a data frame.")
}

# Order the data frame by the first and second column
ordered_airquality <-
airquality[order(airquality$Month, airquality$Day), ]

# Remove variables 'Solar.R' and 'Wind'
cleaned_airquality <-
ordered_airquality[, !(names(ordered_airquality) %in%
c('Solar.R', 'Wind'))]

# Display the cleaned data frame
print("Cleaned data frame:")
print(cleaned_airquality)
```

27.

```
# Load the built-in women dataset
data("women")

# Create a factor corresponding to the height variable
height_factor <- cut(women$height, breaks = c(55, 60, 65, 70, 75), labels = c("Short", "Average", "Tall", "Very Tall"))

# Print the factor
print(height_factor)
```

28.

```
# Generate a random sample from LETTERS
set.seed(123) # for reproducibility
random_letters <- sample(LETTERS, 100, replace = TRUE)

# Create a factor from the random sample
random_factor <- factor(random_letters)

# Extract the first five levels of the factor
five_levels <- levels(random_factor)[1:5]
```

```
# Print the extracted levels
print(five_levels)
```

29.

```
data("iris")

# Step 1
print(dim(iris))
print(str(iris))
print(summary(iris))
print(apply(iris[, 1:4], 2, sd))

# Step 2
mean_by_species <- aggregate(. ~ Species, data = iris, FUN = mean)
print("Mean of features grouped by species:")
print(mean_by_species)

std_dev_by_species <- aggregate(. ~ Species, data = iris, FUN = sd)
```

```
print("Standard Deviation of features grouped by species:")
print(std_dev_by_species)

# Step 3
quantiles_sepal_width <- quantile(iris$Sepal.Width, probs = c(0.25, 0.5, 0.75))
quantiles_sepal_length <- quantile(iris$Sepal.Length, probs = c(0.25, 0.5, 0.75))

print(quantiles_sepal_width)
print(quantiles_sepal_length)

# Step 4
quantiles <- quantile(iris$Sepal.Length, probs = c(0, 0.25, 0.5, 0.75, 1))

categorize_sepal_length <- function(sepal_length) {
  if (sepal_length <= quantiles[2]) {
    return("Q1")
  } else if (sepal_length <= quantiles[3]) {
```

```
    return("Q2")
  } else if (sepal_length <= quantiles[4]) {
    return("Q3")
  } else {
    return("Q4")
  }
}
iris1 <- iris
iris1$Sepal.Length.Cate <- sapply(iris$Sepal.Length,
categorize_sepal_length)
head(iris1)

# step 5
iris$Sepal.Length.Cate <- sapply(iris$Sepal.Length,
categorize_sepal_length)

avg_by_categories <- aggregate(. ~ Species +
Sepal.Length.Cate, data = iris, FUN = mean)

print("Average value of numerical variables by Species
and Sepal.Length.Cate:")
```

```
print(avg_by_categories)


# Step 6
avg_means_by_categories <- aggregate(. ~ Species + Sepal.Length.Cate, data = iris,
                                     FUN = function(x) mean(x, na.rm = TRUE))


# Print the mean of average values
print("Average mean value of numerical variables by Species and Sepal.Length.Cate:")
print(avg_means_by_categories)


# step 7
pivot_table <- iris %>% group_by(Species, Sepal.Length.Cate) %>% summarise(mean_Sepal.Width = mean(Sepal.Width, na.rm = TRUE),
          mean_Petal.Length = mean(Petal.Length, na.rm = TRUE),
          mean_Petal.Width = mean(Petal.Width, na.rm = TRUE))
```

```
# Print the pivot table
print("Pivot Table based on Species and
Sepal.Length.Cate:")
print(pivot_table)
```

30.

```
# Load necessary packages
if (!require(caret)) {
  install.packages("caret")
  library(caret)
}

if (!require(nnet)) {
  install.packages("nnet")
  library(nnet)
}

data("iris")
set.seed(123)
# Split data (80% train, 20% test)
splitIndex <- createDataPartition(iris$Species, p = 0.8,
list = FALSE)
```

```
train_data <- iris[splitIndex, ]
test_data <- iris[-splitIndex, ]

# Train a multinomial logistic regression model and predict
model <- train(Species ~ Petal.Width + Petal.Length, data = train_data, method = "multinom")
predictions <- predict(model, newdata = test_data)

# Create and print confusion matrix
confusion_matrix <- confusionMatrix(predictions, test_data$Species)
print("Confusion Matrix:")
print(confusion_matrix)
```

31.

```
# Given values
values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)

# (i) Compute mean, median, and mode
mean_value <- mean(values)
median_value <- median(values)
```

```
mode_value <-
as.numeric(names(table(values))[table(values) ==
max(table(values))])


cat("Mean:", mean_value, "\n")

cat("Median:", median_value, "\n")

cat("Mode:", mode_value, "\n")


# (ii) Find 2nd highest and 3rd lowest values

sorted_values <- sort(unique(values), decreasing =
TRUE)

second_highest <- sorted_values[2]

third_lowest <- sorted_values[length(sorted_values) -
2]


cat("2nd Highest Value:", second_highest, "\n")

cat("3rd Lowest Value:", third_lowest, "\n")
```

32.

```
data(airquality)

mean_temp <- sum(airquality$Temp, na.rm = TRUE) /
length(airquality$Temp)
```

```
cat("Mean Temperature:", mean_temp, "\n")

first_five_rows <- head(airquality, n = 5)

print(first_five_rows)

subset_data <- airquality[, !(names(airquality) %in% c("Temp", "Wind"))]# Load necessary libraries
```

33.

```
library(reshape2)

# Load the airquality dataset

data(airquality)

# (i) Summary Statistics of airquality dataset

summary(airquality)

# (ii) Melt airquality dataset and display as long-format data

melted_data <- melt(airquality)

print(melted_data)
```

```
# (iii) Melt airquality data and specify month and day as "ID variables"
melted_data_id <- melt(airquality, id.vars = c("Month", "Day"))
print(melted_data_id)

# (iv) Cast the molten airquality data set with respect to Month and Day features
casted_data <- dcast(melted_data_id, Month + Day ~ variable)
print(casted_data)

# (v) Compute the average of Ozone, Solar.R, Wind, and Temp per month using cast function
average_data <- dcast(melted_data, Month ~ variable, fun.aggregate = mean)
print(average_data)print(subset_data)
coldest_day <- airquality$Day[which.min(airquality$Temp)]
cat("Coldest Day:", coldest_day, "\n")
```

34.

```
# Load necessary libraries
```

```
library(ggplot2)

# Load the airquality dataset
data(airquality)

# (i) Find missing values and replace or drop them
missing_percent <- colMeans(is.na(airquality)) * 100
print(missing_percent)

for (col in names(airquality)) {
  if (missing_percent[col] < 10) {
    mean_value <- mean(airquality[, col], na.rm = TRUE)
    airquality[is.na(airquality[, col]), col] <- mean_value
  } else {
    airquality <- airquality[complete.cases(airquality[,
col]), ]
  }
}

# (ii) Apply linear regression on "Ozone" and "Solar.R"
```

```
model <- lm(Ozone ~ Solar.R, data = airquality)
summary(model)

# (iii) Plot Scatter plot between Ozone and Solar with regression line
ggplot(data = airquality, aes(x = Solar.R, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(x = "Solar.R", y = "Ozone", title = "Scatter Plot with Linear Regression")
```

35.

```
# Load necessary libraries
library(reshape2)

# Load the ChickWeight dataset
data(ChickWeight)

# (i) Order the data frame by weight in ascending order grouped by diet and extract last 6 records
```

```
ordered_data <-
ChickWeight[order(ChickWeight$weight), ]

last_6_records <- tail(ordered_data, 6)

print(last_6_records)


# (ii) a. Melting based on "Chick", "Time", "Diet"
features as ID variables

melted_data <- melt(ChickWeight, id.vars = c("Chick",
"Time", "Diet"))

print(melted_data)


# b. Perform cast function to display the mean value of
weight grouped by Diet

mean_weight_by_diet <- dcast(melted_data, Diet ~ .,
fun.aggregate = mean, value.var = "value")

print(mean_weight_by_diet)


# c. Perform cast function to display the mode of
weight grouped by Diet

library(dplyr)

mode_weight_by_diet <- melted_data %>%

  group_by(Diet) %>%
```

```
  summarize(mode_weight =
as.numeric(names(table(value))[table(value) ==
max(table(value))]))

print(mode_weight_by_diet)
```

36.

```
# Load the built-in ChickWeight dataset

data("ChickWeight")


# Ensure that Diet is treated as a factor

ChickWeight$Diet <- as.factor(ChickWeight$Diet)


# Create a multiple regression model

reg_model <- lm(weight ~ Time + Diet, data =
ChickWeight)


# Summary of the regression model

summary(reg_model)


# b. Predict weight for Time=10 and Diet=1

new_data <- data.frame(Time = 10, Diet = factor(1))
```

```
predicted_weight <- predict(reg_model, newdata =
new_data)

cat("Predicted weight for Time = 10 and Diet = 1:",
predicted_weight, "\n")

# c. Find the error in the model for the same data point
actual_weight <-
ChickWeight$weight[ChickWeight$Time == 10 &
ChickWeight$Diet == 1]
error <- actual_weight - predicted_weight

cat("Error in the model:", error, "\n")
```

37.

```
# Load the built-in ChickWeight dataset
data("ChickWeight")

# Ensure that Diet is treated as a factor
ChickWeight$Diet <- as.factor(ChickWeight$Diet)

# Create a multiple regression model
```

```
reg_model <- lm(weight ~ Time + Diet, data =
ChickWeight)

# Summary of the regression model
summary(reg_model)

# b. Predict weight for Time=10 and Diet=1
new_data <- data.frame(Time = 10, Diet = factor(1))
predicted_weight <- predict(reg_model, newdata =
new_data)

cat("Predicted weight for Time = 10 and Diet = 1:",
predicted_weight, "\n")

# c. Find the error in the model for the same data point
actual_weight <-
ChickWeight$weight[ChickWeight$Time == 10 &
ChickWeight$Diet == 1]
error <- actual_weight - predicted_weight

cat("Error in the model:", error, "\n")
```

38. # Load the built-in ChickWeight dataset

data("ChickWeight")

Ensure that Diet is treated as a factor

ChickWeight$Diet <- as.factor(ChickWeight$Diet)

Create a multiple regression model

reg_model <- lm(weight ~ Time + Diet, data = ChickWeight)

Summary of the regression model

summary(reg_model)

b. Predict weight for Time=10 and Diet=1

new_data <- data.frame(Time = 10, Diet = factor(1))

predicted_weight <- predict(reg_model, newdata = new_data)

cat("Predicted weight for Time = 10 and Diet = 1:", predicted_weight, "\n")

```
# c. Find the error in the model for the same data point
actual_weight <-
ChickWeight$weight[ChickWeight$Time == 10 &
ChickWeight$Diet == 1]
error <- actual_weight - predicted_weight

cat("Error in the model:", error, "\n")
```

39.

```
# Load necessary libraries
library(reshape2)

# Load the ChickWeight dataset
data(ChickWeight)

# (i) Order the data frame by weight in ascending order
grouped by diet and extract last 6 records
ordered_data <-
ChickWeight[order(ChickWeight$weight), ]
last_6_records <- tail(ordered_data, 6)
print(last_6_records)
```

```
# (ii) a. Melting based on "Chick", "Time", "Diet"
features as ID variables

melted_data <- melt(ChickWeight, id.vars = c("Chick",
"Time", "Diet"))

print(melted_data)


# b. Perform cast function to display the mean value of
weight grouped by Diet

mean_weight_by_diet <- dcast(melted_data, Diet ~ .,
fun.aggregate = mean, value.var = "value")

print(mean_weight_by_diet)


# c. Perform cast function to display the mode of
weight grouped by Diet

library(dplyr)

mode_weight_by_diet <- melted_data %>%

  group_by(Diet) %>%

  summarize(mode_weight =
as.numeric(names(table(value))[table(value) ==
max(table(value))]))

print(mode_weight_by_diet)
```

40.

```
# Load necessary libraries
library(reshape2)

# Load the airquality dataset
data(airquality)

# (i) Summary Statistics of airquality dataset
summary(airquality)

# (ii) Melt airquality dataset and display as long-format data
melted_data <- melt(airquality)
print(melted_data)

# (iii) Melt airquality data and specify month and day as "ID variables"
melted_data_id <- melt(airquality, id.vars = c("Month", "Day"))
print(melted_data_id)
```

```
# (iv) Cast the molten airquality data set with respect to Month and Day features

casted_data <- dcast(melted_data_id, Month + Day ~ variable)

print(casted_data)


# (v) Compute the average of Ozone, Solar.R, Wind, and Temp per month using cast function

average_data <- dcast(melted_data, Month ~ variable, fun.aggregate = mean)

print(average_data)
```