Yaiza_SME

2023-01-08

Software matemático y estadístico

Lo primero que tenemos que hacer es importar las librerias que vamos a usar

Como vamos a generar dataframes y datasets aleatorios, vamos a definir una seed para que nos de todo el rato los mismos datos.

```
set.seed(123)
```

A continuación, vamos a definir las funciones para generar la información que vamos a necesitar pasar a las funciones. Para generar un dataset tenemos que indicar en número de filas y de columnas que queremos.

```
create_dataset <- function(num_rows, num_columns) {
  #Creamos un data frame vacío que almacenará los datos del dataset
  data <- data.frame()

  #Generamos los datos del dataset
  for (i in 1:num_rows) {
    row <- data.frame(matrix(nrow = 1, ncol = num_columns))
    colnames(row) <- paste0("V", 1:num_columns)
    for (j in 1:num_columns) {
      # Generamos un número aleatorio para cada columna
      value <- sample(0:100, 1)
      row[, j] <- value
    }
    # Añadimos la fila al dataset
    data <- rbind(data, row)
  }

  #Devolvemos el dataset
  return(data)
}
```

Vamos a generar el dataset y el array que vamos a usar.

```
dataset <- create_dataset(10, 8)
dataset
```

```
##     V1 V2 V3 V4 V5 V6 V7 V8
## 1   30 78 50 13 66 41 49 42
## 2  100 13 24 89 90 68 90 56
## 3   91  8 92 98 71 25  6 41
## 4    8 82 35 77 80 42 75 14
```

```
## 5   31  6  8 40 73 22 26 59
## 6   52  6 52 26 95 37 88 33
## 7   92 68 71 75 62 12 81 96
## 8   90 24 37 20 78 40 46 89
## 9   59 94 15 93  5 71 85 85
## 10  38 30 80 49 33  3 12 68
```

```
array <- sample(1:100, 20, replace=TRUE)
array
```

```
##  [1] 25 52 22 89 32 25 87 35 40 30 12 31 30 64 99 14 93 96 71 67
```

En esta función, recibimos como entrada un vector de tipo numérico y un número de intervalos para implementar la discretización del ropio vector que se da como entrada mediante el método equal width.

```
discretizeEW <- function (x, num.bins) {
  #comprobamos los tipos
  resul <- class(x)=="numeric" | class(x) == "integer"
  resul2 <- class(num.bins) == "integer" | class(num.bins)=="numeric"
  if (resul & resul2) {
    inter <- as.integer((max(x) - min(x))/num.bins)
    cut.points <- c()
    x.discretized <- c()
    aux <- c()
    for (i in 1:(num.bins + 1)) {
      cut.points[i] <- min(x) + inter * (i-1)
    }

    k <- 1
    for (i in 1:(num.bins)){
      data <- c()
      for (j in x) {
          if (j >= cut.points[i] & j <= cut.points[i+1]) {
            data[k] <- j
            k <- k + 1
        }
      }
      x.discretized <- append(x.discretized, data)
      k <- 1
    }
    # print(x.discretized)
    # print(cut.points)
  }
  return(c(x.discretized, cut.points))
}
discretizeEW(array, 3)
```

```
##  [1] 25 22 32 25 35 40 30 12 31 30 14 52 64 67 89 87 99 93 96 71 12 41 70 99
```

En esta función, al igual que en la anterior, recibimos un vector de tipo numérico y un número de intervalos. La diferencia esta que, en este caso, discretizamos el vector con el método equal frequency.

```
discretizeEF <- function (x, num.bins) {
  resul <- class(x)=="numeric" | class(x) == "integer"
  resul2 <- class(num.bins) == "integer" | class(num.bins)=="numeric"
  if (resul & resul2) {
    n = as.integer(length(x) / num.bins)
    x.discretized <- c()
    cut.points = c()
    k <- 1
    for (i in 1:num.bins) {
        data <- c()
        for (j in ((i-1) * n):(i * n)) {
          if (j < length(x)) {
            data[k] <- x[j]
            k <- k + 1
          }
        }
        x.discretized <- append(x.discretized, data)
        k <- 1
        #print(x.discretized)
        cut.points <-  append(cut.points, x[j])
    }
    #print(cut.points)
    x.discretized <- x.discretized[2: length(x.discretized)] #Añade un NA al principio y lo eliminamos
    return(c(x.discretized, cut.points))
  }
}

discretizeEF(array, 3)
```

```
##  [1] 25 52 22 89 32 25 25 87 35 40 30 12 31 31 30 64 99 14 93 96 25 31 96
```

A continuación, vamos a calcular la entropía de un vector numérico. Para ello, vamos a hacer uso de la función contar que, dado el vector numérico, el elemento actual y la lista auxiliar, mira si el elemento está en la lista auxiliar. En caso de que no esté, suma uno al contador. Finalmente, añade el elemento a la lista auxiliar.

Gracias a la función contar, podemos saber cuantos elementos diferentes entre si contiene el array.

```
contar <- function(x, elem, aux) {
  cont <- 0
  for (j in 1:length(x)) {
    act <- x[j]
    if (!elem %in% aux && act == elem) {
      cont <- cont + 1
    }
  }
  aux <- c(aux, elem)
  return(list(cont, aux))
}

entropy <- function(x) {
  tam <- length(x)
  pi <- rep(0, length(unique(x)))
```

```
  aux <- list()
  i <- 0
  for (j in x) {
    if (!j %in% aux) {
      cont_aux <- contar(x, j, aux)
      pi[i + 1] <- cont_aux[[1]]
      aux <- cont_aux[[2]]
      i <- i + 1
    }
  }

  sum <- 0
  for (i in 1:length(pi)) {
    sum <- sum - (pi[i] / tam) * log2(pi[i] / tam)
  }
  return(sum)
}

entropy(array)
```

```
## [1] 4.121928
```

Con la función column_variances, calculamos la varianza que tiene cada columna de un dataframe teniendo en cuenta que el dataframe puede tener varias columnas o una única.

```
column_variances <- function(matriz) {
  if (inherits(matrix, "data.frame")) {
    matrz <- as.matrix(matriz)
  }

  if (length(dim(matriz)) == 2) {
    # Calculamos la media aritmética de cada columna
    means <- colMeans(matriz)
    # Calculamos la varianza de cada columna
    vari <-
      sapply(seq_len(ncol(matriz)), function(i) {
        sum((matriz[, i] - means[i]) ^ 2) / nrow(matriz)
      })
  } else {
    # Si la matriz solo tiene una columna, calculamos la varianza de esa columna
    mean <- mean(matriz)
    vari <- sum((matriz - mean) ^ 2) / length(matriz)
  }

  return(vari)
}

column_variances(dataset)
```

```
## [1]  947.09 1132.09  703.84  933.40  665.21  430.89  949.16  634.41
```

En este caso, calculamos la entropia de cada columna de un dataframe usando la función entropy que hemos implementado con anterioridad

```
column_entropy <- function(df) {
  entropies <- c()
  for (i in 1:length(df)) {
    entropies <- c(entropies, entropy(df[, i]))
  }
  return (entropies)
}
dataset <- create_dataset(3, 11)
column_entropy(dataset)
```

```
##  [1] 1.584963 1.584963 1.584963 1.584963 1.584963 1.584963 1.584963 1.584963
##  [9] 1.584963 1.584963 1.584963
```

Ahora bien, una vez implementadas las funciones column_variances() y colum_entropy(), implementamos la función filter_variables() donde le pasamos el datafreme y los límites por los que queremos filtrar los datos para crear el nuevo dataframe.

```
filter_variables <- function(df, entropy_threshold, variance_threshold) {
  # Calculamos la varianza y la entropía de cada columna
  variances <- column_variances(df)
  entropies <- column_entropy(df)

  # Filtramos las columnas que cumplen con los requisitos
  filtered_columns <- character(0)
  for (i in seq_along(df)) {
    col <- names(df)[i]
    if (entropies[i] >= entropy_threshold && variances[i] >= variance_threshold) {
      filtered_columns <- c(filtered_columns, col)
    }
  }

  # Creamos el nuevo dataset con las columnas filtradas
  filtered_df <- df[filtered_columns]

  return (filtered_df)
}
filter_variables(dataset, 1, 1)
```

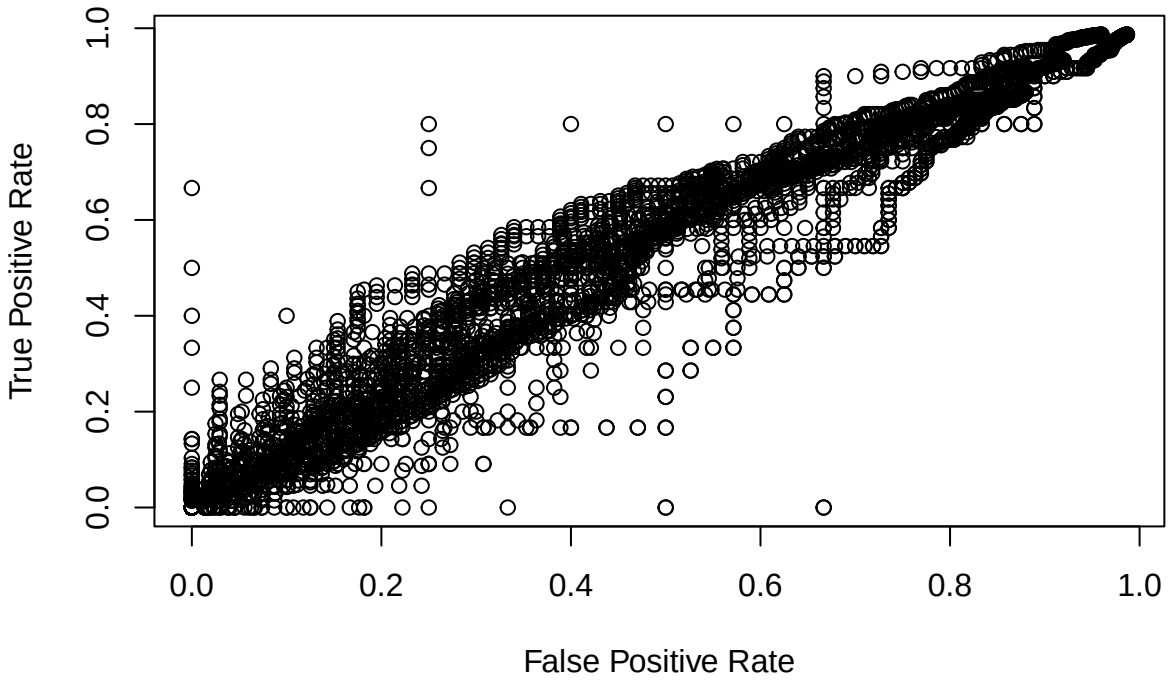
```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 1 22 78 84 36  7 50 73 49 97  73  85
## 2 75 83 45 16 61 45 53 34 93  78  23
## 3 86  6 92 78 22 25 31  6 26  41   4
```

Ahora bien. Vamos a calcular el area bajo la curva ROC. Para ello, vamos a usar la función auxiliar auc_aux() en la que, una vez calculados los ratios de falsos positivos y verdaderos positibos, va calculando y sumando el area bajo la curva. Finalmente, nos sacará el gráfico de la curva dibujado por los valores tprs y fprs.

```
auc_aux <- function(fpr, tpr) {
  area <- 0
  for (i in 1:(length(fpr) - 1)) {
    base_mayor <- fpr[i + 1] - fpr[i]
```

```
    base_menor <- tpr[i + 1] - tpr[i]
    altura <- tpr[i + 1]
    area <- area + (base_mayor + base_menor) * altura / 2
  }
  return(area)
}

AUC <- function(df) {
  df <- df[order(df$value), ]

  min_value <- min(df$value)
  max_value <- max(df$value)

  cutoffs <- seq(min_value, max_value, length.out = 50)

  predictions <- list()
  labels <- list()
  tprs <- c()
  fprs <- c()

  tp_total <- 0
  tn_total <- 0
  fp_total <- 0
  fn_total <- 0

  for (cutoff in cutoffs) {
    prediction <- ifelse(df$value > cutoff, 1, 0)
    predictions <- c(predictions, prediction)
    labels <- c(labels, df$label)

    for (i in 1:nrow(df)) {
      row <- df[i, ]
      prediction <- ifelse(row$value > cutoff, 1, 0)
      label <- row$label
      tp_total <- tp_total + (prediction == 1 & label == 1)
      tn_total <- tn_total + (prediction == 0 & label == 0)
      fp_total <- fp_total + (prediction == 1 & label == 0)
      fn_total <- fn_total + (prediction == 0 & label == 1)
      if (tp_total + fn_total == 0 || is.na(tp_total + fn_total)) {
        tpr <- 0
      } else {
        tpr <- tp_total / (tp_total + fn_total)
      }

      if (fp_total + tn_total == 0 || is.na(fp_total + tn_total)) {
        fpr <- 0
      } else {
        fpr <- fp_total / (fp_total + tn_total)
      }
      tprs <- c(tprs, tpr)
      fprs <- c(fprs, fpr)
    }
```

```
    tprs <- tprs[order(fprs)]
    fprs <- sort(fprs)
    tp_total <- 0
    tn_total <- 0
    fp_total <- 0
    fn_total <- 0
  }

  plot(fprs, tprs, xlab = 'False Positive Rate', ylab = 'True Positive Rate')
  roc_auc <- auc_aux(tprs, fprs)
  print(roc_auc)
}
rows <- data.frame(value = sample(0:100, 150, replace = TRUE),
                   label = sample(c(TRUE, FALSE), 150, replace = TRUE))

df <- rows

AUC(df)
```




```
## [1] 0.4665142
```

Ahora bien, vamos a normalizar los datos del dataset. Para ello, tenemos que tener en cuenta que los datos pueden ser de tipo numeric o categorical. En cualquier caso, para poder normalizarlos o estandarizarlos, los datos tienen que ser numéricos. Dependiendo de la opción que se quiera, es decir, normalizar o estandarizar, se realizara de forma diferente.

```r
normalize_dataset <- function(dataset, option) {
  min_value <- min(dataset[1])
  max_value <- max(dataset[1])
  mean_value <- sum(dataset[1]) / length(dataset[1])
  std_value <- sqrt(sum(sapply(dataset[1], function(x) {(x - mean_value) ^ 2})) / length(dataset[1]))
  for (i in 1:nrow(dataset)) {
    for (j in 1:ncol(dataset)) {
      if (is.numeric(dataset[i, j])) {
        if (option == "normalize") {
          dataset[i, j] <- round((dataset[i, j] - min_value) / (max_value - min_value), 4)
        } else if (option == "standardize") {
          dataset[i, j] <- round((dataset[i, j] - mean_value) / std_value, 4)
        }
      }
    }
  }
  return(dataset)
}
```

Vamos a ver que nos devuelve la función normalize_dataset dependiendo de la opción que se especifique.

```r
normalize_dataset(dataset, "normalize")
```

```
##       V1      V2     V3      V4      V5     V6     V7      V8     V9    V10
## 1 0.0000  0.8750 0.9688  0.2188 -0.2344 0.4375 0.7969  0.4219 1.1719 0.7969
## 2 0.8281  0.9531 0.3594 -0.0938  0.6094 0.3594 0.4844  0.1875 1.1094 0.8750
## 3 1.0000 -0.2500 1.0938  0.8750  0.0000 0.0469 0.1406 -0.2500 0.0625 0.2969
##       V11
## 1  0.9844
## 2  0.0156
## 3 -0.2812
```

```r
normalize_dataset(dataset, "standardize")
```

```
##        V1      V2      V3      V4      V5      V6      V7      V8      V9
## 1 -0.7427 -0.4844 -0.4567 -0.6781 -0.8119 -0.6135 -0.5074 -0.6181 -0.3967
## 2 -0.4982 -0.4613 -0.6366 -0.7704 -0.5628 -0.6366 -0.5997 -0.6873 -0.4152
## 3 -0.4475 -0.8165 -0.4198 -0.4844 -0.7427 -0.7288 -0.7012 -0.8165 -0.7242
##       V10     V11
## 1 -0.5074 -0.4521
## 2 -0.4844 -0.7381
## 3 -0.6550 -0.8257
```

Para ir terminando, vamos a calcular la correlación o la información mutua del tipo de la columna con la columna adyacente. Es decir, por pares. Para ello, necesitamos saber en primera instancia que tipo de datos manejamos. Una vez que sabemos esto, podemos proceder a calcular la correlación o la información mutua. Para saber si calculamos una o la otra, nos fijamos en los tipos de datos que tenemos en la columna. Si es categorical, calculamos la información mutua. En caso contrario, la correlación.

```r
get_column_type <- function(dataset, column) {
  # Tomamos una muestra de los valores de la columna
```

```
  sample <- dataset[, column]

  # Comprobamos si todos los elementos de la muestra son enteros
  if (all(is.integer(sample)) || all(is.numeric(sample)) ) {
    # Si todos son enteros, comprobamos si todos los enteros son valores posibles de la columna
    unique_values <- unique(sample)
    if (all(0 <= unique_values & unique_values < length(unique_values))) {
      # Si todos los enteros son valores posibles de la columna, entonces es una columna categórica
      return("categorical")
    }
  }

  # Si no es una columna categórica, entonces es numérica
  return("numerical")
}

calculate_correlation <- function(dataset) {
  #Determinamos el tipo de cada columna
  column_types <-
    lapply(seq_along(dataset), function(i)
      get_column_type(dataset, i))

  #Calculamos la correlación o la información mutua entre todos los pares de columnas
  correlations <- list()
  for (i in seq_along(dataset)) {
    for (j in (i + 1):length(dataset)) {
      if (j > length(column_types)) {
        next
      }
      column_i <- dataset[, i]
      column_j <- dataset[, j]
      if (column_types[[i]] == "numerical" &&
          column_types[[j]] == "numerical") {
        # Calculamos la correlación entre dos columnas numéricas
        mean_i <- mean(column_i)
        stdev_i <- sd(column_i)
        mean_j <- mean(column_j)
        stdev_j <- sd(column_j)
        correlation <-
          sum(mapply(function(x, y)
            (x - mean_i) * (y - mean_j), column_i, column_j)) / (stdev_i * stdev_j)
        correlations <- c(correlations, list(c(i, j, correlation)))
      } else if (column_types[[i]] == "categorical" &&
                 column_types[[j]] == "categorical") {
        # Calculamos la información mutua entre dos columnas categóricas
        values_i <- unique(column_i)
        values_j <- unique(column_j)
        mutual_information <- 0
        for (value_i in values_i) {
          for (value_j in values_j) {
            p_i <- sum(column_i == value_i) / length(column_i)
            p_j <- sum(column_j == value_j) / length(column_j)
            p_ij <-
```

```
              sum((column_i == value_i) &
                    (column_j == value_j)) / length(column_i)
            mutual_information <-
              mutual_information + p_ij * log(p_ij / (p_i * p_j), base = 2)
          }
        }
        correlations <- c(correlations, list(c(i, j, mutual_information)))
      }
    }
  }
  return(correlations)
}
cor <- calculate_correlation(dataset)

cor
```

```
## [[1]]
## [1]  1.000000  2.000000 -1.173383
##
## [[2]]
## [1]  1.0000000  3.0000000 -0.4020912
##
## [[3]]
## [1] 1.0000000 4.0000000 0.7110873
##
## [[4]]
## [1] 1.000000 5.000000 1.185697
##
## [[5]]
## [1]  1.000000  6.000000 -1.535279
##
## [[6]]
## [1]  1.000000  7.000000 -1.850077
##
## [[7]]
## [1]  1.000000  8.000000 -1.720558
##
## [[8]]
## [1]  1.000000  9.000000 -1.341451
##
## [[9]]
## [1]  1.000000 10.000000 -1.062654
##
## [[10]]
## [1]  1.000000 11.000000 -1.995806
##
## [[11]]
## [1]  2.000000  3.000000 -1.350648
##
## [[12]]
## [1]  2.000000  4.000000 -1.930983
##
## [[13]]
```

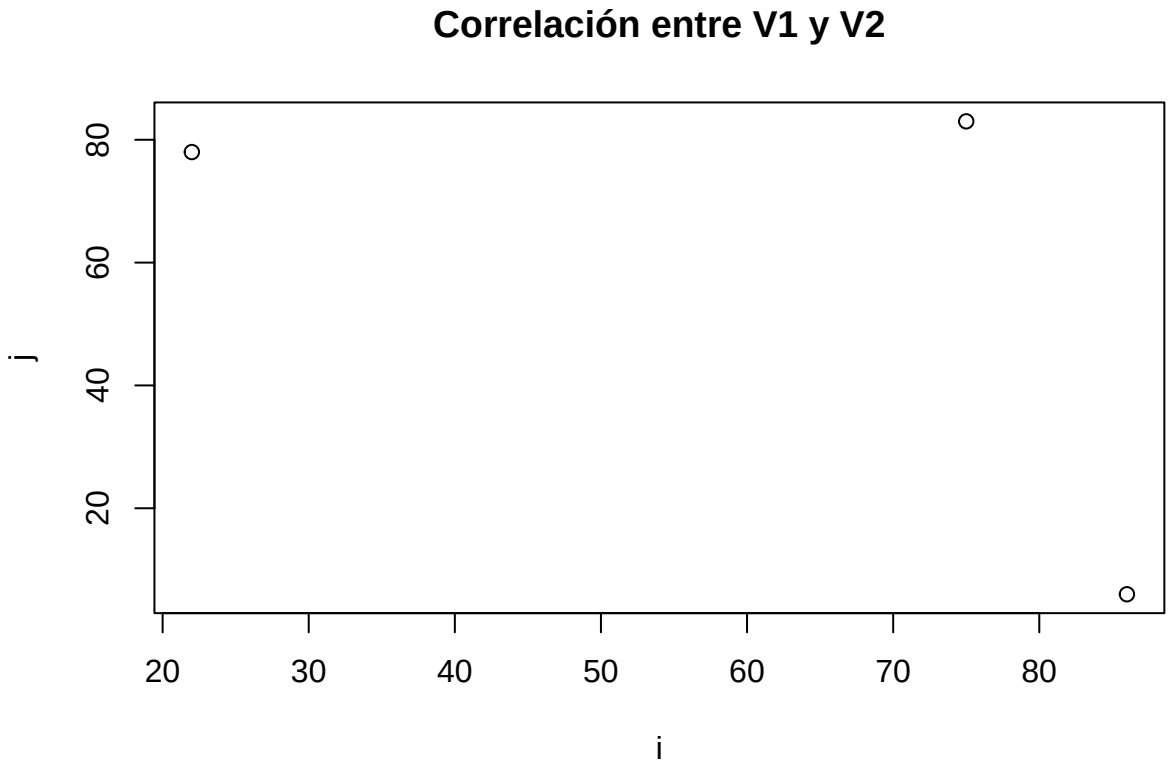
```
## [1] 2.0000000 5.0000000 0.6086658
##
## [[14]]
## [1] 2.000000 6.000000 1.938721
##
## [[15]]
## [1] 2.000000 7.000000 1.700675
##
## [[16]]
## [1] 2.000000 8.000000 1.835158
##
## [[17]]
## [1] 2.0000 9.0000 1.9883
##
## [[18]]
## [1]  2.000000 10.000000  1.995539
##
## [[19]]
## [1]  2.00000 11.00000  1.27575
##
## [[20]]
## [1] 3.00000 4.00000 1.68819
##
## [[21]]
## [1]  3.000000  5.000000 -1.816121
##
## [[22]]
## [1]  3.0000000  6.0000000 -0.9469313
##
## [[23]]
## [1]  3.000000  7.000000 -0.372284
##
## [[24]]
## [1]  3.0000000  8.0000000 -0.6529181
##
## [[25]]
## [1]  3.00000  9.00000 -1.18343
##
## [[26]]
## [1]  3.000000 10.000000 -1.446097
##
## [[27]]
## [1]  3.000000 11.000000  0.274444
##
## [[28]]
## [1]  4.000000  5.000000 -1.083822
##
## [[29]]
## [1]  4.000000  6.000000 -1.743872
##
## [[30]]
## [1]  4.000000  7.000000 -1.367888
##
## [[31]]
```

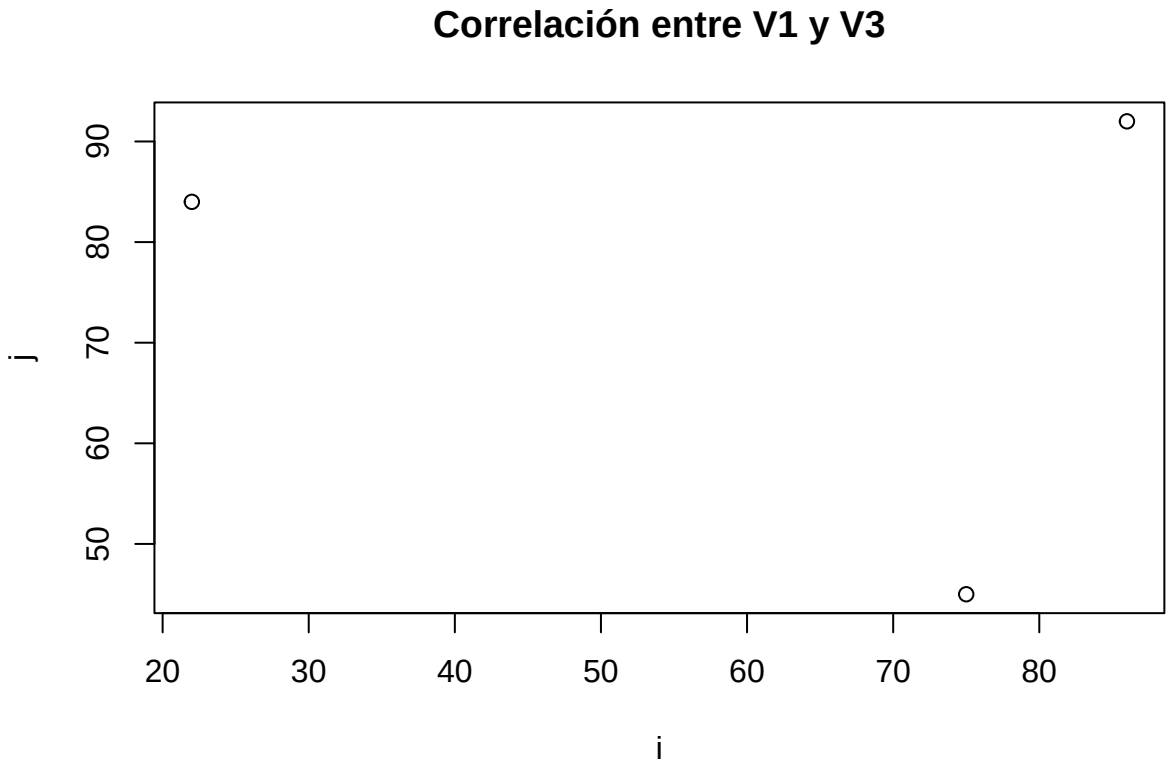
```
## [1]  4.000000  8.000000 -1.564757
##
## [[32]]
## [1]  4.000000  9.000000 -1.863429
##
## [[33]]
## [1]  4.000000 10.000000 -1.961445
##
## [[34]]
## [1]  4.0000000 11.0000000 -0.8305854
##
## [[35]]
## [1] 5.0000000 6.0000000 0.1220346
##
## [[36]]
## [1]  5.0000000  7.0000000 -0.4849807
##
## [[37]]
## [1]  5.0000000  8.0000000 -0.1988929
##
## [[38]]
## [1] 5.000000 9.000000 0.399335
##
## [[39]]
## [1]  5.000000 10.000000  0.734478
##
## [[40]]
## [1]  5.000000 11.000000 -1.078965
##
## [[41]]
## [1] 6.0000 7.0000 1.9071
##
## [[42]]
## [1] 6.000000 8.000000 1.974242
##
## [[43]]
## [1] 6.000000 9.000000 1.980442
##
## [[44]]
## [1]  6.000000 10.000000  1.901603
##
## [[45]]
## [1]  6.00000 11.00000  1.61502
##
## [[46]]
## [1] 7.000000 8.000000 1.978919
##
## [[47]]
## [1] 7.000000 9.000000 1.804402
##
## [[48]]
## [1]  7.000000 10.000000  1.626628
##
## [[49]]
```

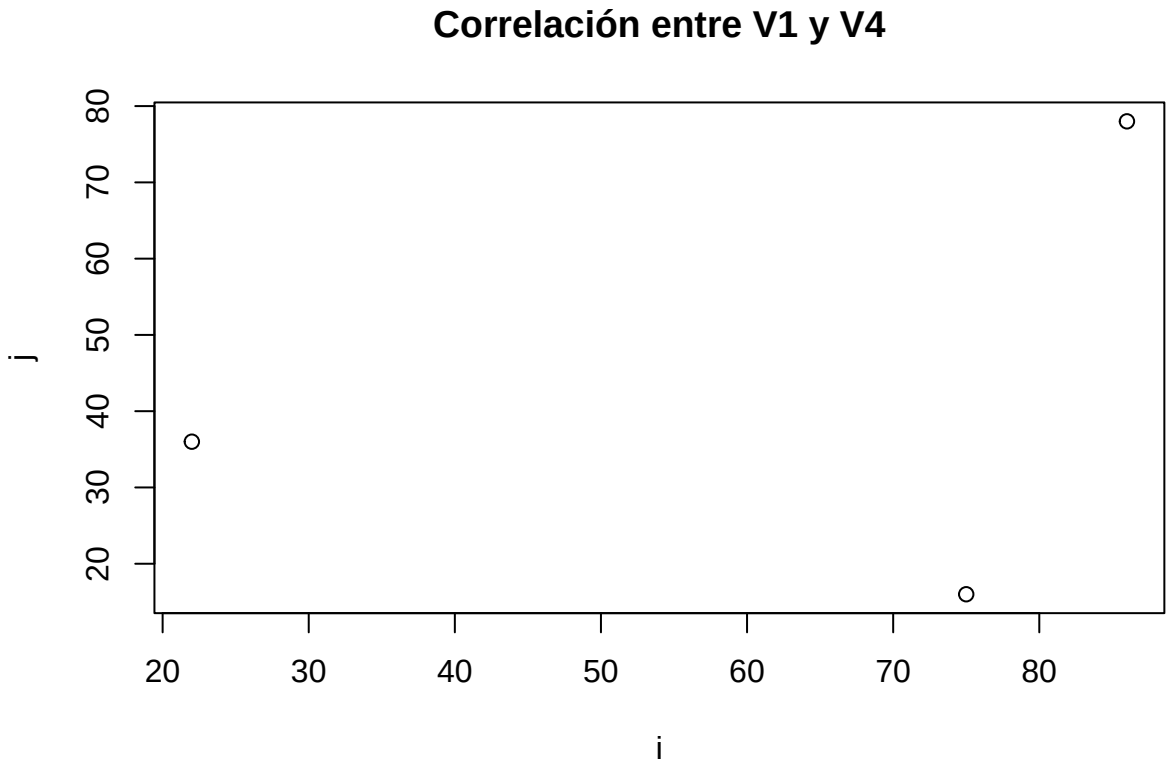
```
## [1]  7.000000 11.000000  1.895371
##
## [[50]]
## [1] 8.0000 9.0000 1.9103
##
## [[51]]
## [1]  8.000000 10.000000  1.777991
##
## [[52]]
## [1]  8.000000 11.000000  1.782945
##
## [[53]]
## [1]  9.000000 10.000000  1.969446
##
## [[54]]
## [1]  9.00000 11.00000  1.43465
##
## [[55]]
## [1] 10.000000 11.000000  1.170089
##
## [[56]]
## [1] 11 11  2
```

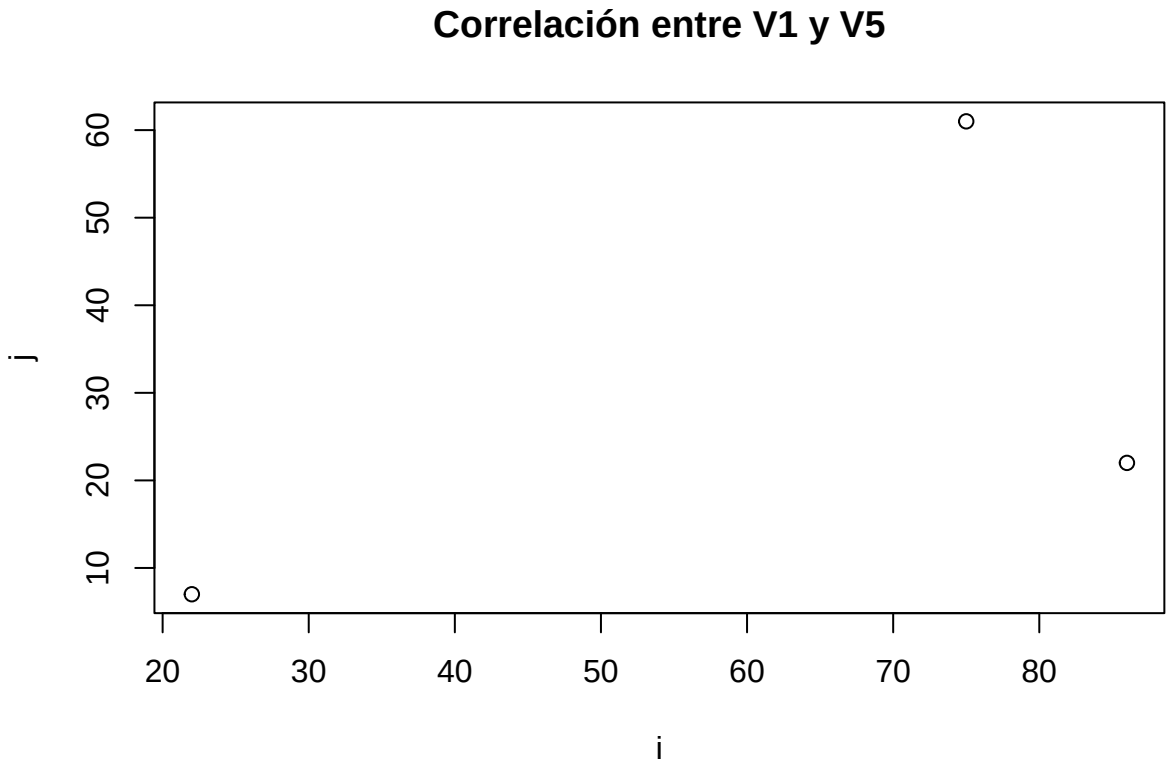
Finalmente, obtenemos gráficamente lo calculado en la función anterior.

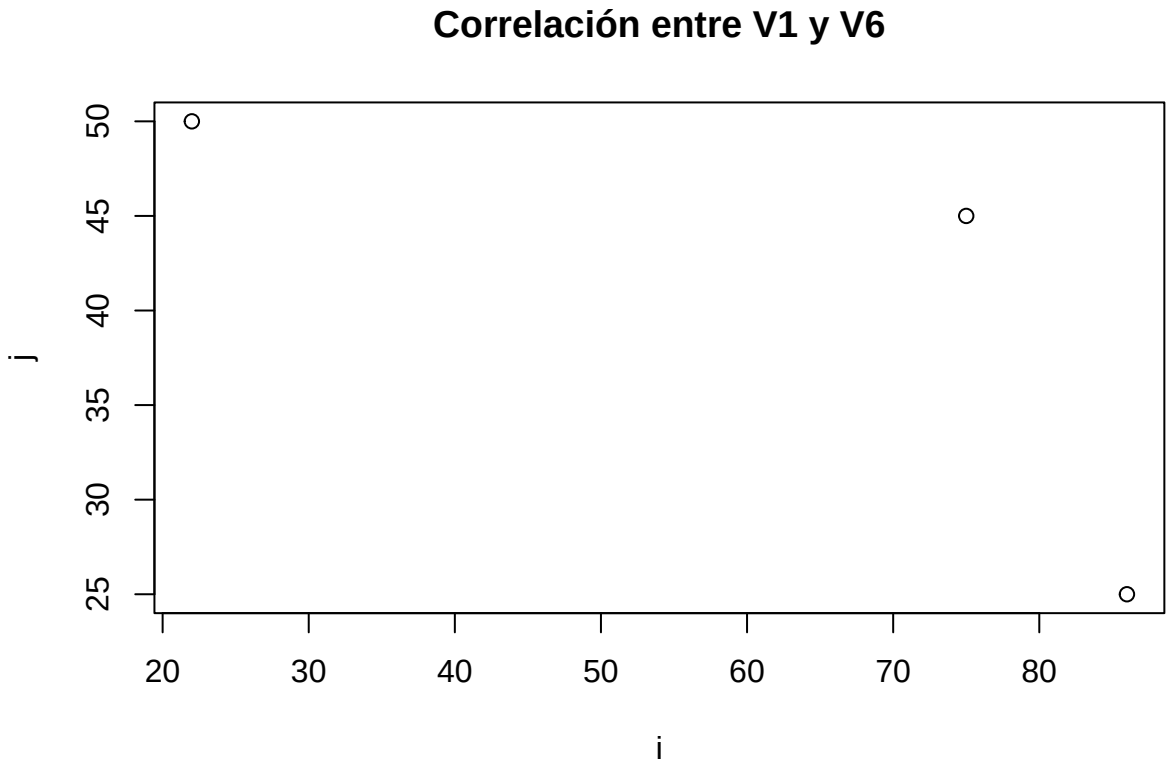
```
plot_corr_mutual_info <- function(data) {
  correlations <- calculate_correlation(data)

  # Creamos una lista vacía para almacenar los gráficos
  imgs <- list()

  # Recorremos cada tupla de correlación
  for (correlation in correlations) {
    i <- data[, correlation[1]]
    j <- data[, correlation[2]]

    # Comprobamos que las columnas son numéricas
    if(is.numeric(i) && is.numeric(j)){
      # Crea el gráfico y lo añade a la lista
      img <- plot(i, j, main = paste("Correlación entre",names(data)[correlation[1]], "y", names(data)[
      imgs <- c(imgs, list(img))
    }
  }

  # Iteramos por cada gráfico para mostrarlos
  lapply(imgs, print)
}

plot_corr_mutual_info(dataset)
```

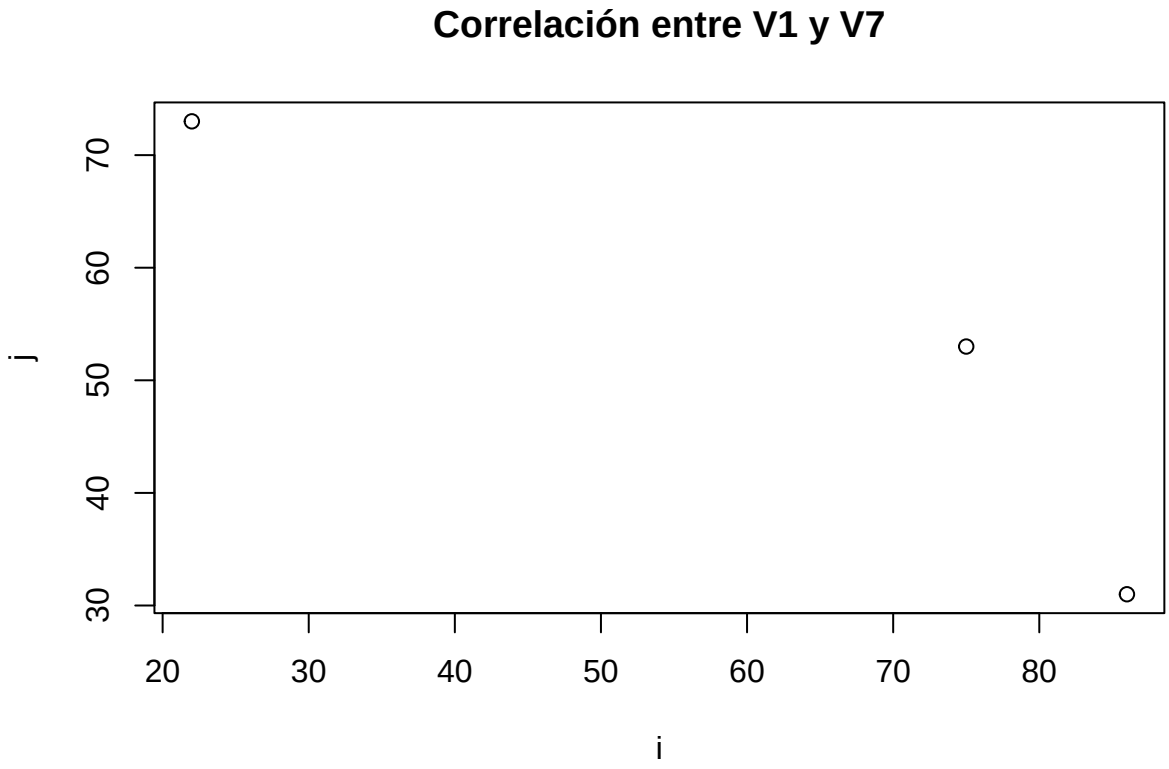

Correlación entre V1 y V2
j
20
40
60
80
20
30
40
50
60
70
80
i

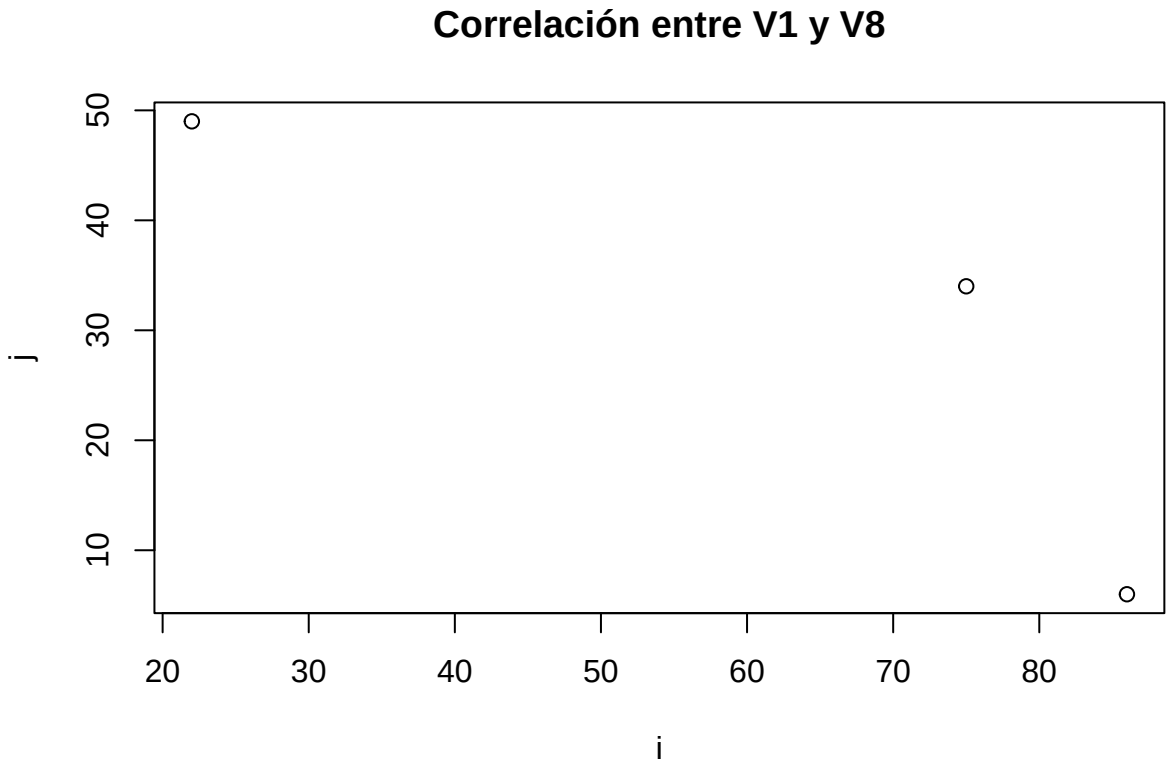

Correlación entre V1 y V3
j
90
80
70
60
50
20
30
40
50
60
70
80
i

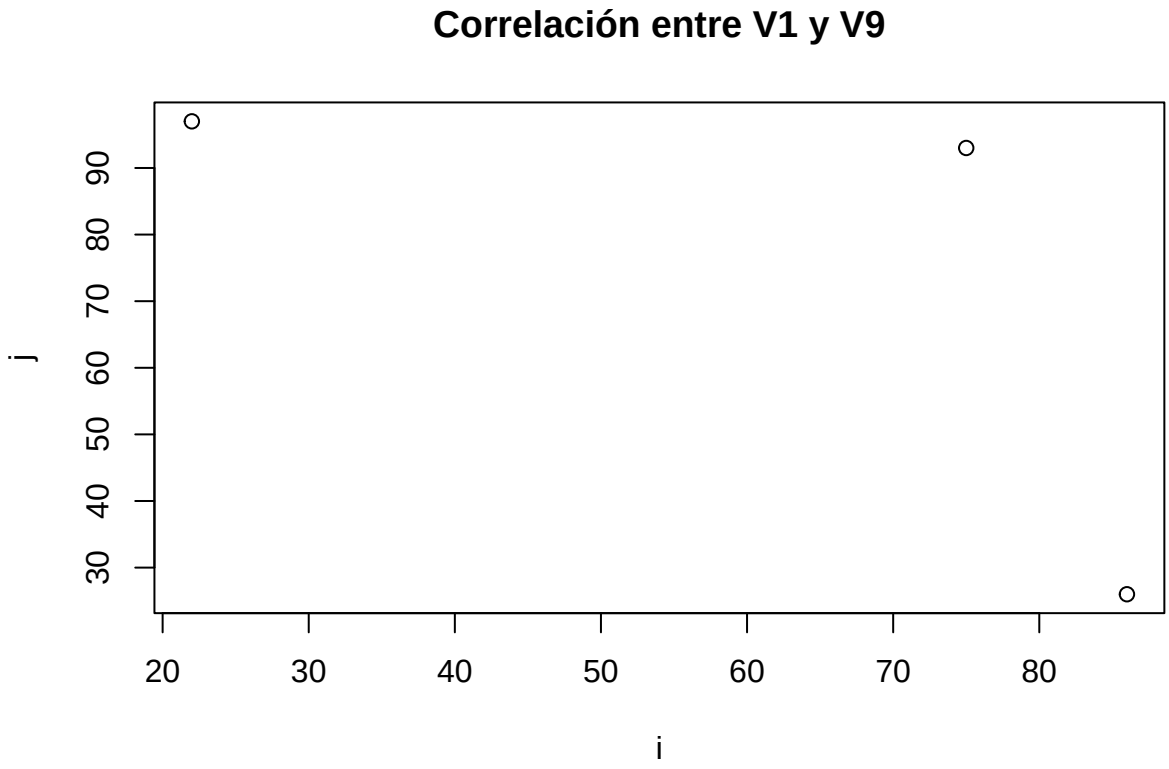

Correlación entre V1 y V4
j
20
30
40
50
60
70
80
i
20
30
40
50
60
70
80

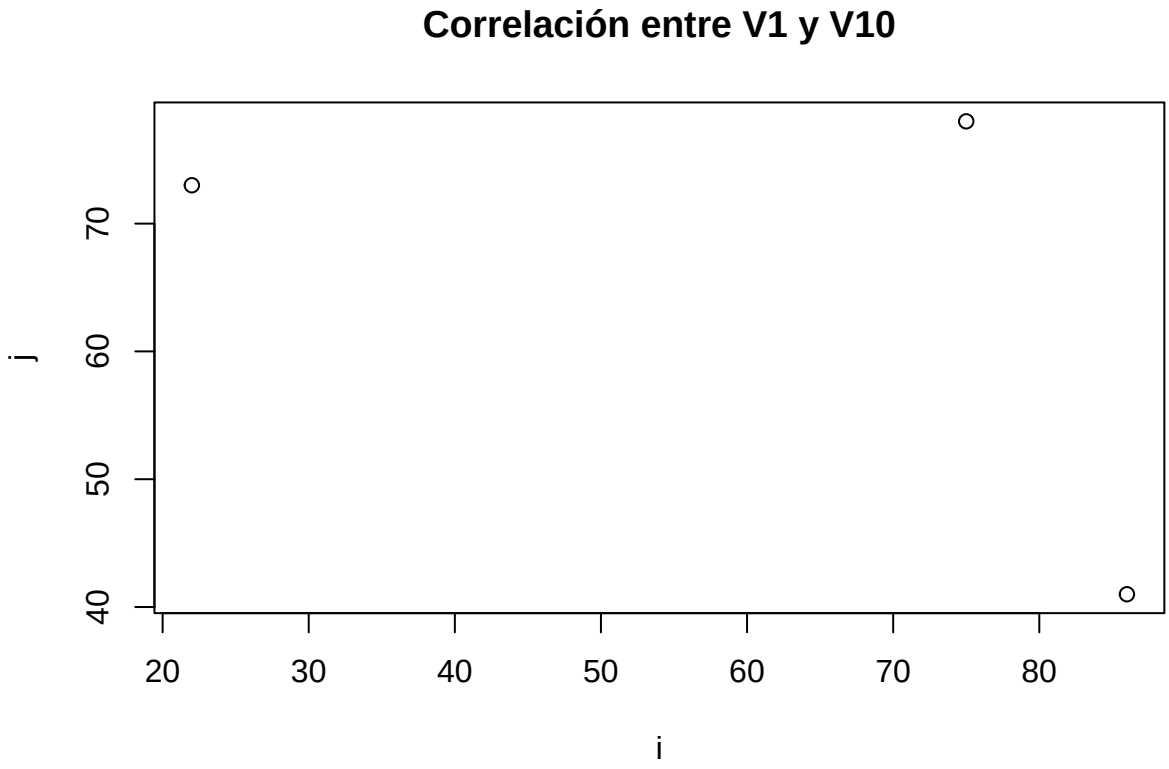

Correlación entre V1 y V5
j
10
20
30
40
50
60
20
30
40
50
60
70
80
i

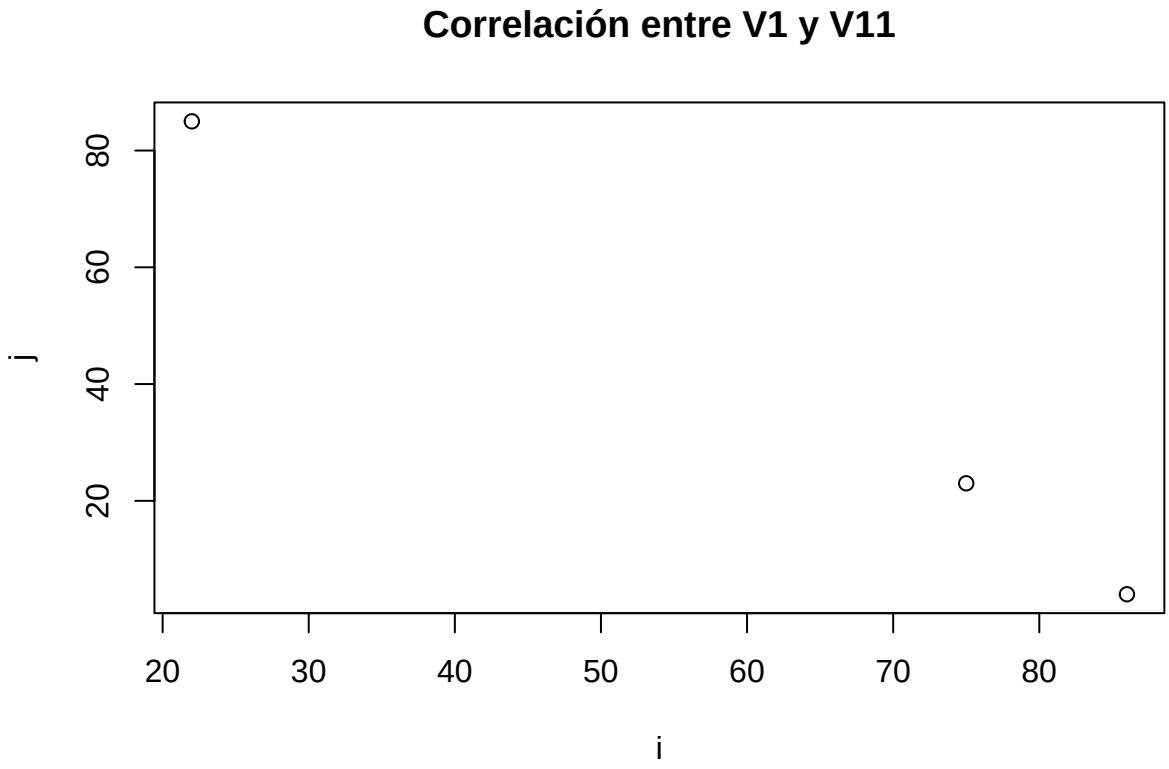

Correlación entre V1 y V6
j
25
30
35
40
45
50
20
30
40
50
60
70
80
i

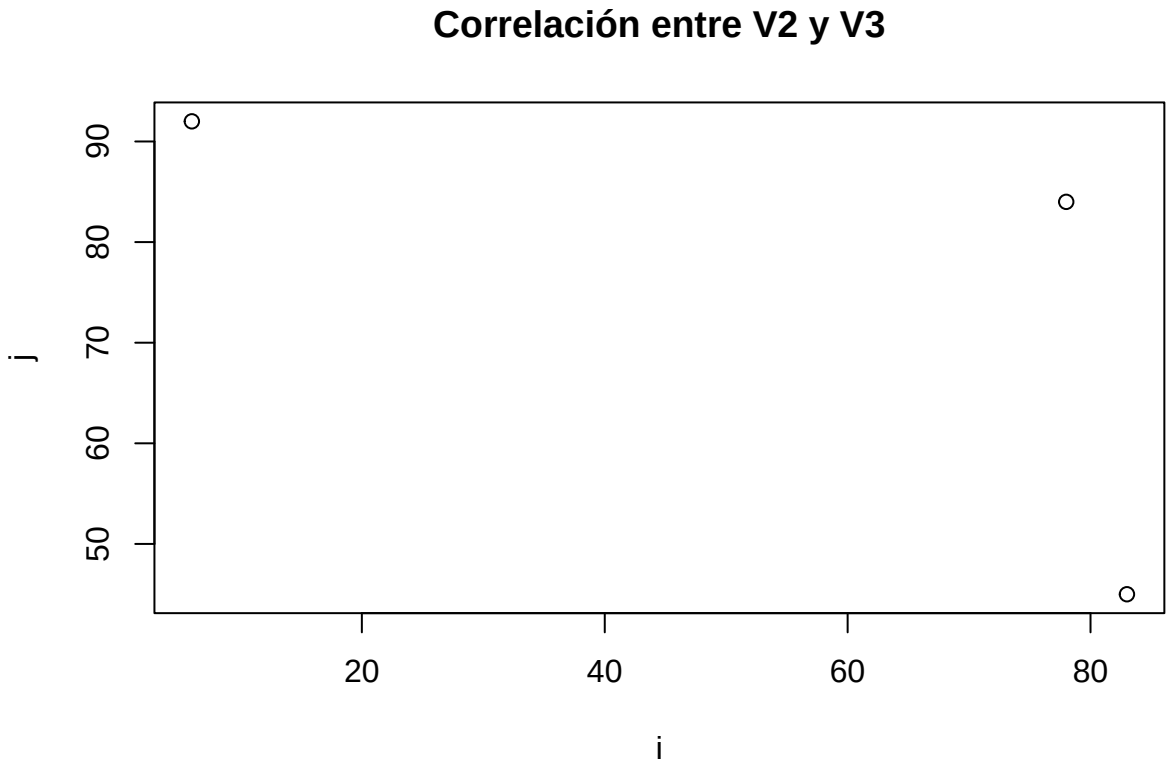

Correlación entre V1 y V7
j
30
40
50
60
70
20
30
40
50
60
70
80
i

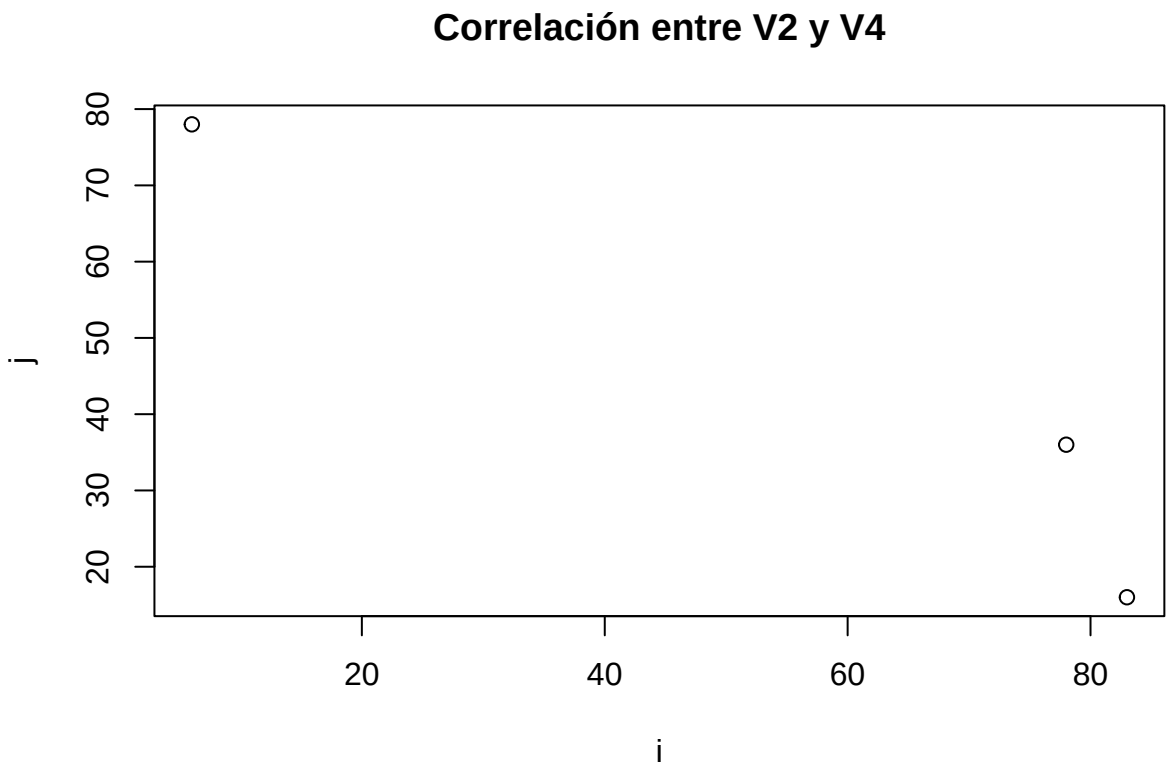

Correlación entre V1 y V8
j
10
20
30
40
50
20
30
40
50
60
70
80
i

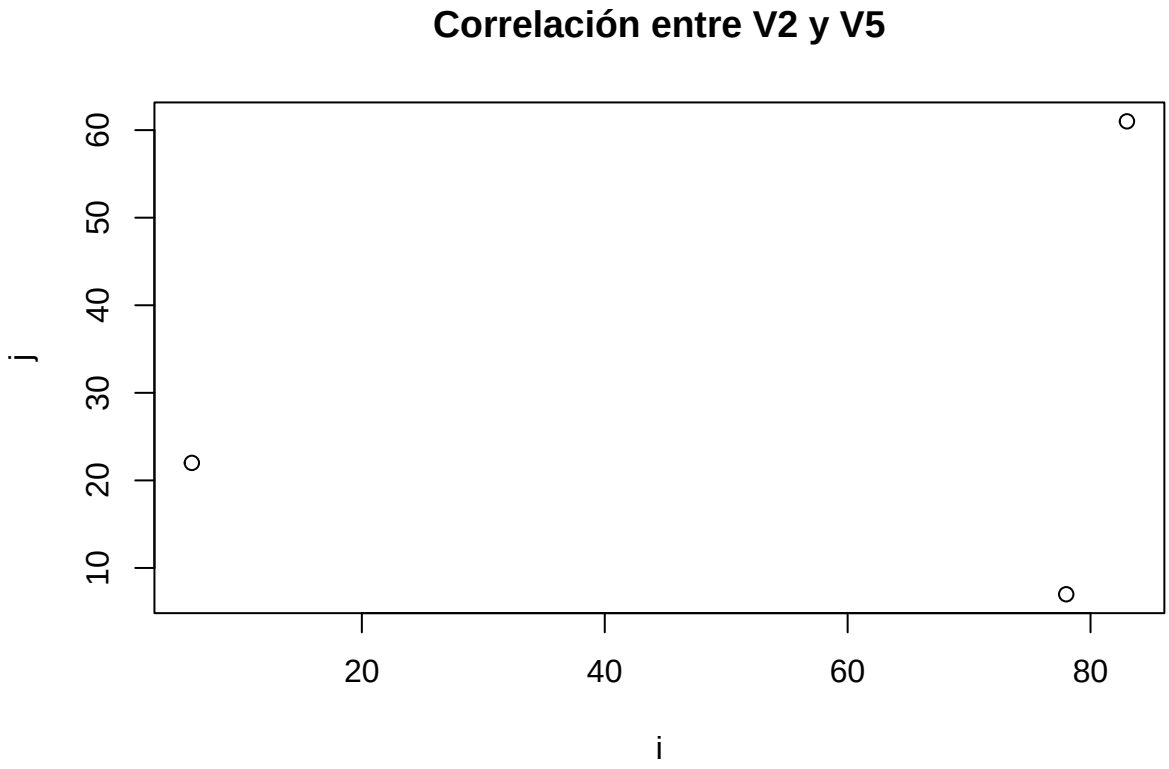

Correlación entre V1 y V9
j
30
40
50
60
70
80
90
20
30
40
50
60
70
80
i

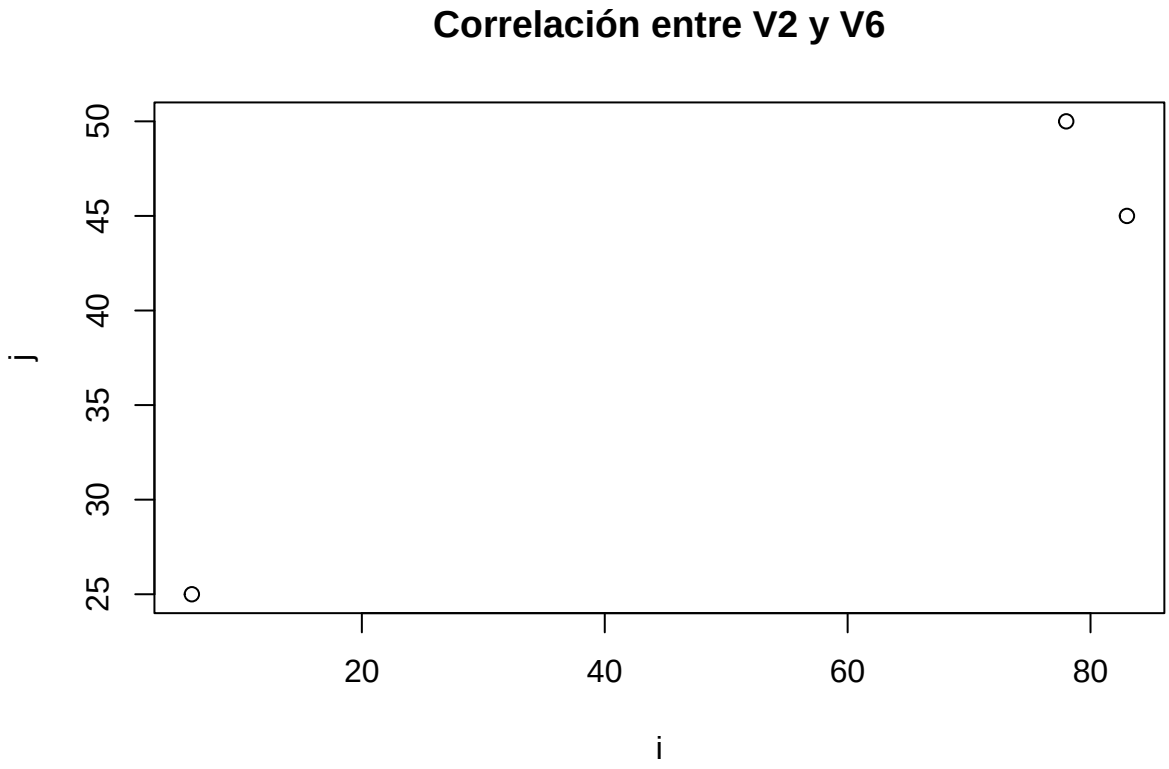

Correlación entre V1 y V10
j
40
50
60
70
20
30
40
50
60
70
80
i

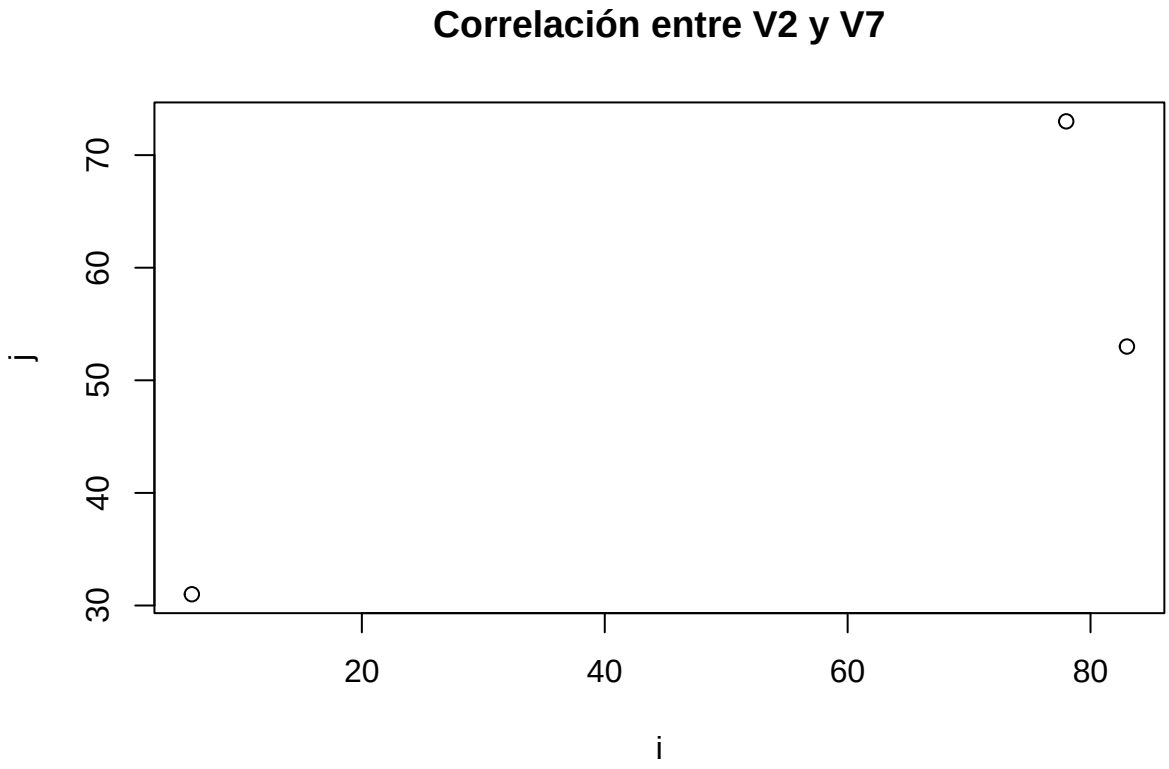

Correlación entre V1 y V11
j
20
40
60
80
i
20
30
40
50
60
70
80

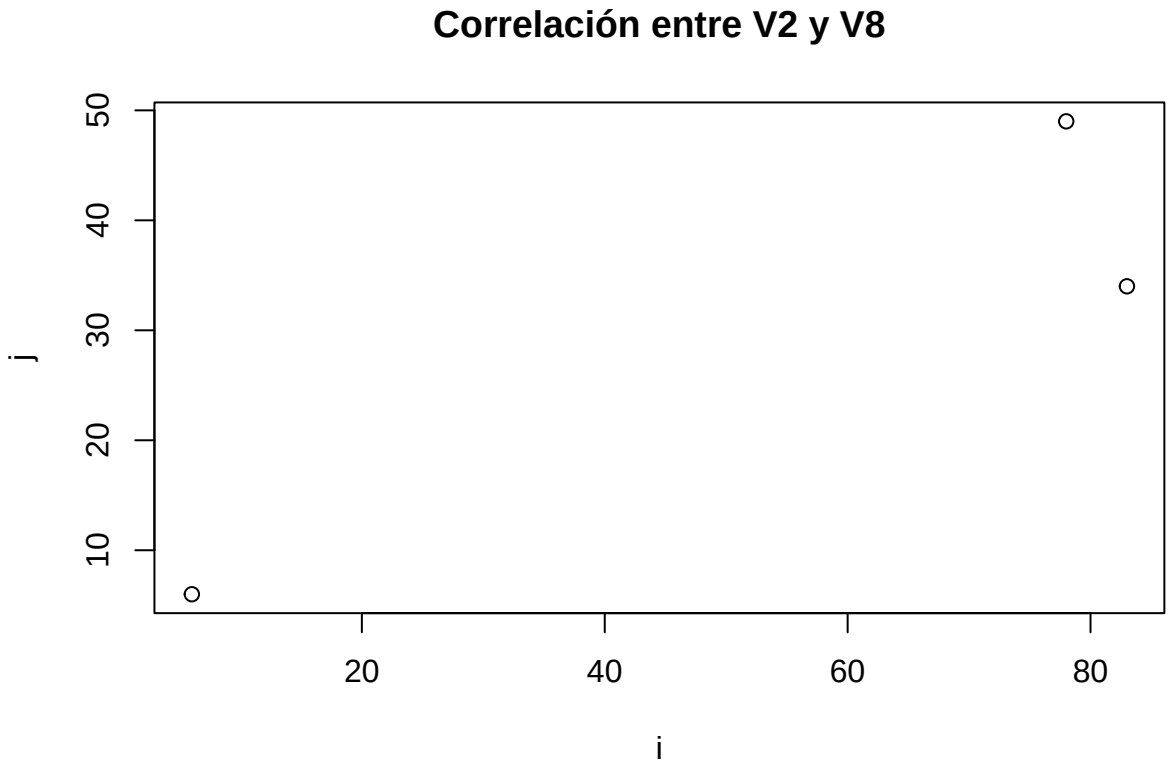

Correlación entre V2 y V3
j
90
80
70
60
50
20
40
60
80
i

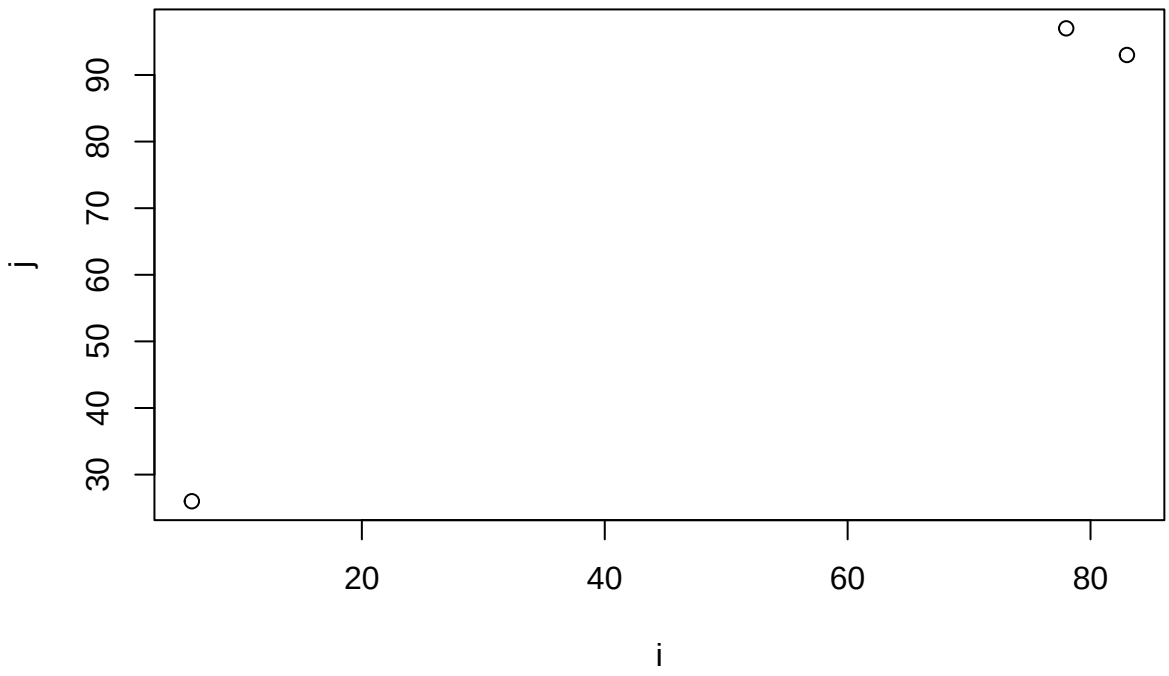

Correlación entre V2 y V4
j
20
30
40
50
60
70
80
20
40
60
80
i

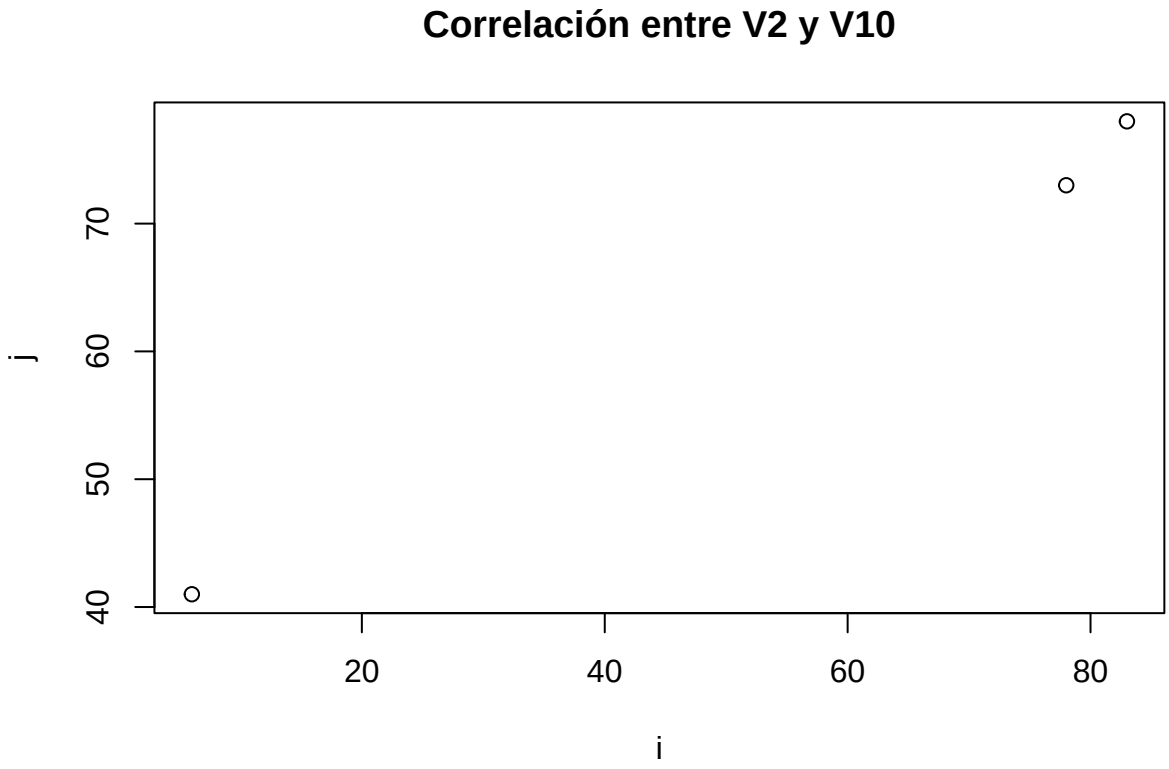

Correlación entre V2 y V5
j
10
20
30
40
50
60
20
40
60
80
i

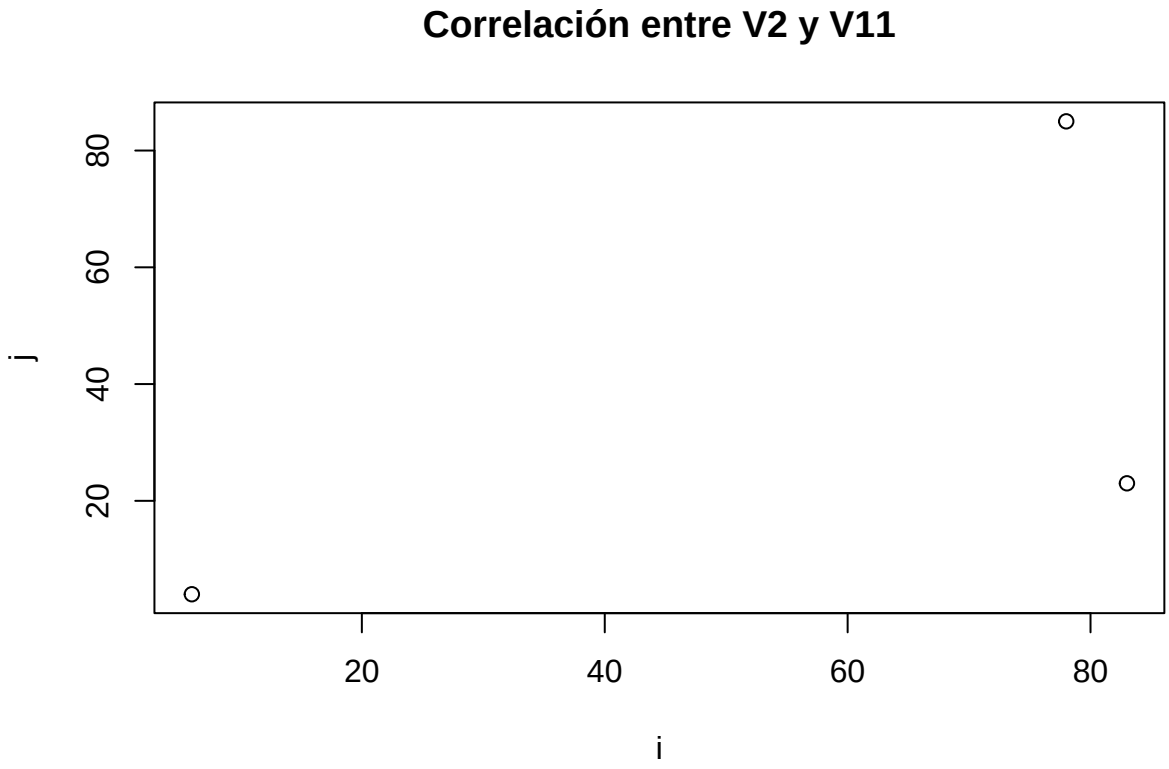

Correlación entre V2 y V6
j
25
30
35
40
45
50
20
40
60
80
i

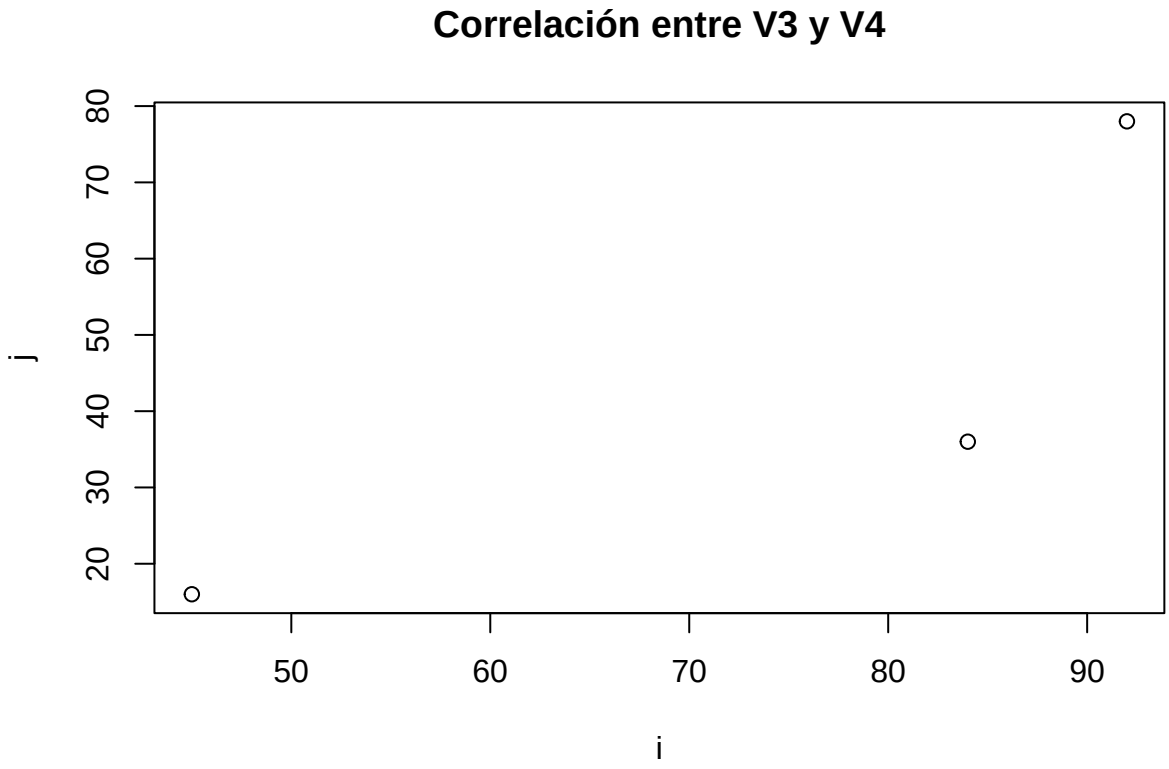

Correlación entre V2 y V7
j
30
40
50
60
70
20
40
60
80
i

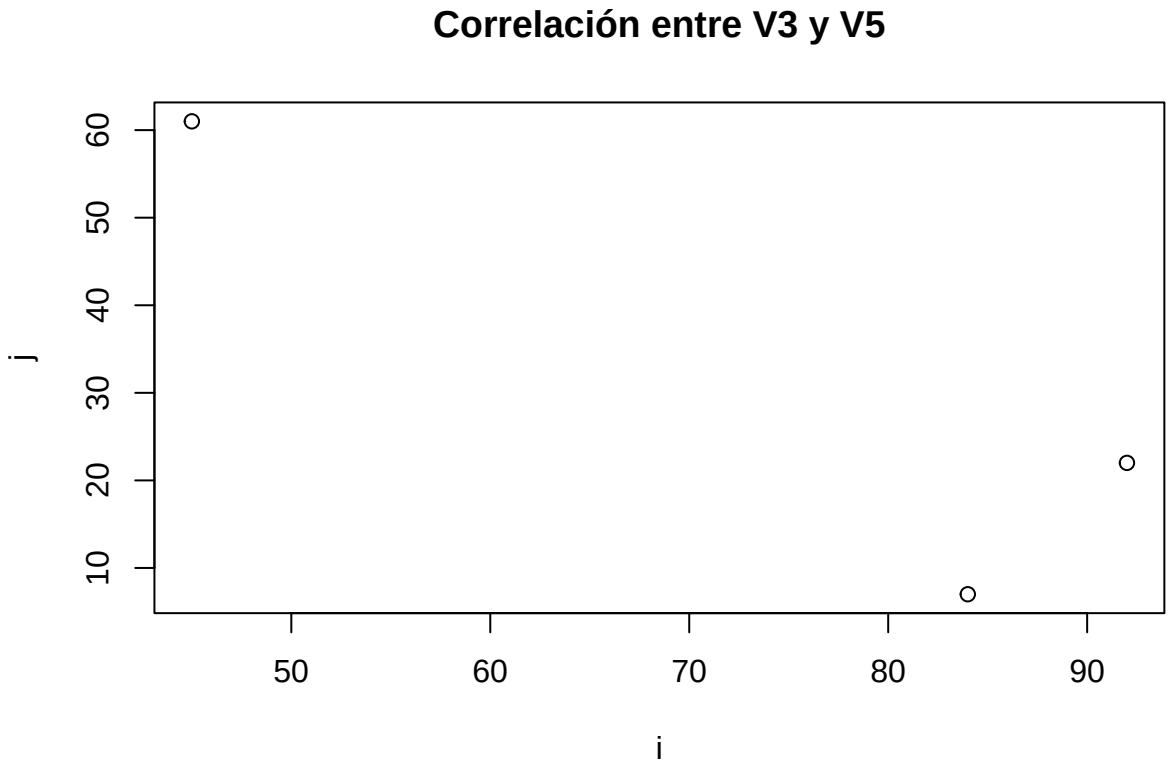

Correlación entre V2 y V8
j
10
20
30
40
50
20
40
60
80
i

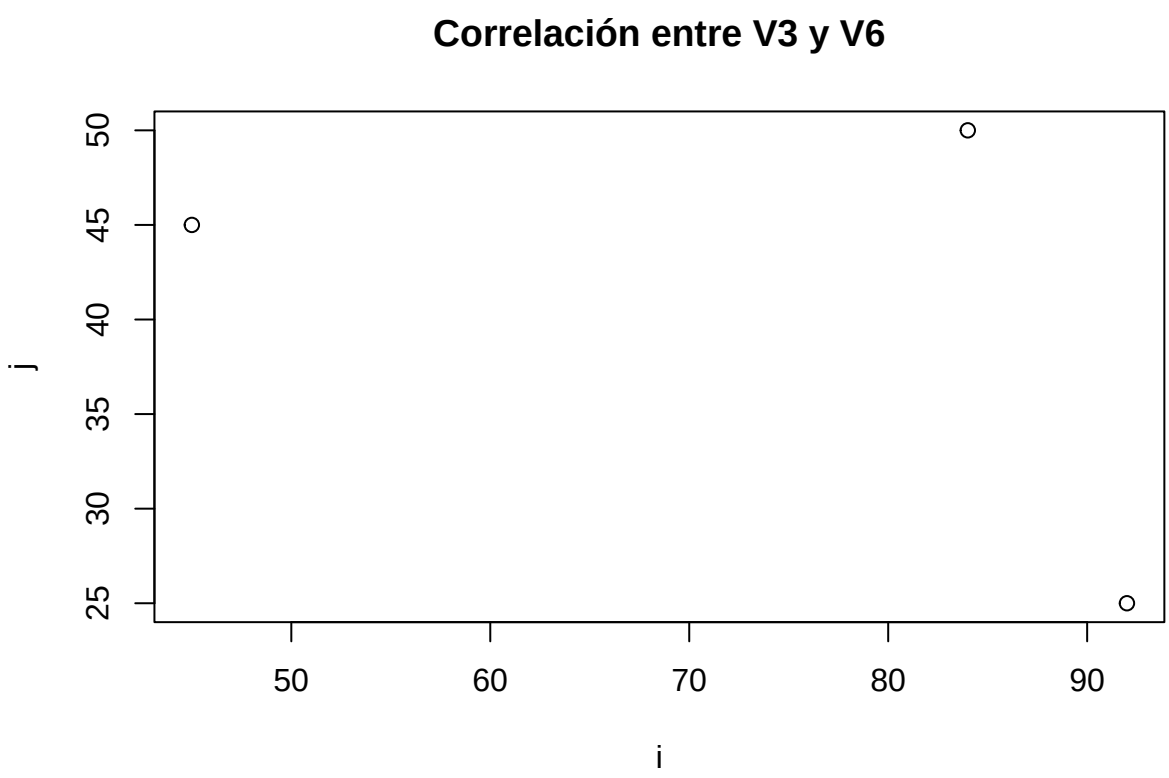

Correlación entre V2 y V9
j
30
40
50
60
70
80
90
20
40
60
80
i

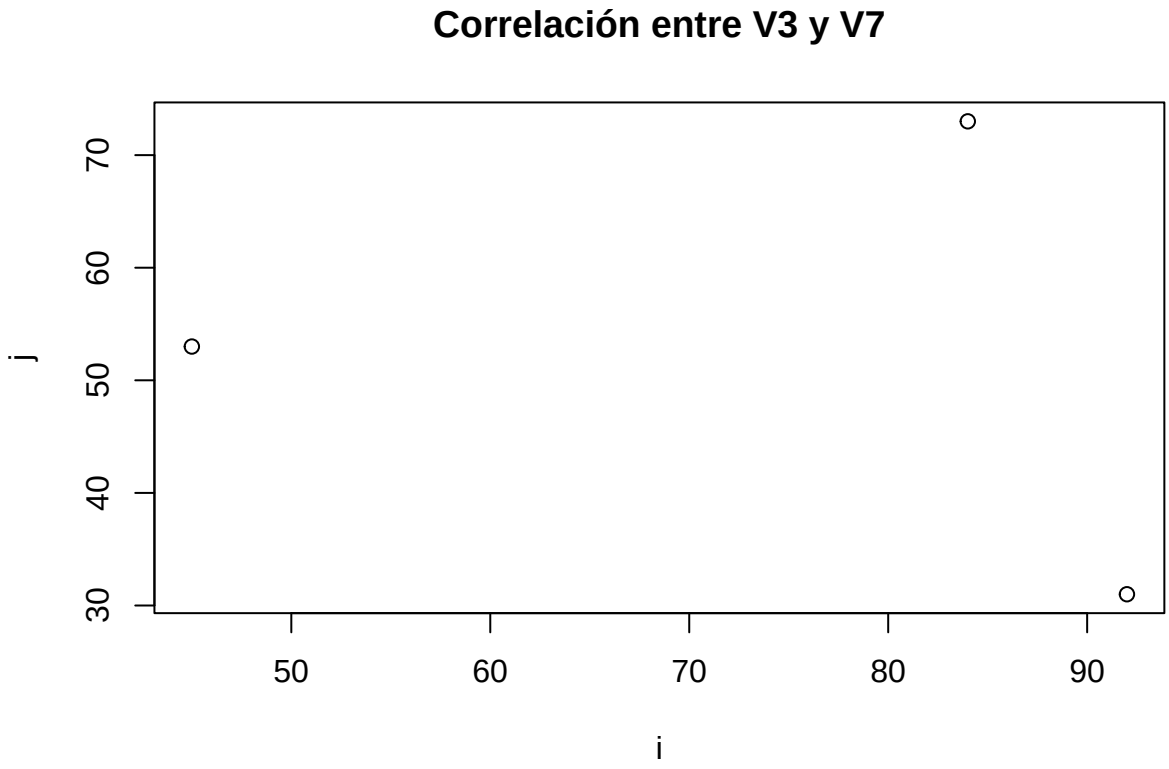

Correlación entre V2 y V10
j
40
50
60
70
20
40
60
80
i

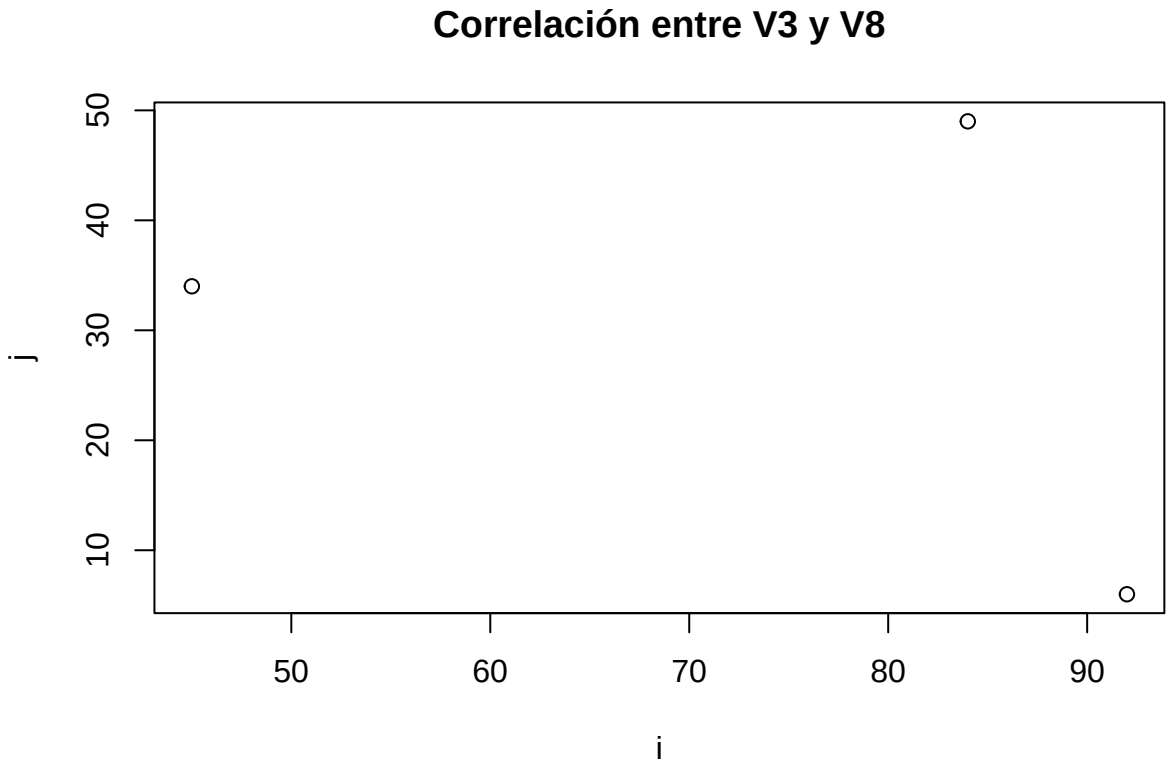

Correlación entre V2 y V11
j
20
40
60
80
i
20
40
60
80

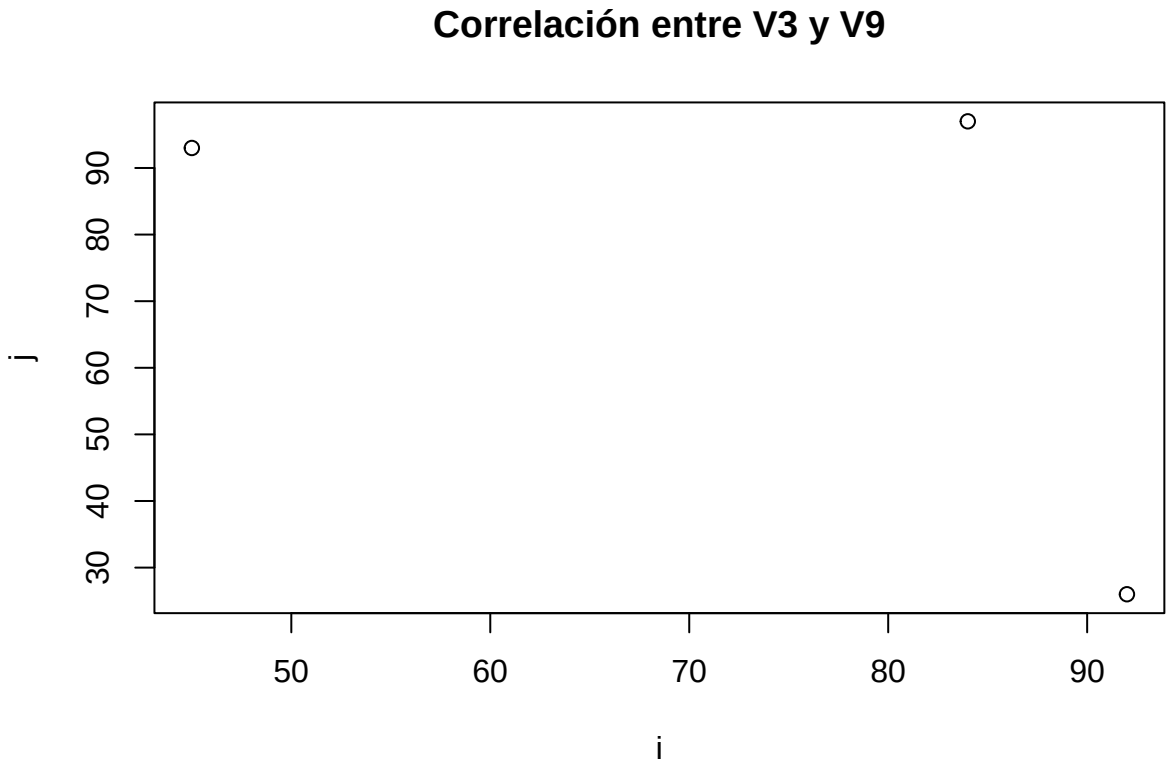

Correlación entre V3 y V4
j
20
30
40
50
60
70
80
i
50
60
70
80
90

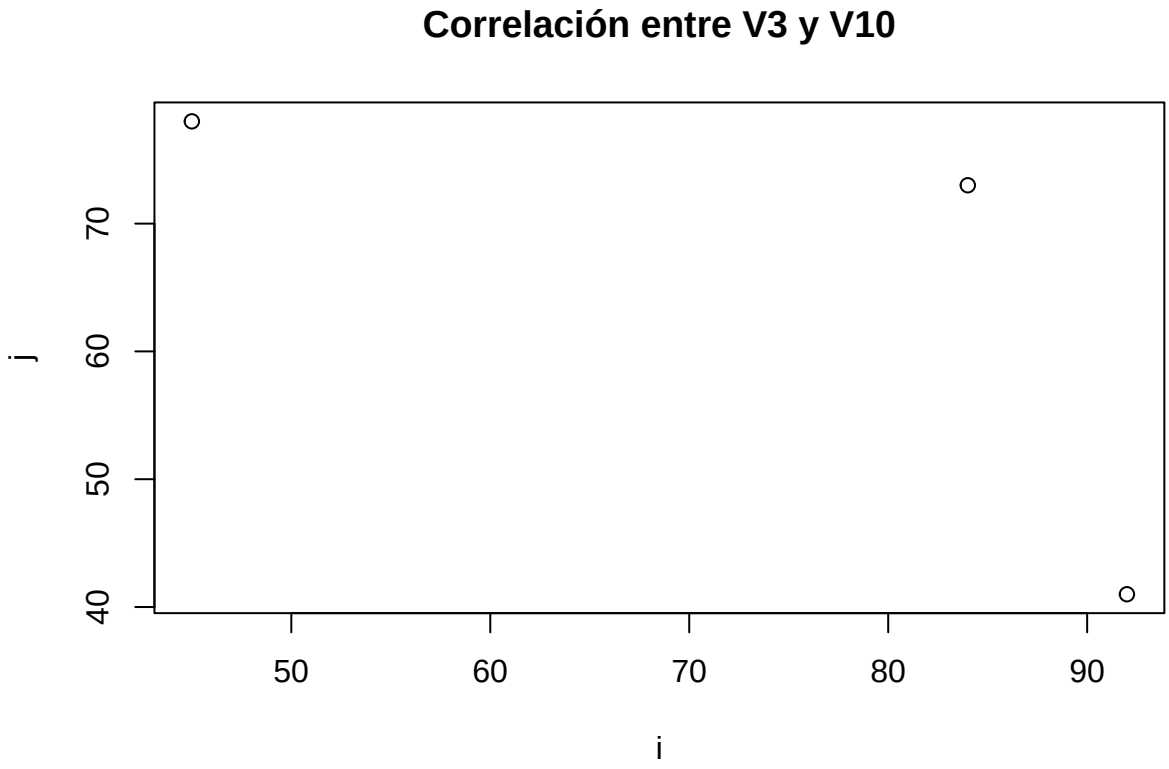

Correlación entre V3 y V5
j
10
20
30
40
50
60
50
60
70
80
90
i

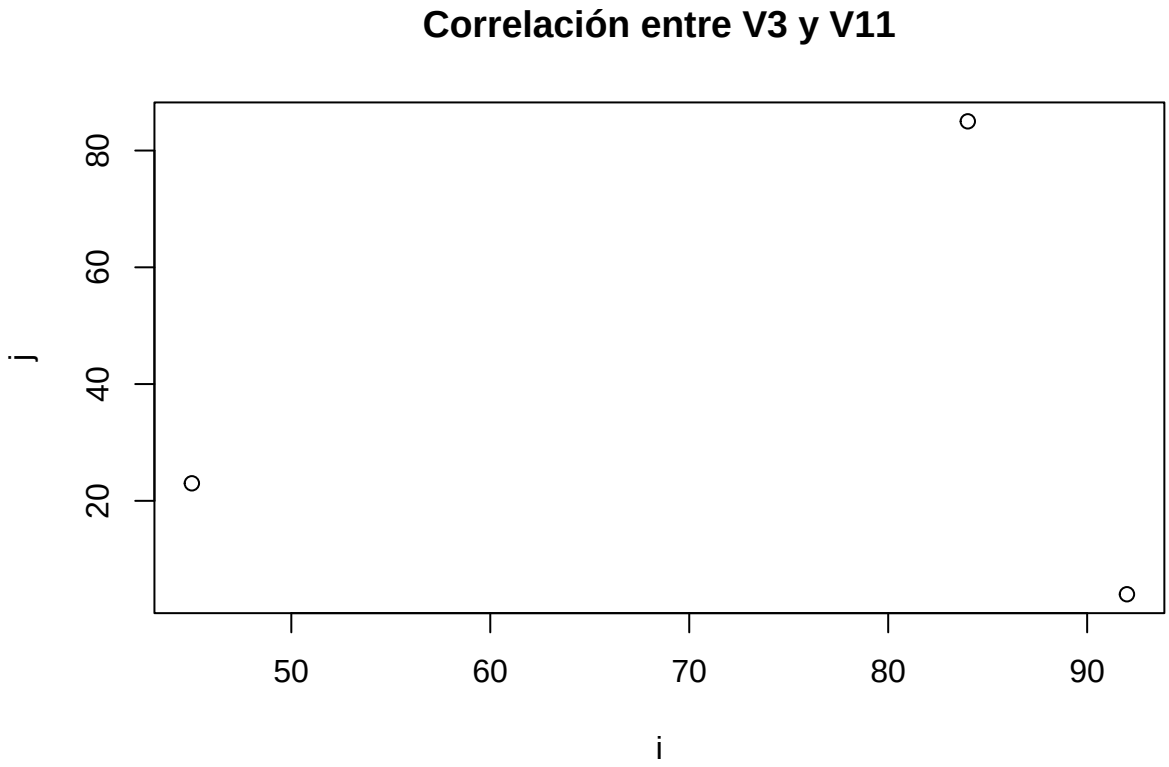

Correlación entre V3 y V6
j
25
30
35
40
45
50
50
60
70
80
90
i

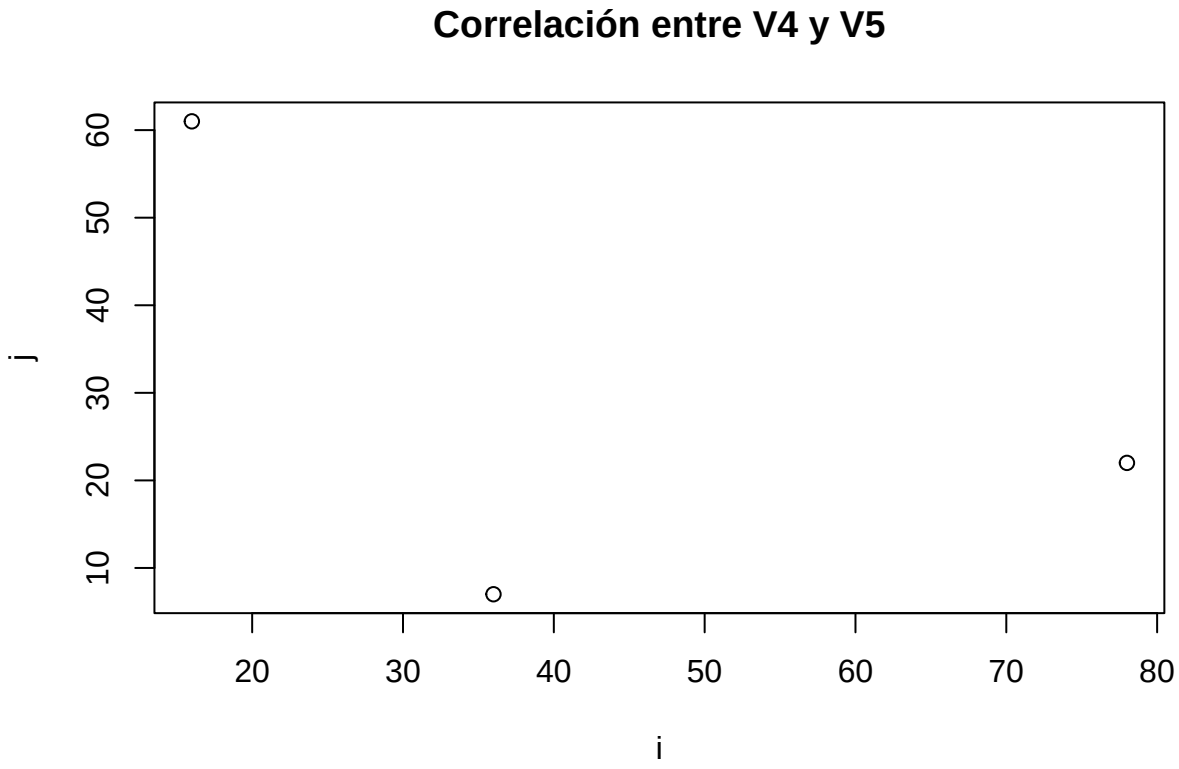

Correlación entre V3 y V7
j
30
40
50
60
70
50
60
70
80
90
i

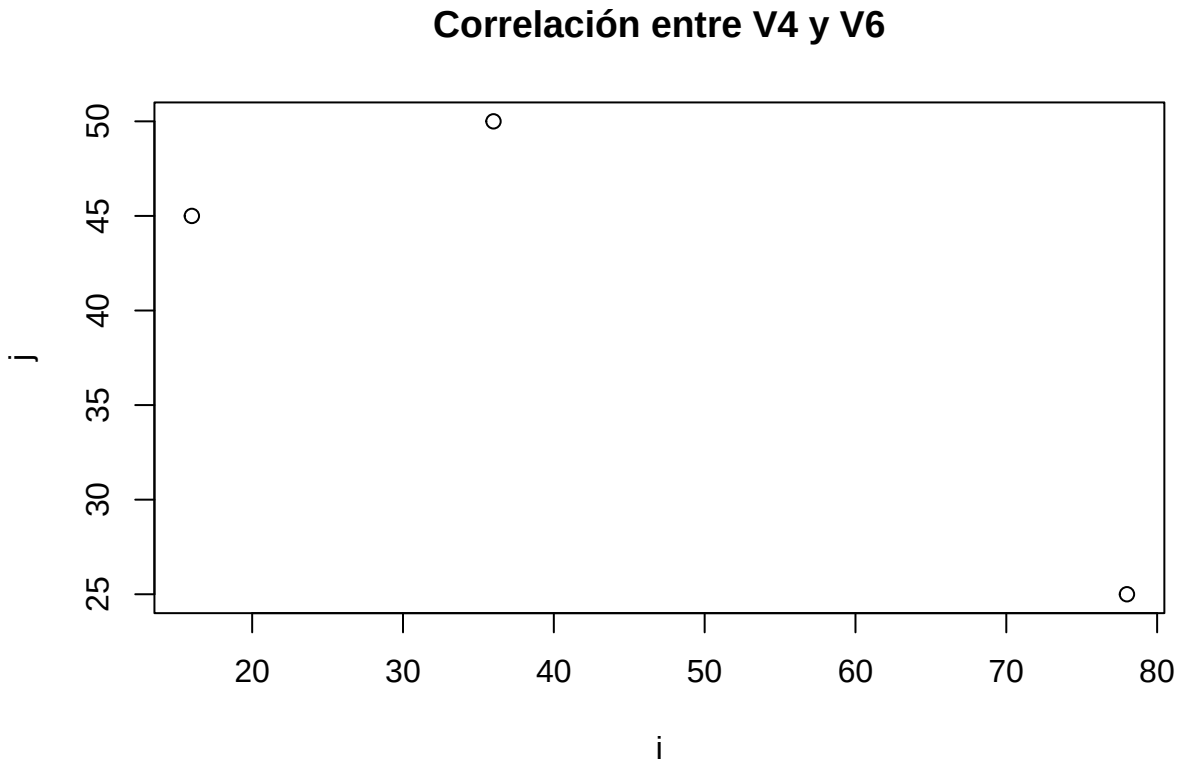

Correlación entre V3 y V8
j
10
20
30
40
50
50
60
70
80
90
i

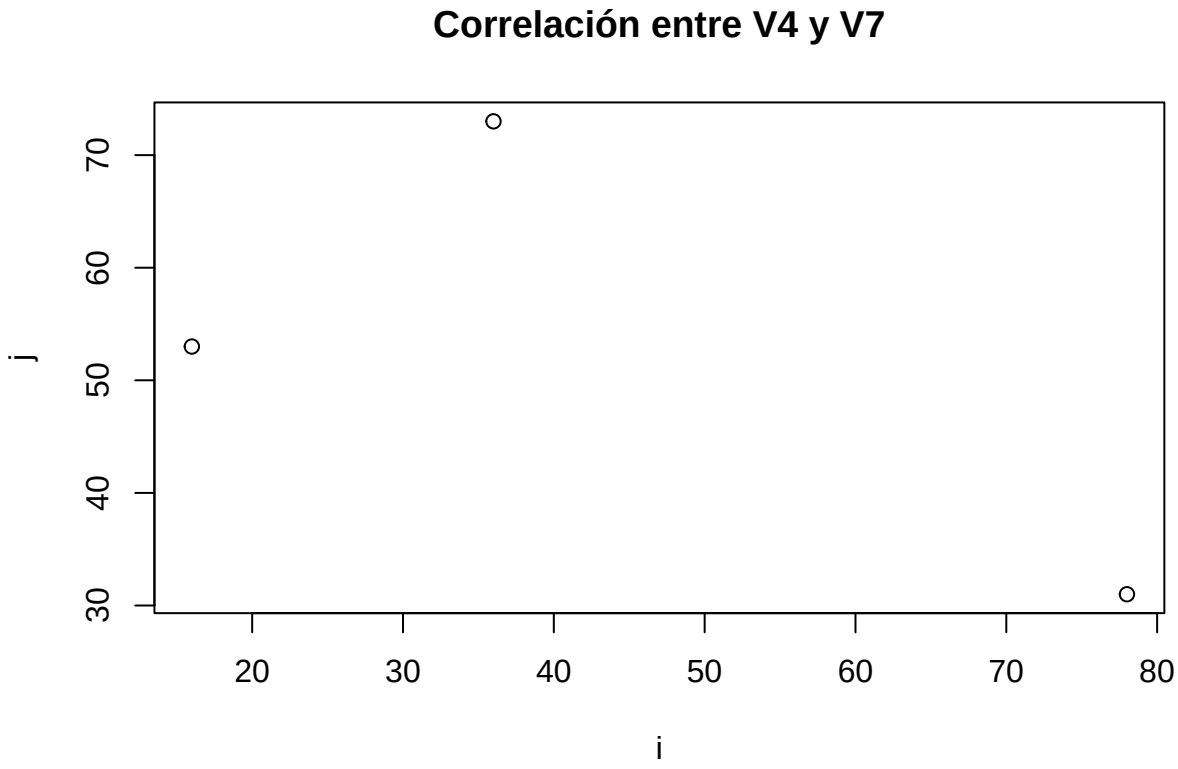

Correlación entre V3 y V9
j
30
40
50
60
70
80
90
50
60
70
80
90
i

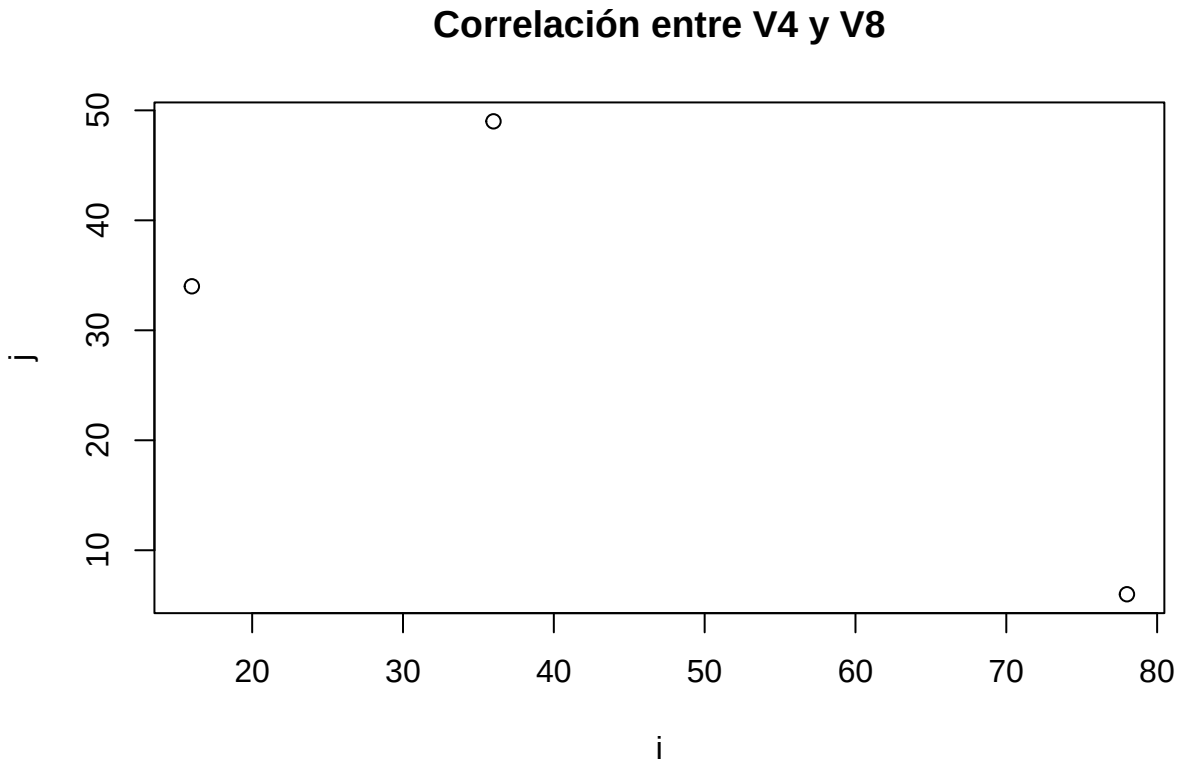

Correlación entre V3 y V10
j
40
50
60
70
50
60
70
80
90
i

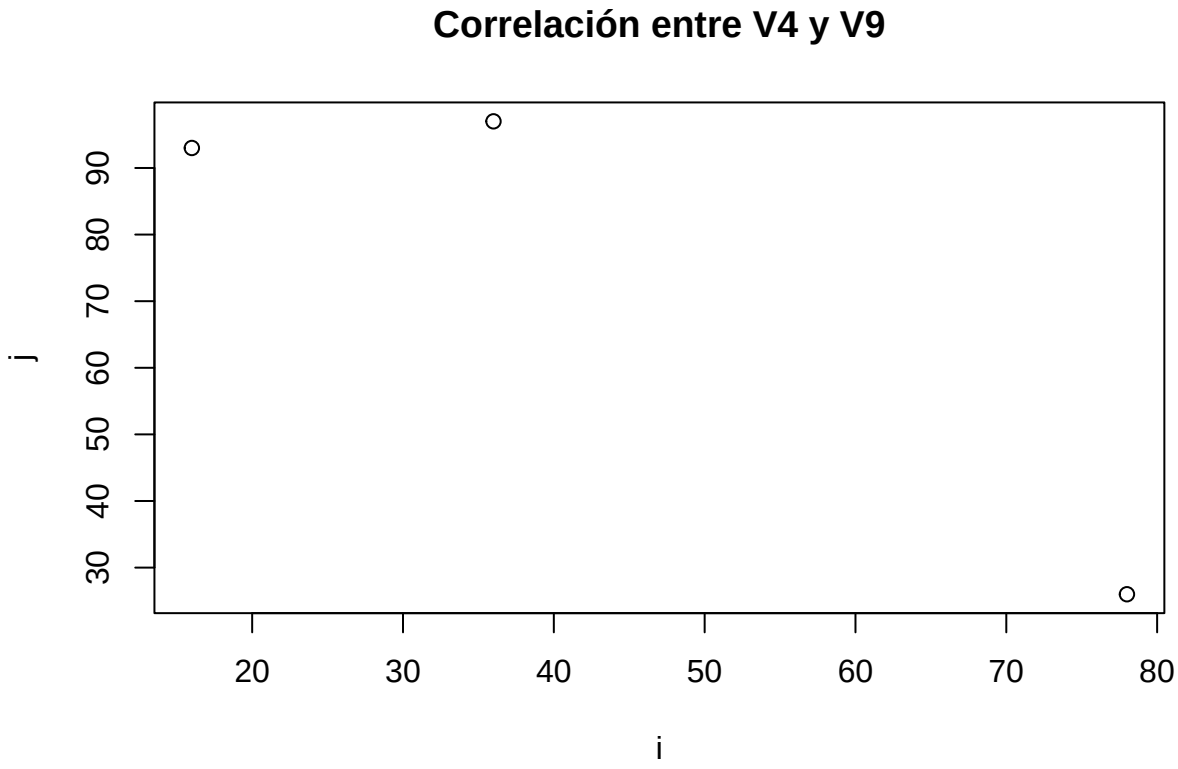

Correlación entre V3 y V11
j
20
40
60
80
50
60
70
80
90
i

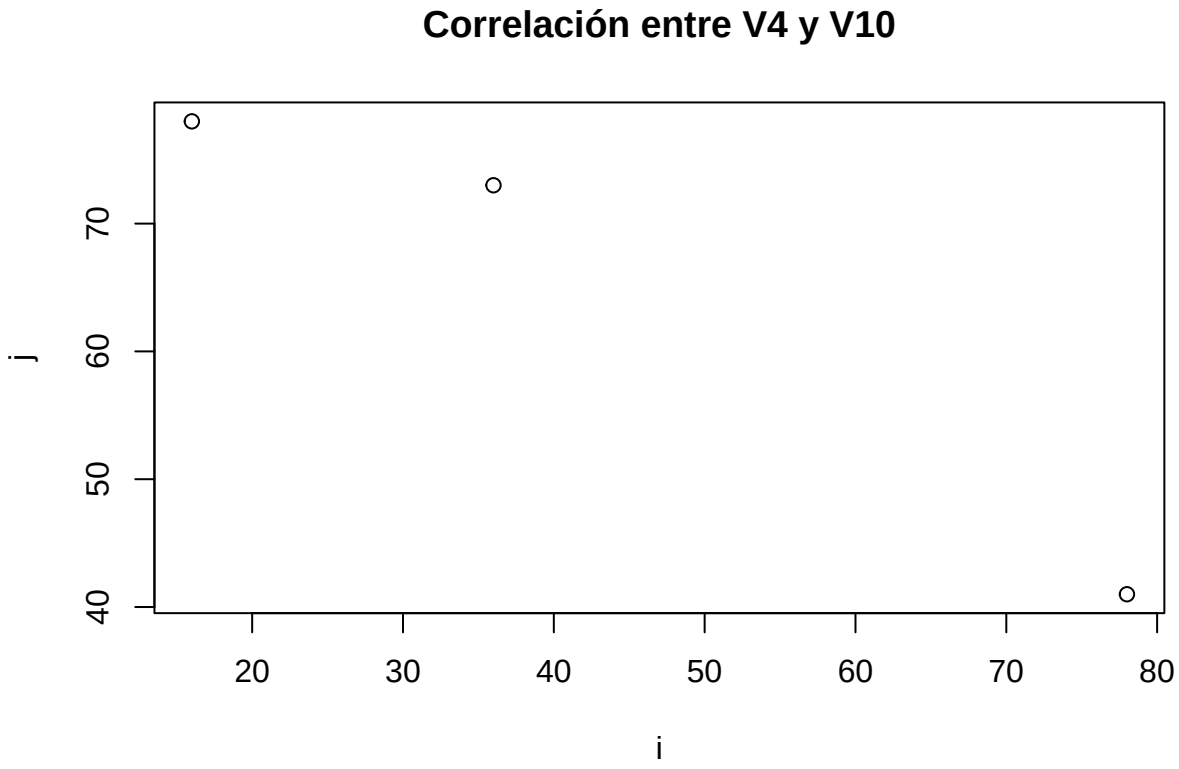

Correlación entre V4 y V5
j
10
20
30
40
50
60
20
30
40
50
60
70
80
i

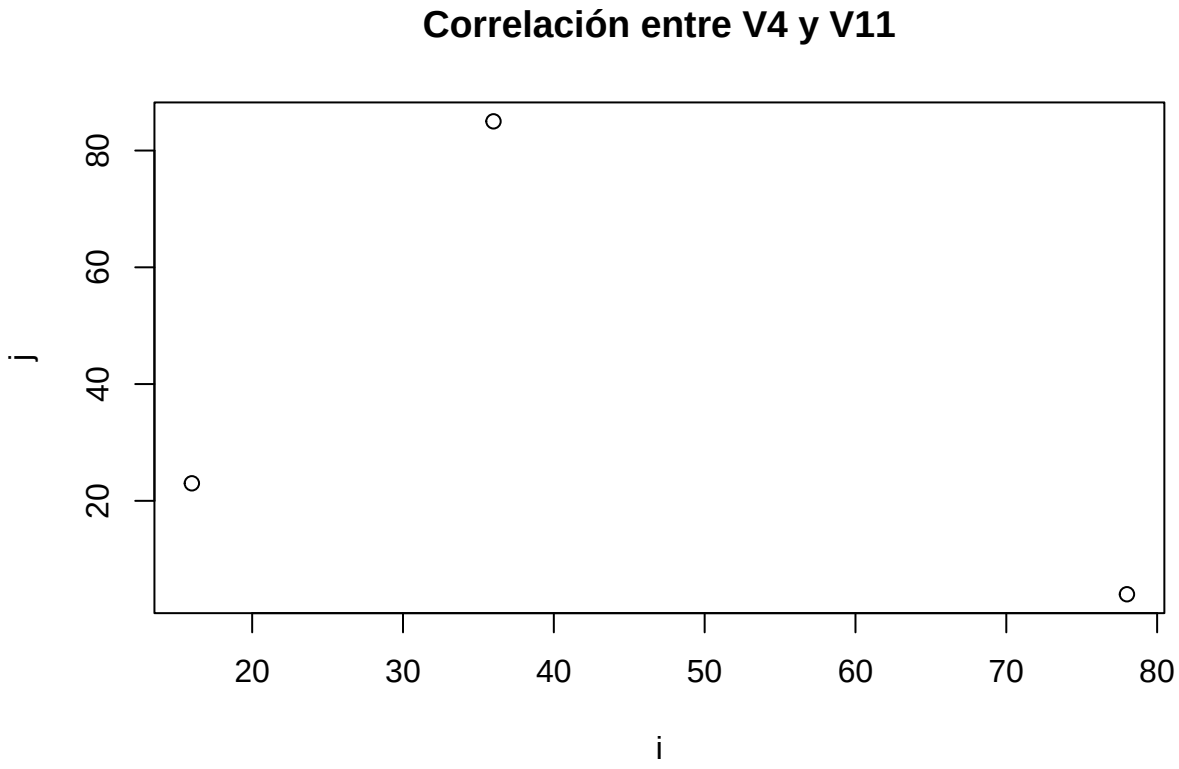

Correlación entre V4 y V6
j
25
30
35
40
45
50
20
30
40
50
60
70
80
i

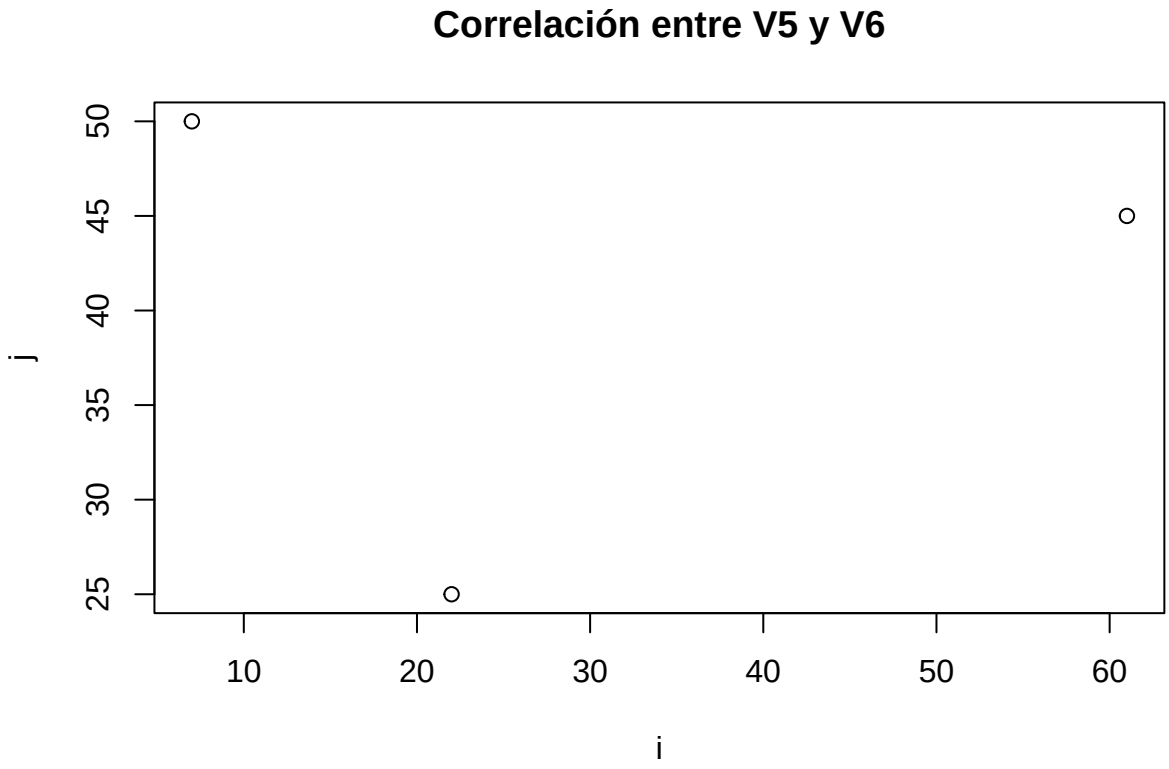

Correlación entre V4 y V7
j
30
40
50
60
70
20
30
40
50
60
70
80
i

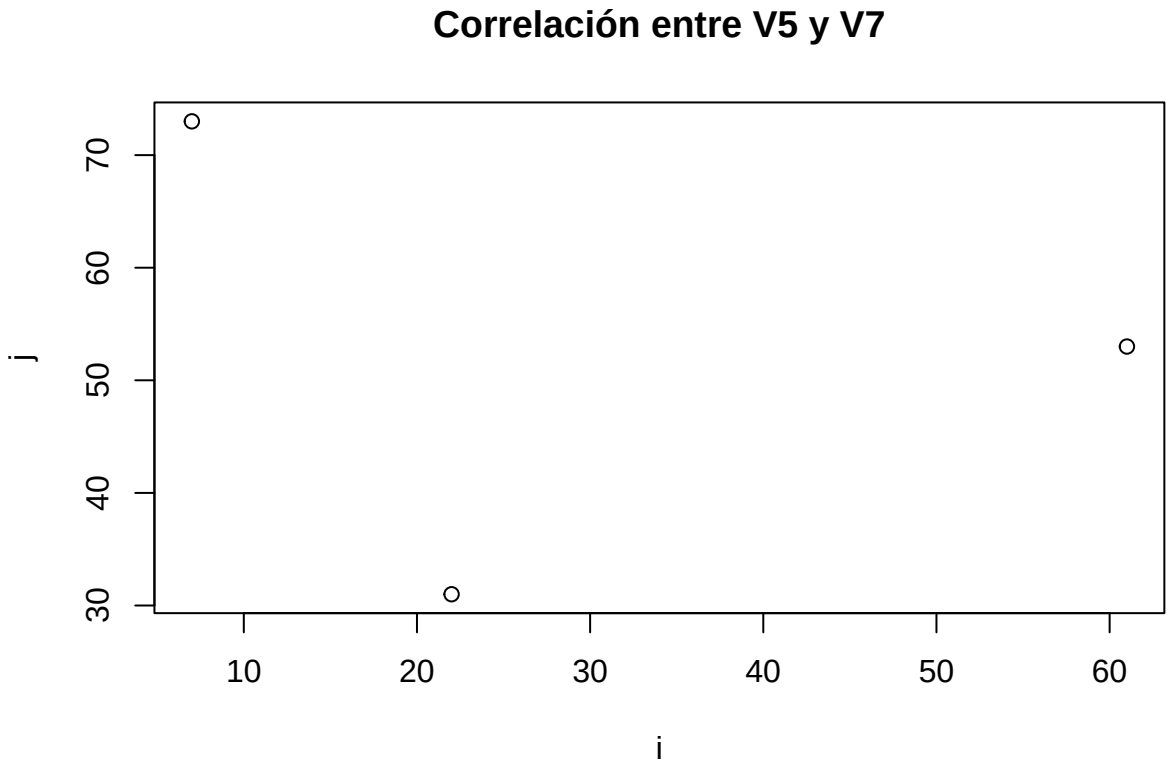

Correlación entre V4 y V8
j
i
10
20
30
40
50
20
30
40
50
60
70
80

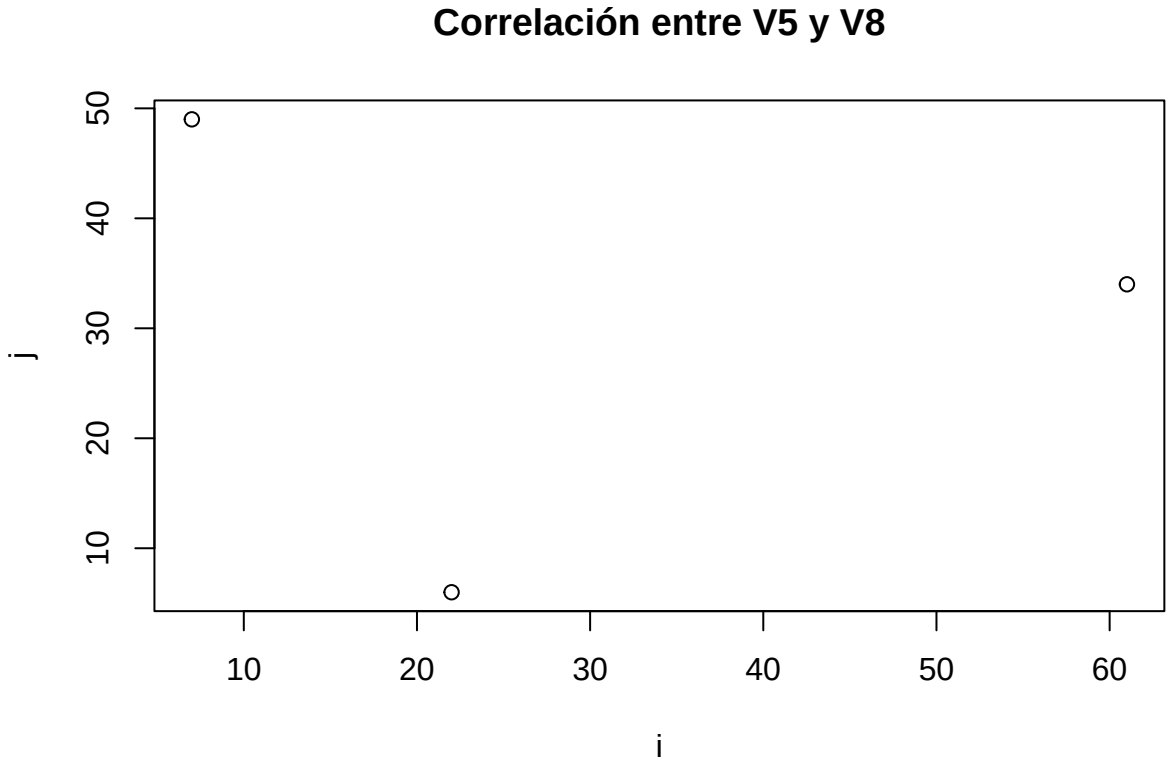

Correlación entre V4 y V9
j
30
40
50
60
70
80
90
20
30
40
50
60
70
80
i

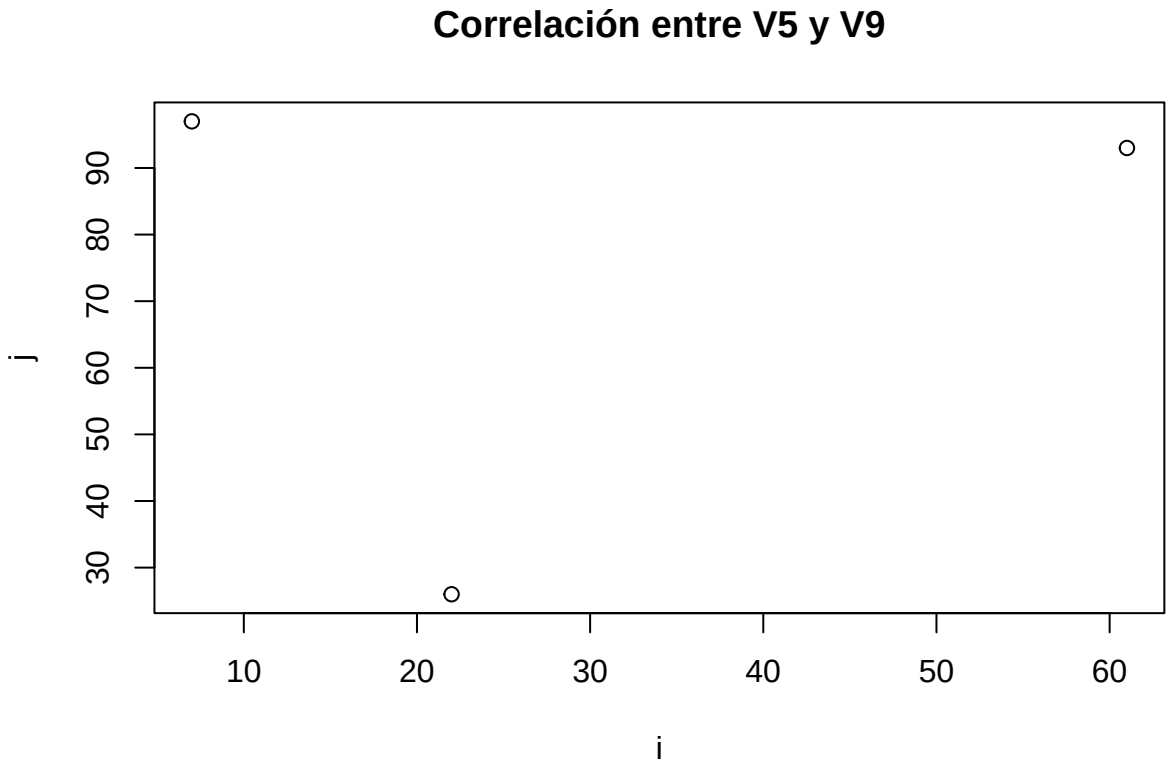

Correlación entre V4 y V10
j
40
50
60
70
20
30
40
50
60
70
80
i

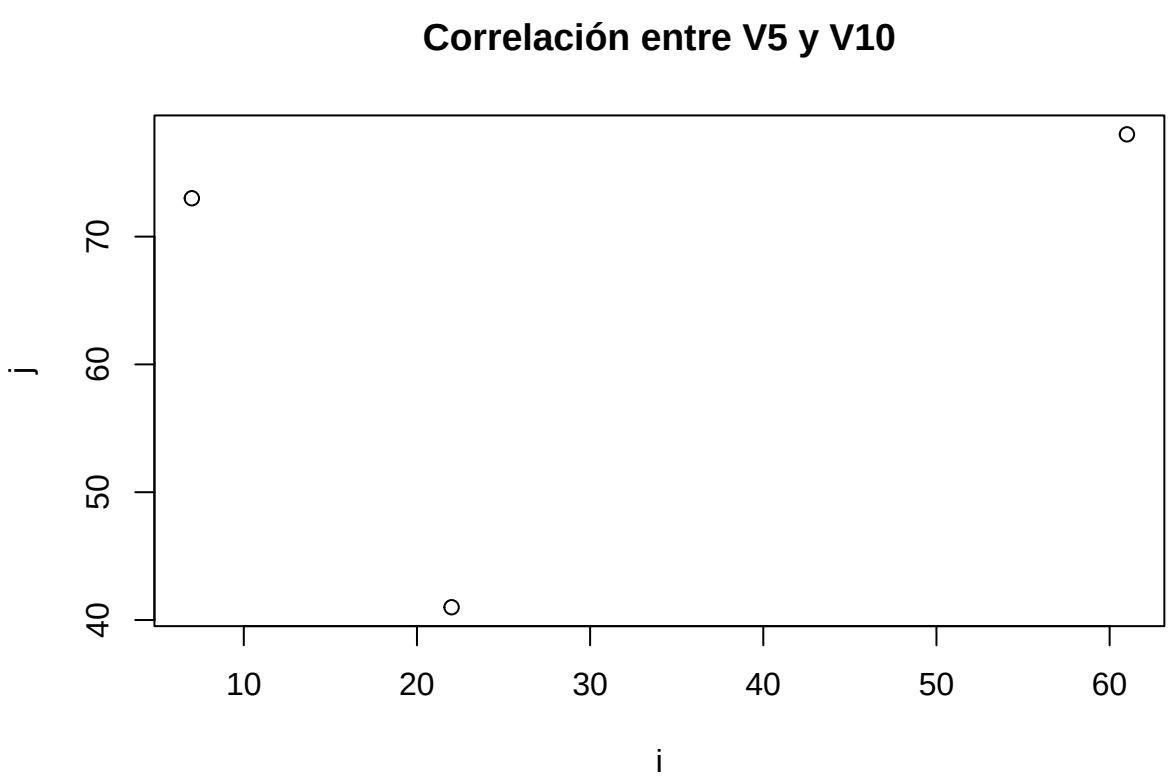

Correlación entre V4 y V11
j
20
40
60
80
20
30
40
50
60
70
80
i

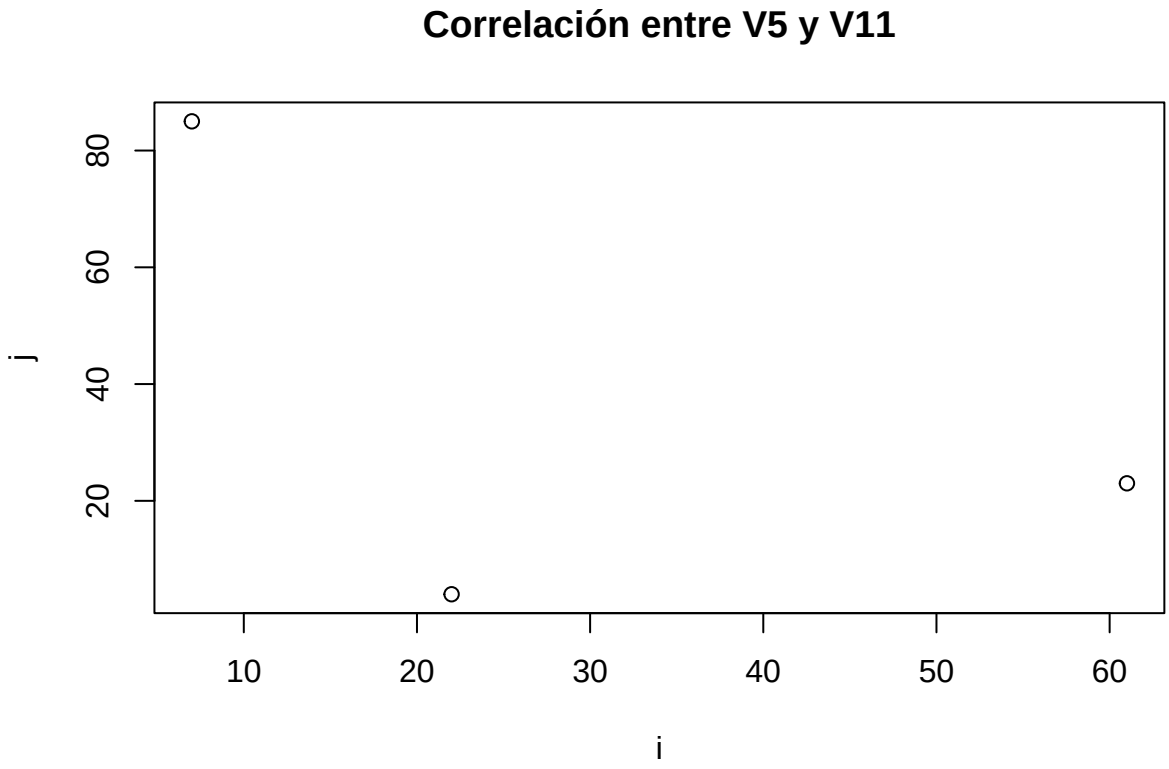

Correlación entre V5 y V6
j
25
30
35
40
45
50
10
20
30
40
50
60
i

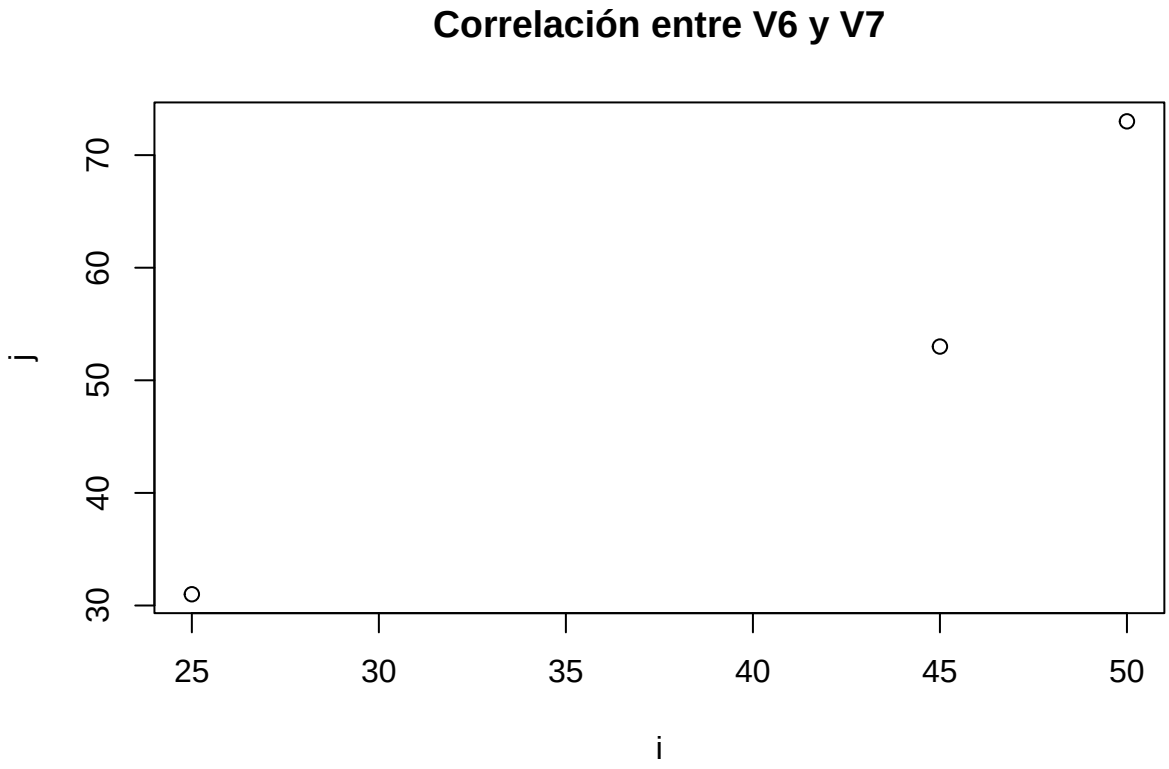

Correlación entre V5 y V7
j
30
40
50
60
70
10
20
30
40
50
60
i

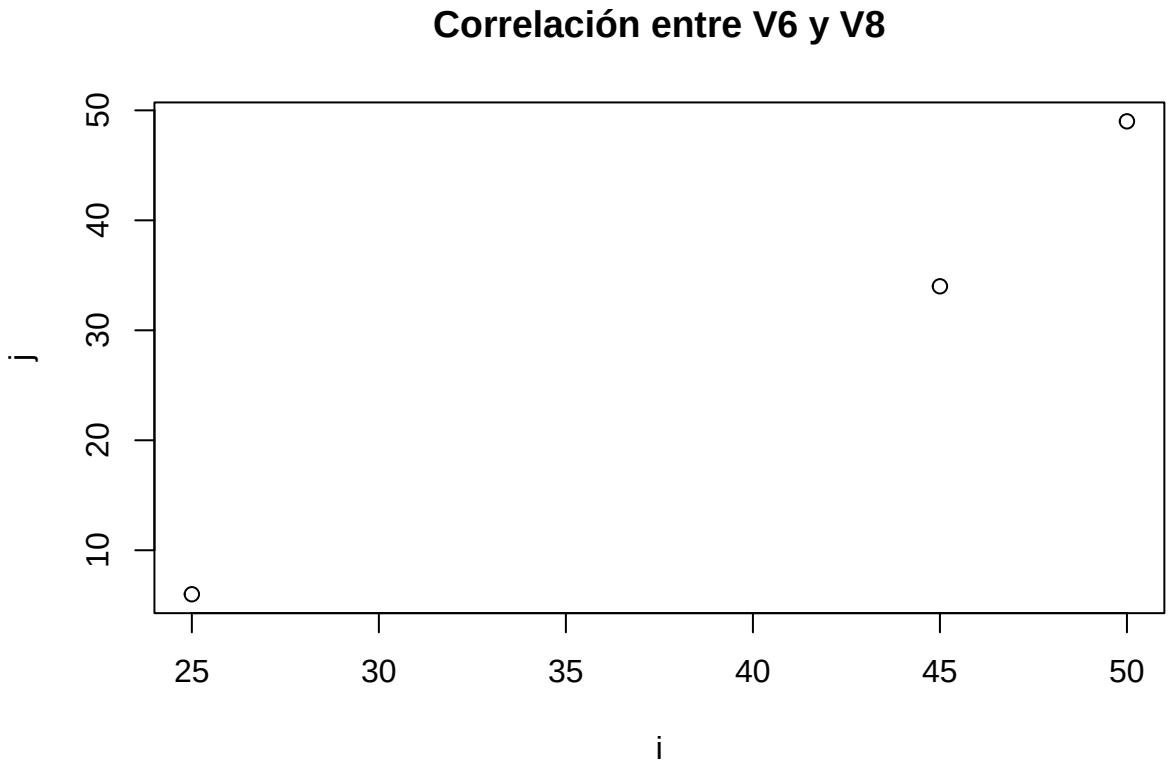

Correlación entre V5 y V8
j
i
10
20
30
40
50
60

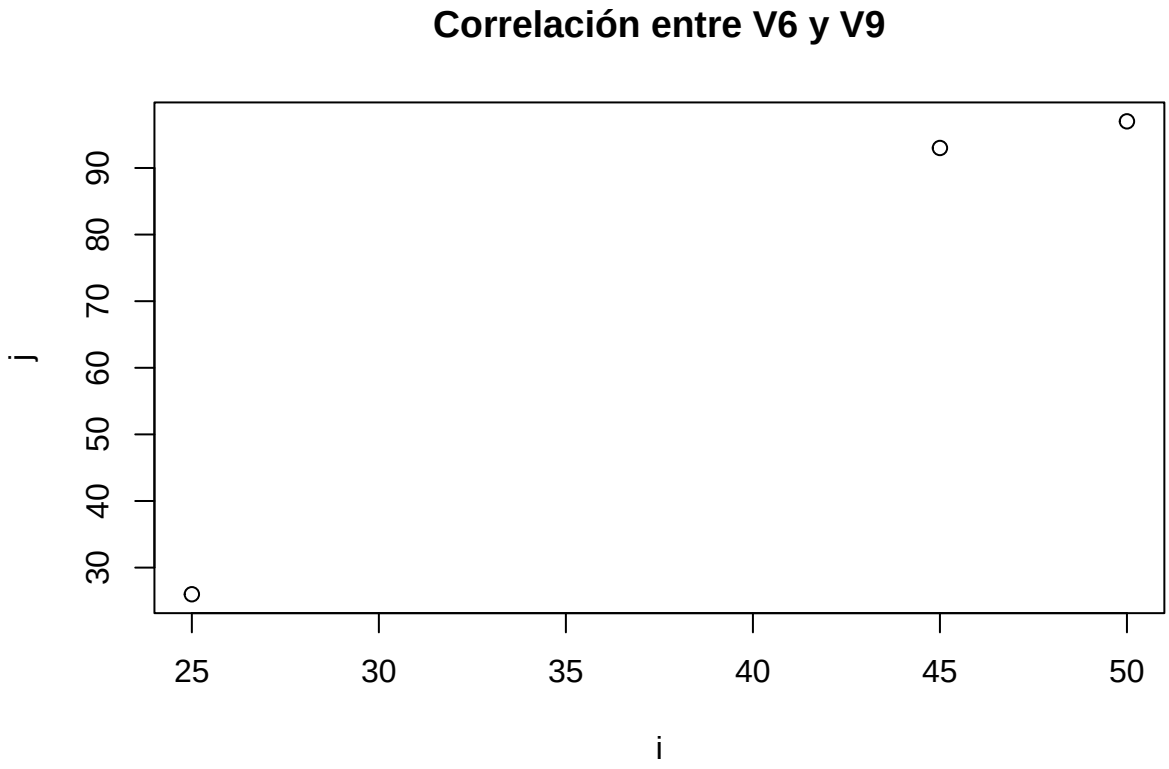

Correlación entre V5 y V9
j
30
40
50
60
70
80
90
10
20
30
40
50
60
i

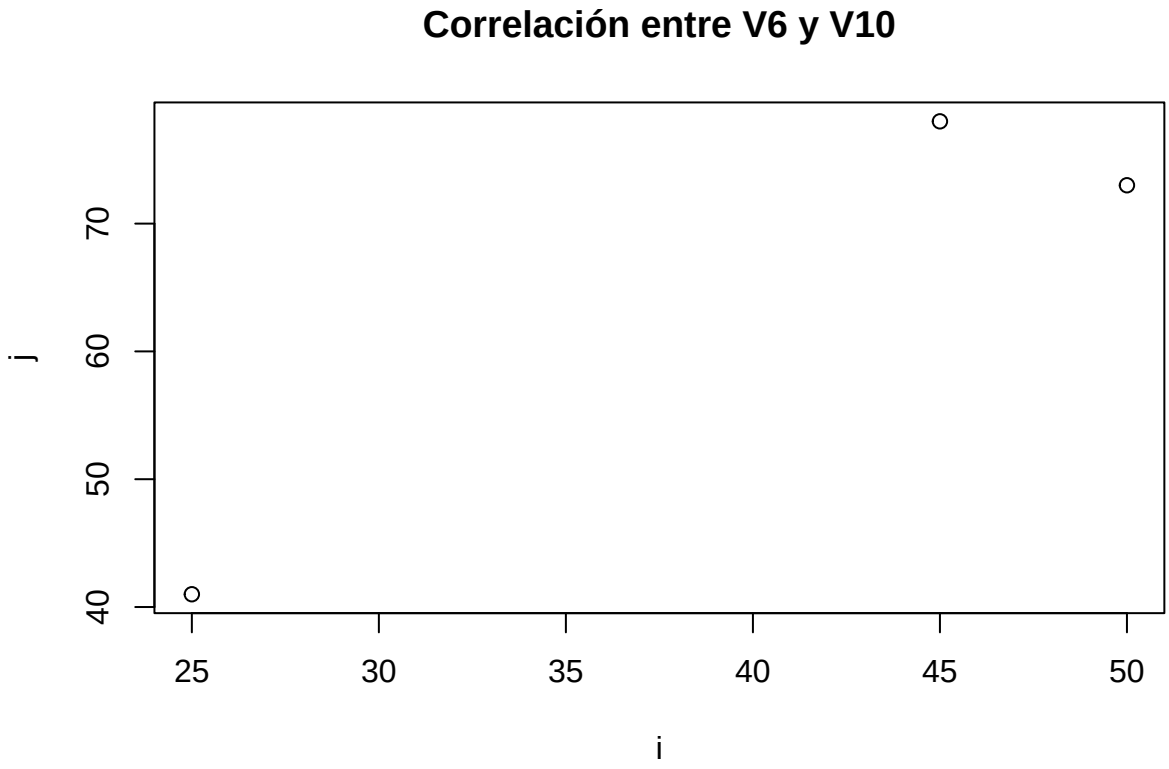

Correlación entre V5 y V10
j
40
50
60
70
10
20
30
40
50
60
i

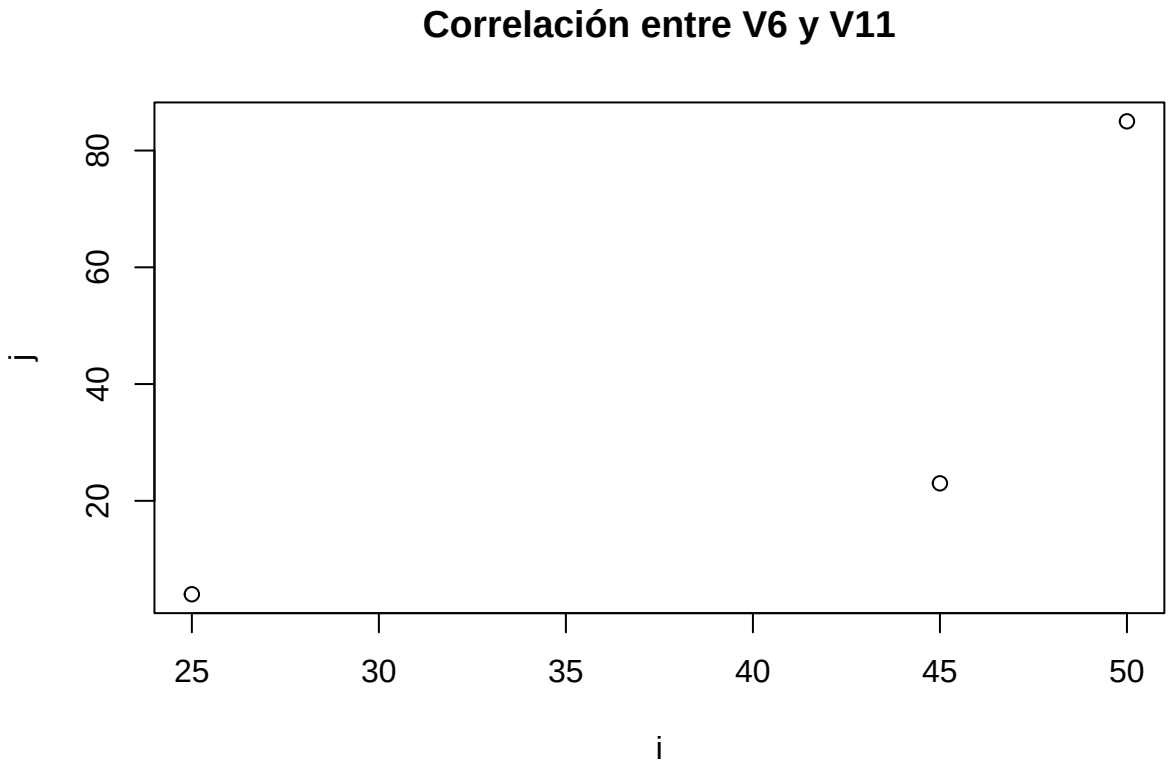

Correlación entre V5 y V11
j
20
40
60
80
10
20
30
40
50
60
i

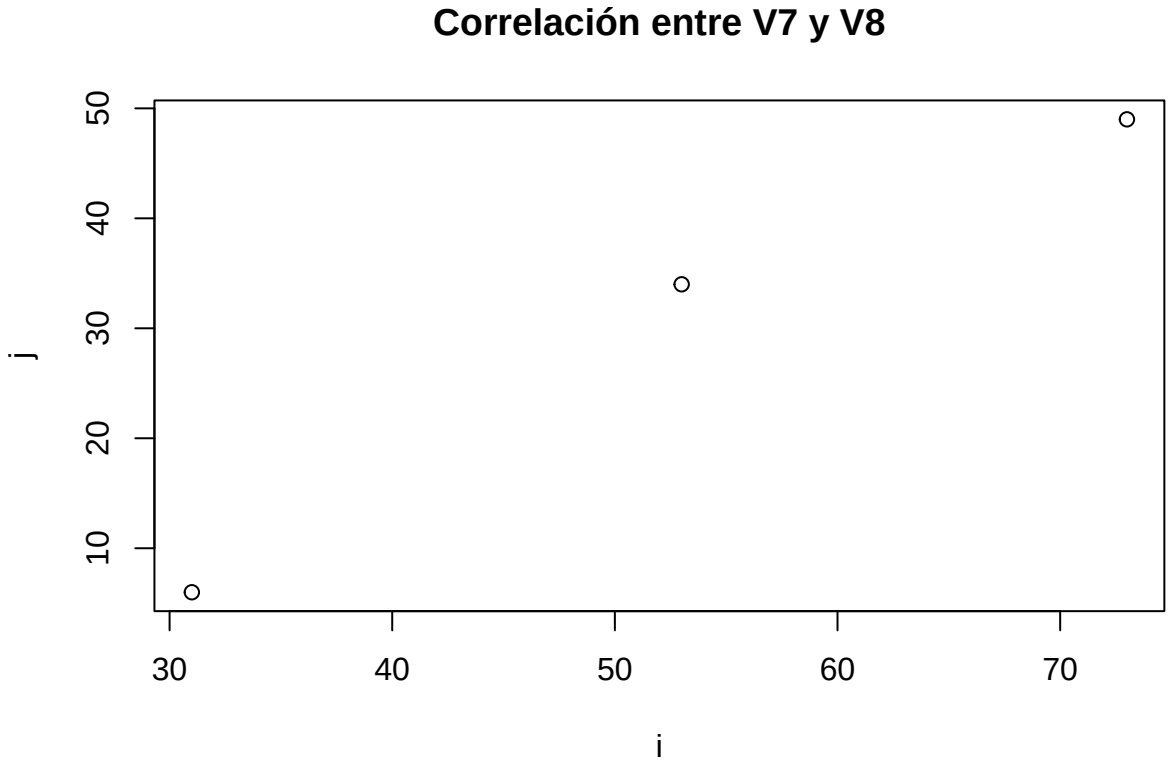

Correlación entre V6 y V7
j
30
40
50
60
70
25
30
35
40
45
50
i

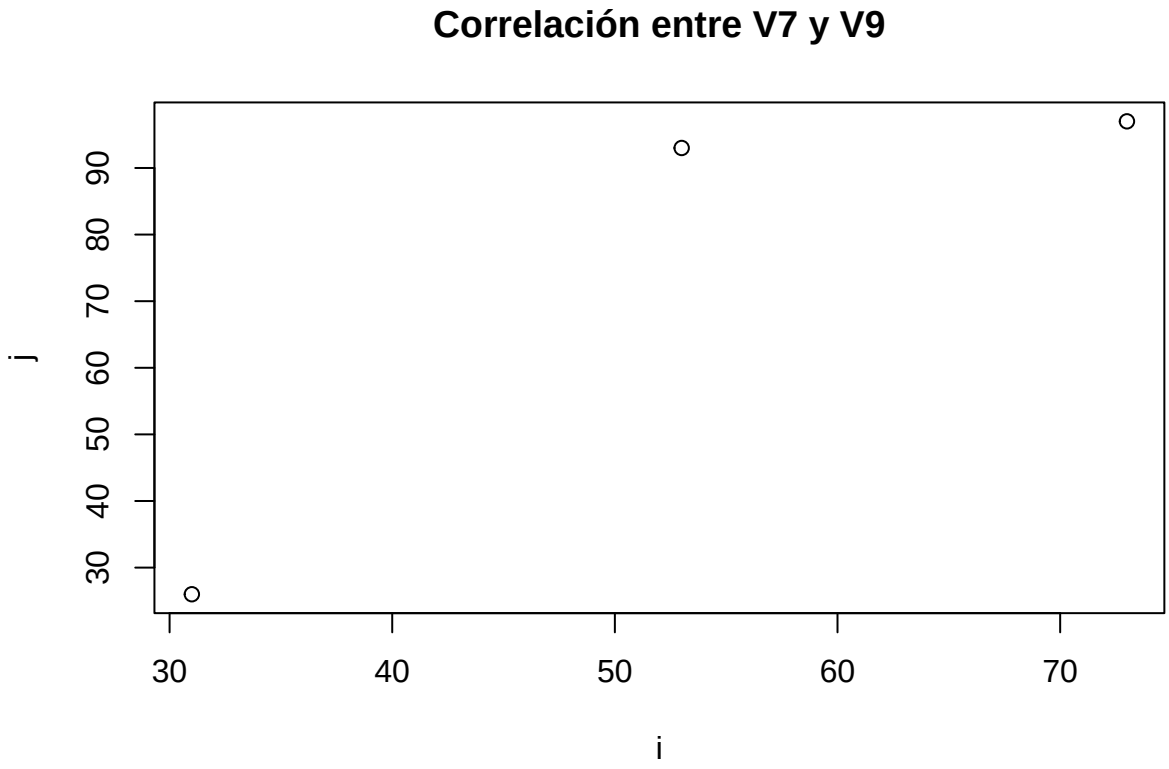

Correlación entre V6 y V8
j
10
20
30
40
50
25
30
35
40
45
50
i

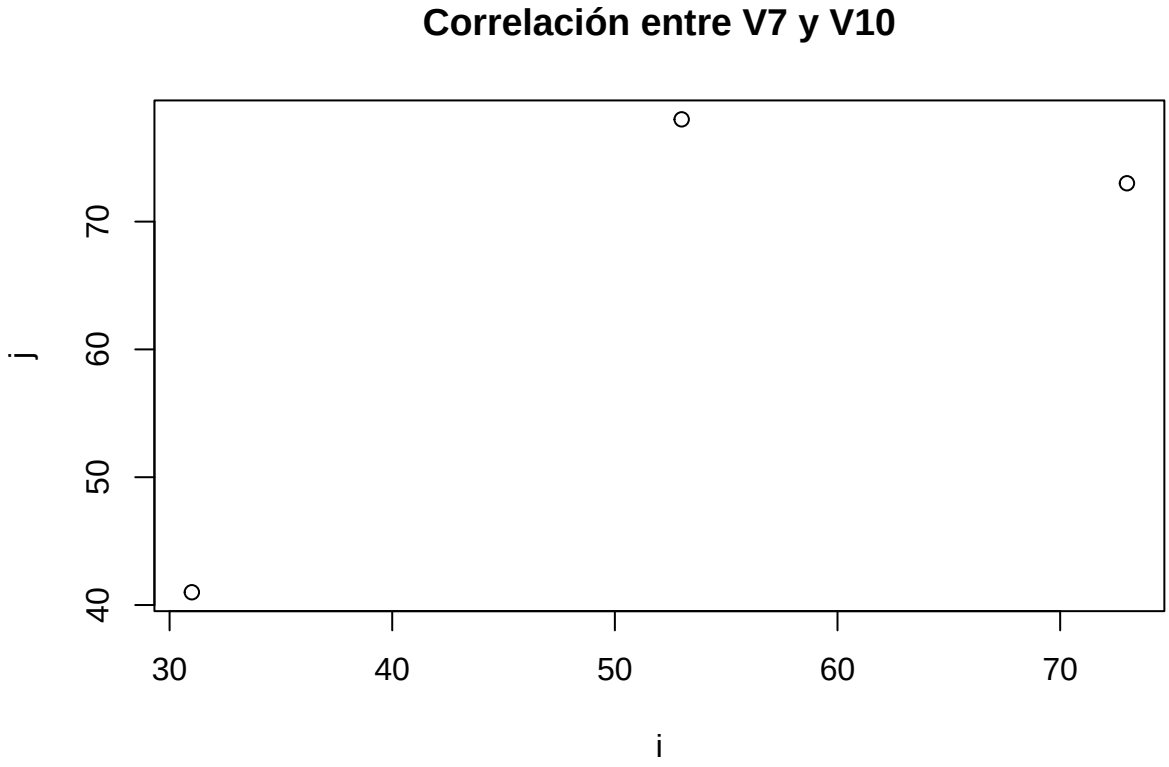

Correlación entre V6 y V9
j
30
40
50
60
70
80
90
25
30
35
40
45
50
i

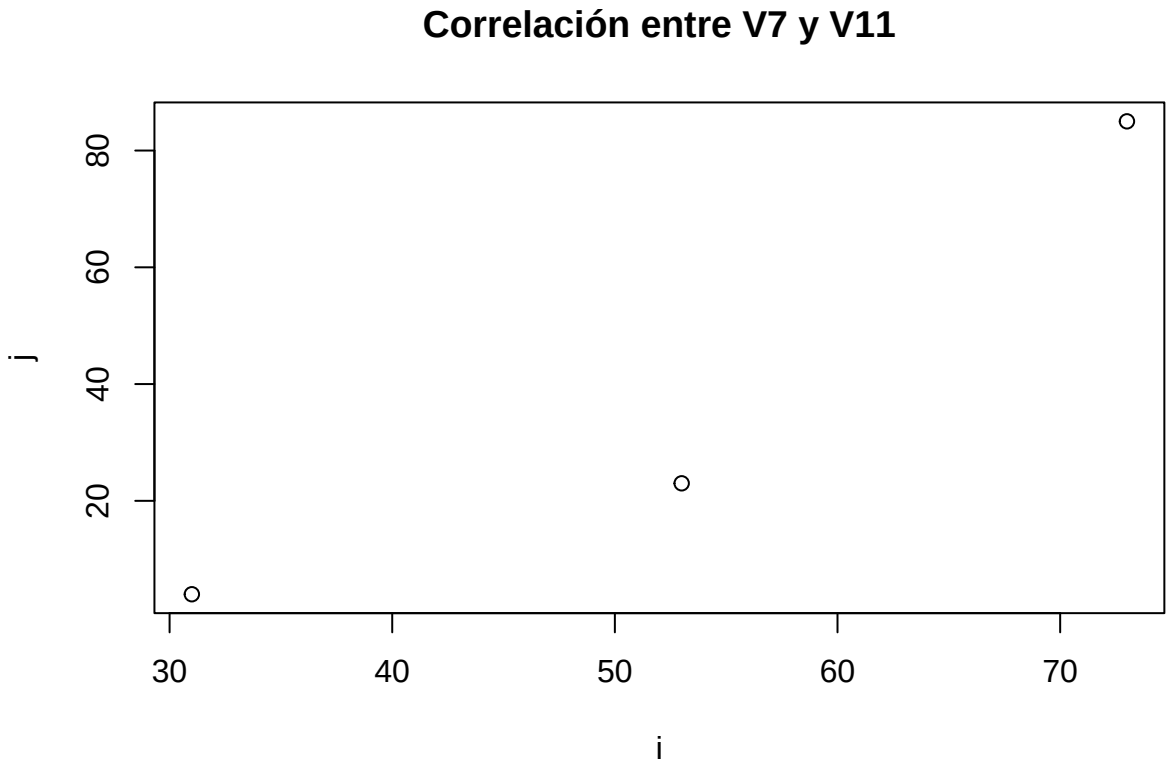

Correlación entre V6 y V10
j
40
50
60
70
25
30
35
40
45
50
i

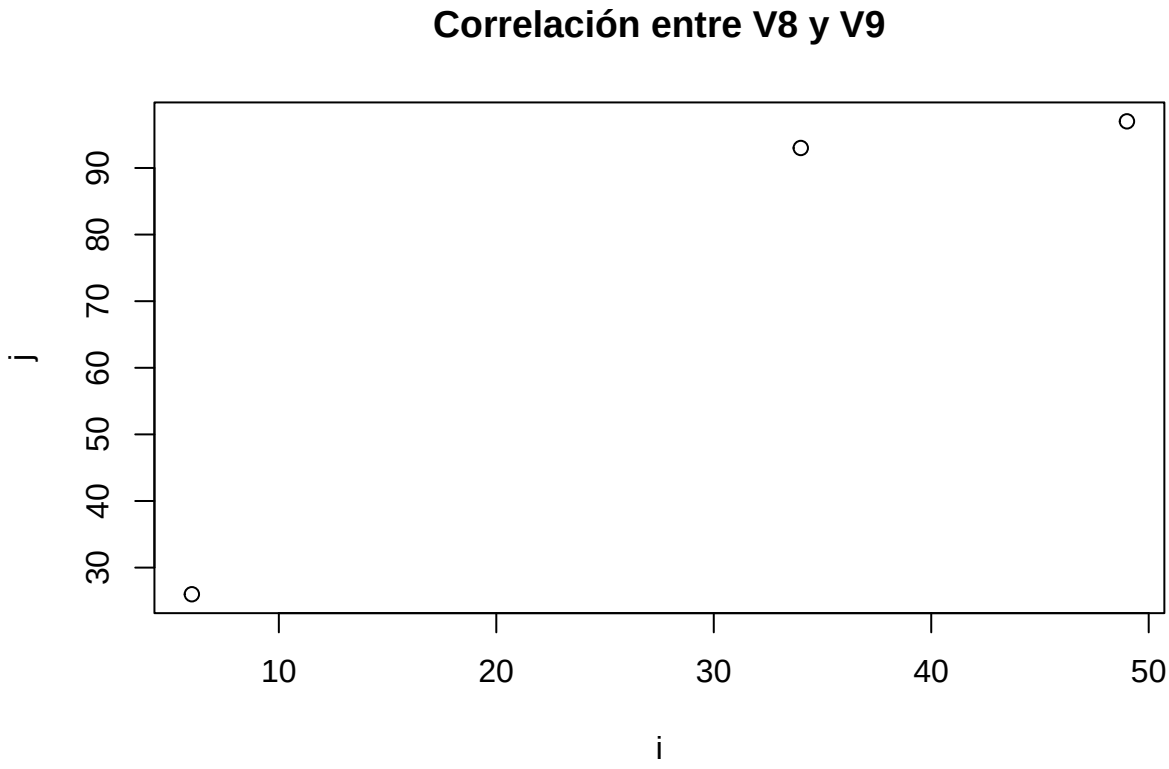

Correlación entre V6 y V11
j
20
40
60
80
25
30
35
40
45
50
i

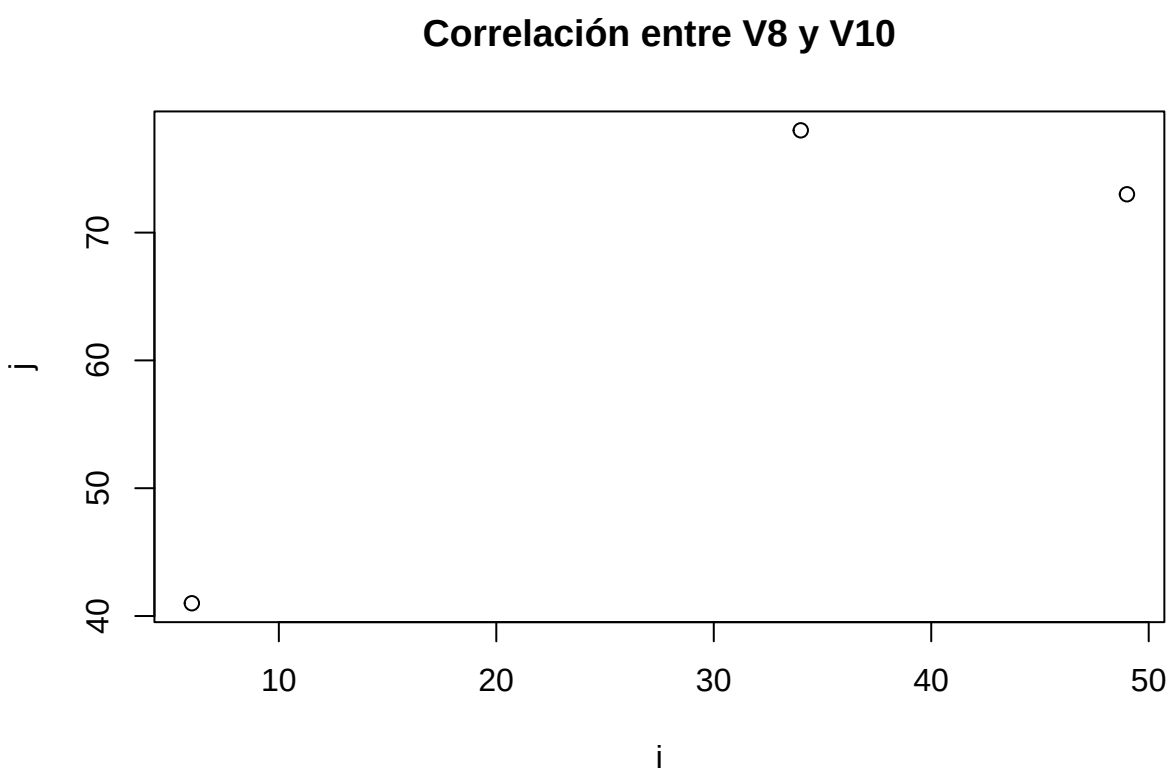

Correlación entre V7 y V8
j
10
20
30
40
50
30
40
50
60
70
i

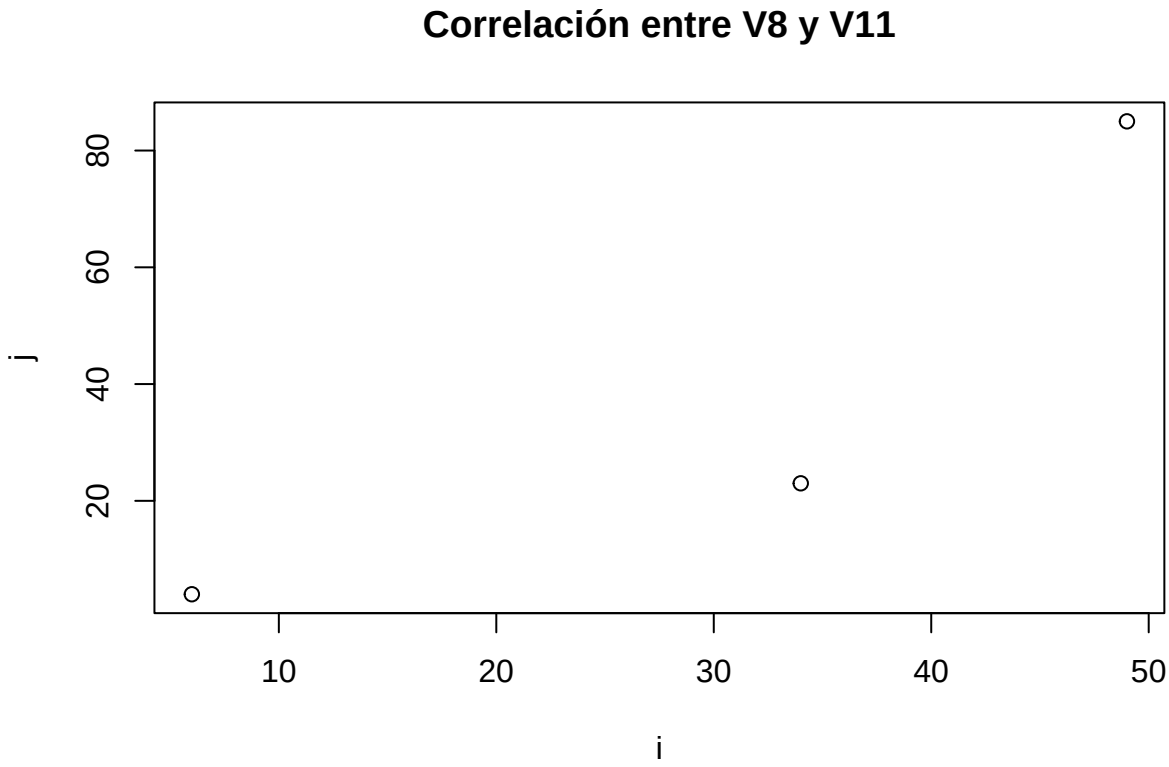

Correlación entre V7 y V9
j
30
40
50
60
70
80
90
30
40
50
60
70
i

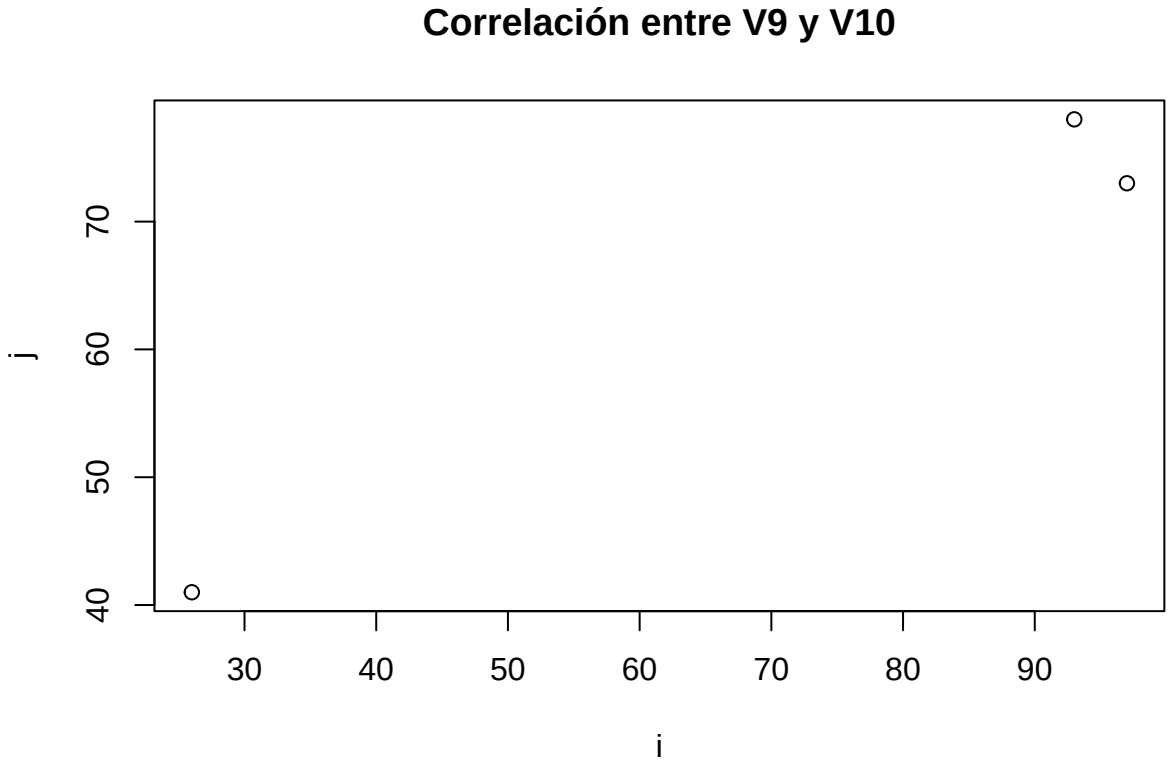

Correlación entre V7 y V10
j
40
50
60
70
30
40
50
60
70
i

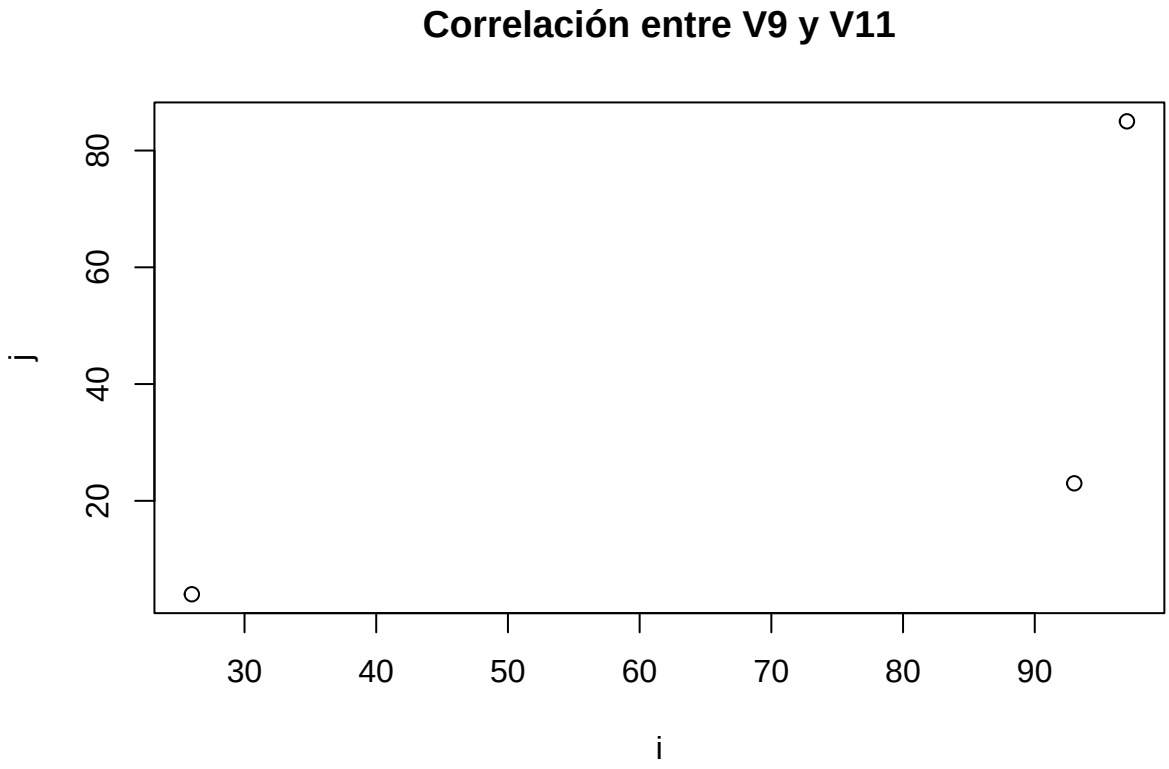

Correlación entre V7 y V11
j
20
40
60
80
30
40
50
60
70
i

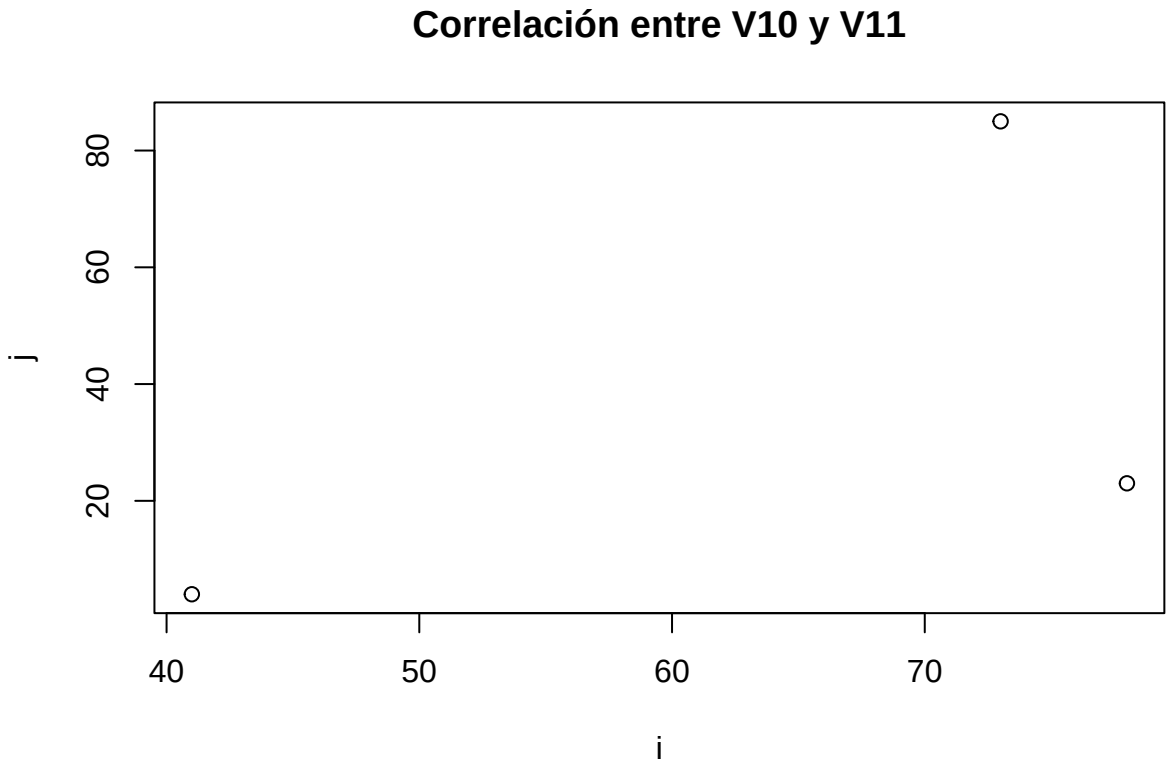

Correlación entre V8 y V9
j
30
40
50
60
70
80
90
10
20
30
40
50
i

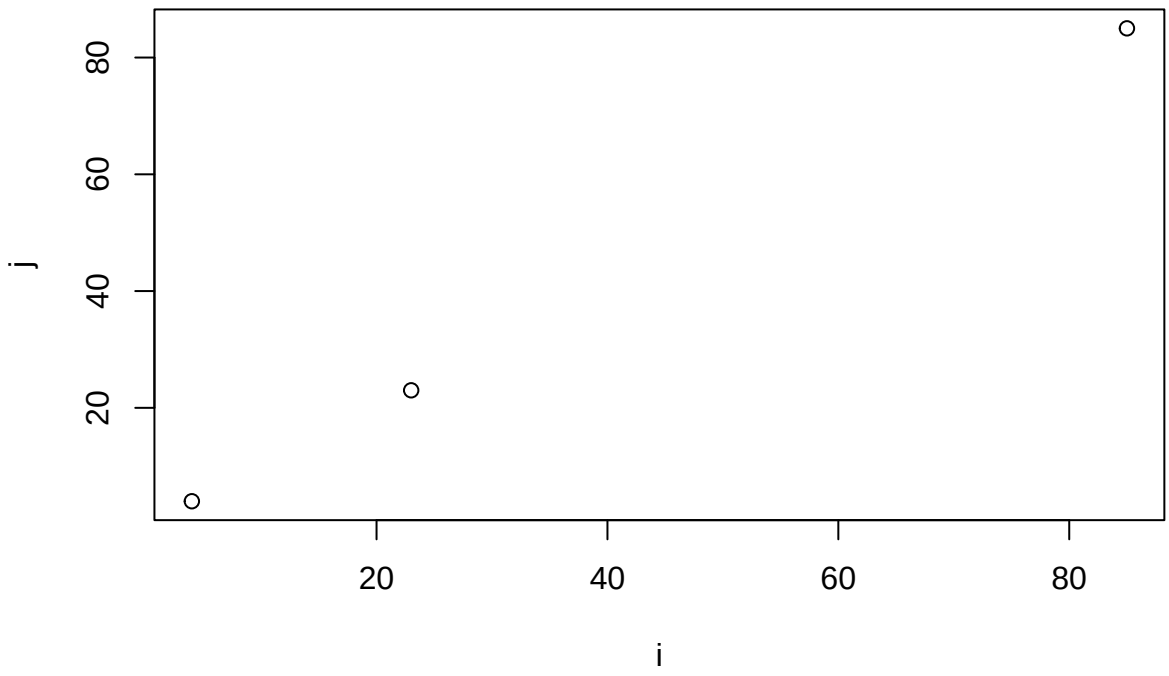

Correlación entre V8 y V10
j
40
50
60
70
10
20
30
40
50
i


Correlación entre V8 y V11
j
20
40
60
80
10
20
30
40
50
i


Correlación entre V9 y V10
j
40
50
60
70
30
40
50
60
70
80
90
i


Correlación entre V9 y V11
j
20
40
60
80
30
40
50
60
70
80
90
i


Correlación entre V10 y V11
j
20
40
60
80
40
50
60
70
i




```
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
```

```
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL
## NULL

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
```

```
##
## [[9]]
## NULL
##
## [[10]]
## NULL
##
## [[11]]
## NULL
##
## [[12]]
## NULL
##
## [[13]]
## NULL
##
## [[14]]
## NULL
##
## [[15]]
## NULL
##
## [[16]]
## NULL
##
## [[17]]
## NULL
##
## [[18]]
## NULL
##
## [[19]]
## NULL
##
## [[20]]
## NULL
##
## [[21]]
## NULL
##
## [[22]]
## NULL
##
## [[23]]
## NULL
##
## [[24]]
## NULL
##
## [[25]]
## NULL
##
## [[26]]
## NULL
```

```
##
## [[27]]
## NULL
##
## [[28]]
## NULL
##
## [[29]]
## NULL
##
## [[30]]
## NULL
##
## [[31]]
## NULL
##
## [[32]]
## NULL
##
## [[33]]
## NULL
##
## [[34]]
## NULL
##
## [[35]]
## NULL
##
## [[36]]
## NULL
##
## [[37]]
## NULL
##
## [[38]]
## NULL
##
## [[39]]
## NULL
##
## [[40]]
## NULL
##
## [[41]]
## NULL
##
## [[42]]
## NULL
##
## [[43]]
## NULL
##
## [[44]]
## NULL
```

```
##
## [[45]]
## NULL
##
## [[46]]
## NULL
##
## [[47]]
## NULL
##
## [[48]]
## NULL
##
## [[49]]
## NULL
##
## [[50]]
## NULL
##
## [[51]]
## NULL
##
## [[52]]
## NULL
##
## [[53]]
## NULL
##
## [[54]]
## NULL
##
## [[55]]
## NULL
##
## [[56]]
## NULL
```