Introducción al manejo de datos con R

Programación y Conceptos Estádisticos

Laura Valencia Daniel Jiménez

BIT

17 -02 -2021

R como calculadora

En R las sumas se realizan de la siguiente manera

```
100+200
```

```
## [1] 300
```

R como calculadora

Otra forma de hacer esta operación es a través de las asignaciones

```
a <- 12
b <- 55
c <- a+b
print(c)
```

```
## [1] 67
```

R como calculadora

Las demás operaciones pueden realizar de la siguiente manera

```
## [1] 101
```

```
## [1] 144
```

```
## [1] 250
```

```
## [1] 4
```

Operaciones Lógicas

En está sección se introducen los conceptos de lógica matemática.

Las operaciones lógicas son aquellas que intentan descubir valores de falso o verdadero como veremos en el siguiente ejemplo

```
4>8
```

```
## [1] FALSE
```

Operaciones Lógicas

Otra propiedad es la siguiente

```
3==9/3
```

```
## [1] TRUE
```

Operaciones Lógicas

Estas operaciones se pueden combinar y son muy útiles a la hora de trabajar con programación

```
3==9/3 & 2<3
```

```
## [1] TRUE
```

Otro ejemplo será el siguiente :

```
3==9/3 & 2<2
```

```
## [1] FALSE
```

Operaciones Lógicas

Suponga el siguiente ejemplo: Los Views de su última publicación son los siguientes

```
Views<-c(20,30,12,60,13)
```

Ahora suponga que quiere saber cuales son mayores a 50

```
Views>50
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE
```

Operaciones Lógicas

Para ver los datos de mayores de 50

```
Views[Views>50]
```

```
## [1] 60
```

Operaciones Lógicas

Ahora suponga que su publicación sale en dos tipos de revistas, Revista X y Revista Y, Ahora evalue ¿Cuál es mejor? por días

```
Revista_X<-c(30,45,23)
Revista_Y<-c(20,55,33)
Revista_X<=Revista_Y
```

```
## [1] FALSE  TRUE  TRUE
```

Operaciones Lógicas

El comando `if` se usa cuando queremos que una operación se ejecute bajo una condición

```
x<-5
if(x>3){
  print("El valor es mayor a tres")
}
```

```
## [1] "El valor es mayor a tres"
```

Operaciones Lógicas

`else` se usa para complementar alguna de las condiciones o ejecutar otra alternativa bajo el supuesto de que la condición inicial no pueda ser ejecutada

```
x<-50
y<-90

if(x>60){
  print("El número de x cumplio la condición")
} else {
  print("EL número de y cumplio la condición")
}
```

```
## [1] "EL número de y cumplio la condición"
```

Operaciones Lógicas

Un `loop` es una serie de repeticiones de procesos hasta que se cumpla una condición.

```
i <- 1
while (i < 6) {
print(i)
i = i+1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Operaciones Lógicas

la función `break` es la orden donde se rompe un loop cuando llega a un paso pre-determinado

```
x<-1:10
for(i in x){
  if(i==5){
    break
  }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

Operaciones Lógicas

Para que la anterior sentencia continue y omita el valor de break es necesario trabajar con `next`

```
x<-1:10
for(i in x){
  if(i==5){
    next
  }
  print(i)
}
```

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

Crear FUnciones

Para crear funciones se trabaja con el comando `function`

```
potencia<-function(x,y){
  z=x^y
  print(paste0("Este es el valor del número elevado ",z))
}
potencia(3,2)
```

```
## [1] "Este es el valor del número elevado 9"
```

Funciones logarítmicas

Las función es logarítmicas hacen parte del mundo del análisis matemático , en donde se estudia la propiedad de los números reales positivos dada una base con lo cual se logra obtener una mejor aproximación de la naturaleza de los datos.

Funciones logarítmicas

El logarítmo se puede calcular de manera directa

```
log(100)
```

```
## [1] 4.60517
```

O asignandole una base

```
log(100,base = 4)
```

```
## [1] 3.321928
```

Funciones exponenciales

Las funciones exponenciales, como su nombre lo indica, es elevar un número a un exponente

```
exp(3)
```

```
## [1] 20.08554
```

Esto lo usaremos en la parte de probabilidad.

Funciones con objetos

Una forma de trabajar en R es con la creación de listas las cuales se pueden desarrollar de la siguiente manera :

```
# Lista
a<-c(1:10)
print(a)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
b<-seq(0,10,by = 2)
print(b)
```

```
## [1]  0  2  4  6  8 10
```

Funciones con objetos

Para ordenar en R usamos el comando `sort`

```
c<-c(10,45,8,32)
sort(c)
```

```
## [1]  8 10 32 45
```

Extracción de componentes

En programación y en especial en el análisis de datos, la extracción de elementos es muy importante y es por ello que se explicará de dos maneras , la primera extraer un valor de un array y la segunda un segmento de un dataframe que en este caso será `iris` que viene por defecto en R

```
# Extraer elemento de un array

a<-c(1:10)
a[1] # Extrae el primer elemento
```

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

```
a[3] # Extrae el tercer elemento
```

```
## [1] 3
```

Extracción de componentes

```
iris[1,]# Extrae la primer Columna
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

```
iris[,1] # Extrae los valores de la primer fila
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Extracción de componentes

```
iris[1,2] # Extrae el segundo valor de la primer columna
```

```
## [1] 3.5
```

Funciones estadísticas

- Promedio: mean()
- Varianza: var()
- Desviación Estandar : sd()
- Percentiles: quantile()
- Covarianza: cov(,)
- Correlación de pearson: cor(,)
- Regresión : lm()
- Máximo: max ()
- Mínimo: min()

Funciones estadísticas con R

Ahora veremos algunas aplicaciones de estas funciones en R. Lo que usaremos para ello es un tipo de programación basado en `dplyr`, pero que tenga más paquetes para hacer el trabajo más ordenado, por lo tanto trabajaremos con la librería `tidyverse`.

Para empezar examinemos la naturaleza de los datos.

```
library(tidyverse)
iris%>%
  glimpse() # Función para ver el formato en que estan los da
```

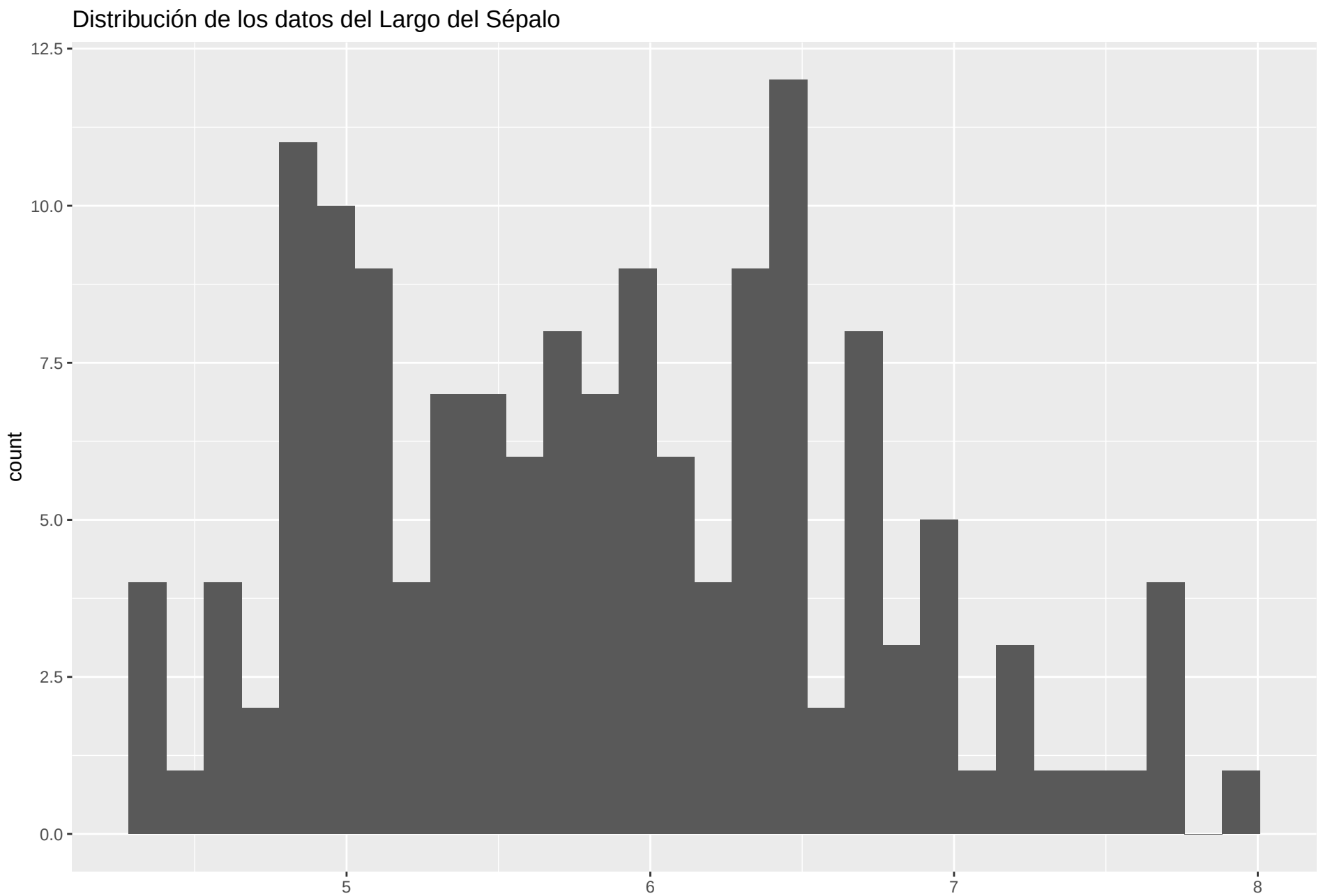
```
## Rows: 150
## Columns: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa
```

Funciones estadísticas con R

Ahora exploremos un poco del comportamiento de una de las variables, para esto es necesario y recomendable hacerlo a través de un histograma.

```
iris%>%
  ggplot(aes(Sepal.Length))+ # Función para gráficar
  geom_histogram() + # Geometría
  labs(title = 'Distribución de los datos del Largo del Sépalo
```

Funciones estadísticas con R



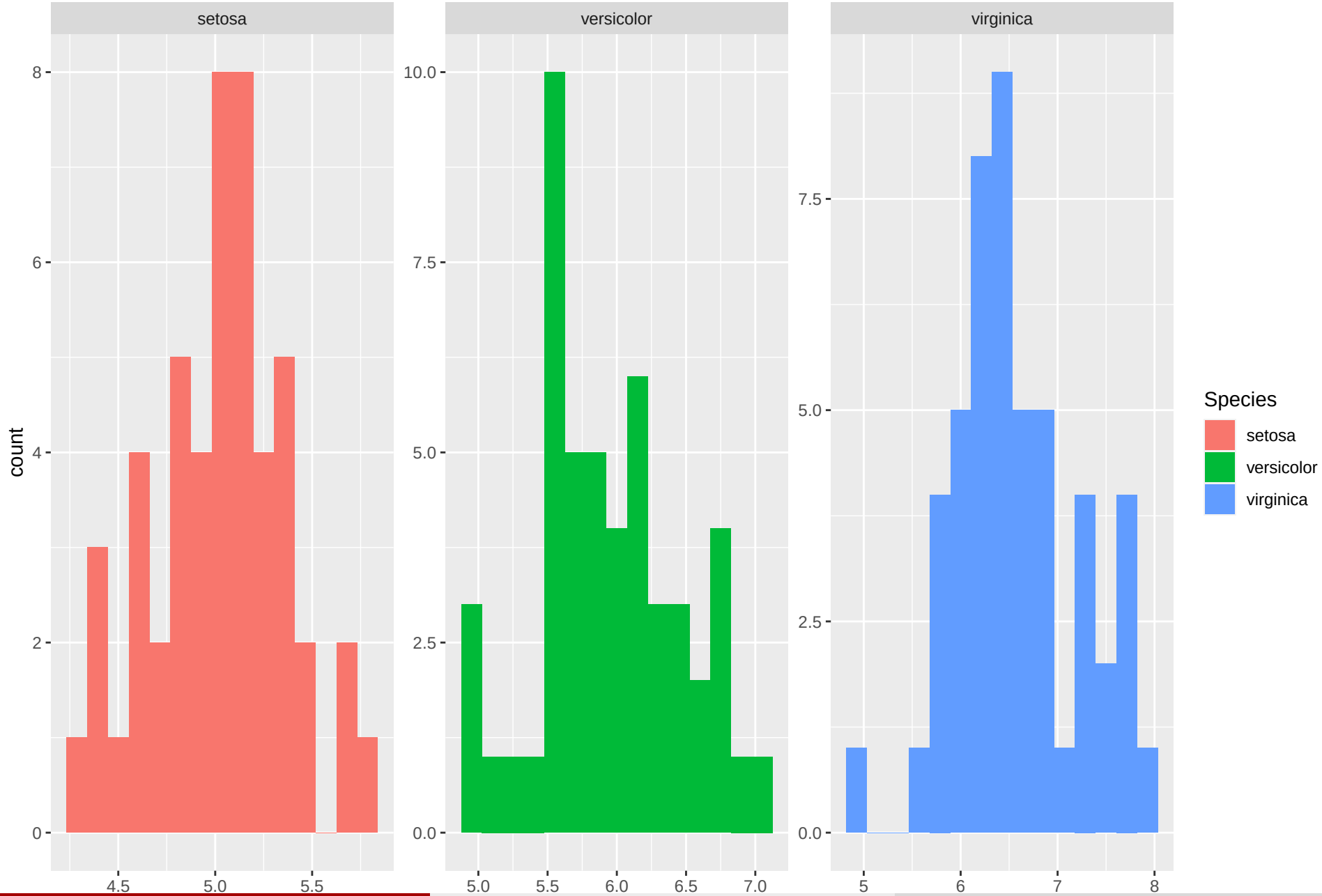
Funciones estadísticas con R

Ahora si lo quiere trabajar por especies puede usar la función `facet_wrap(~variable a distribuir)`. Es recomendable asignarle colores a cada una de las variables .

```
iris%>%
  ggplot(aes(Sepal.Length, fill=Species))+ # Función para gráf
  geom_histogram(bins = 15) + # Geometría
  facet_wrap(~Species, scales='free') # Distribución del gráf
  labs(title = 'Distribución de los datos del Largo del Sépalo
```

Con esto podremos saber a que tipo de distribución de probabilidad nos enfrentamos y entender más a fondo la naturaleza de los datos.

Funciones estadísticas con R



Funciones estadísticas con R

Ahora creemos un resumen estadístico

```
iris%>%
  group_by(Species)%>% # Se agrupa por especie
  summarize(Promedio=mean(Sepal.Length), # Se crea el promedi
            Mediana=median(Sepal.Length), # Mediana por espec
            p25=quantile(Sepal.Length)[2], # Quantil 25
            p75=quantile(Sepal.Length)[4]) # Quantil 75
```

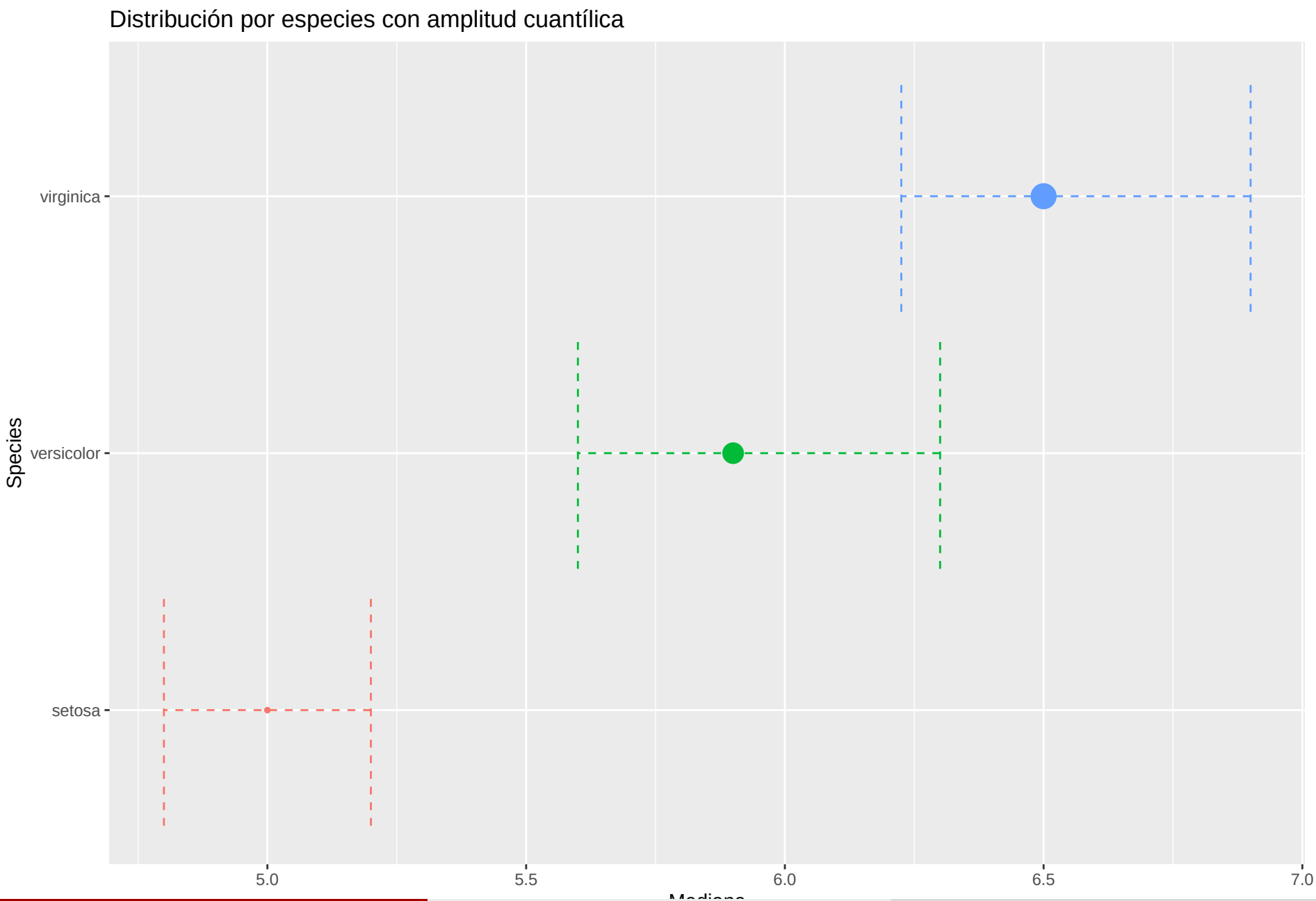
```
## # A tibble: 3 x 5
##   Species    Promedio Mediana   p25   p75
## * <fct>         <dbl>   <dbl> <dbl> <dbl>
## 1 setosa         5.01     5    4.8    5.2
## 2 versicolor     5.94     5.9  5.6    6.3
## 3 virginica      6.59     6.5  6.22   6.9
```

Esta estadística de resumen describe muy bien la naturaleza de los datos y con base a ella podemos hacer la siguiente inferencia gráfica

Funciones estadísticas con R

```
iris%>%
  group_by(Species)%>% # Se agrupa por especie
  summarize(Promedio=mean(Sepal.Length), # Se crea el promedi
            Mediana=median(Sepal.Length), # Mediana por espec
            p25=quantile(Sepal.Length)[2], # Quantil 25
            p75=quantile(Sepal.Length)[4])%>%
  ggplot(aes(Mediana,Species, color=Species))+
  geom_point(show.legend = FALSE,aes(size=Mediana))+
  geom_errorbar(aes(xmin=p25,xmax=p75),show.legend = FALSE,lin
  labs(title = 'Distribución por especies con amplitud cuantíl
```

Funciones estadísticas con R



Matrices

Ojo<- El mundo y todo lo que nos rodea es matricial, así que esto será fundamental más adelante.

Las matrices se crean con la función `matrix`

```
# Ejemplo tomado de `help(matrix)`
mdat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3, byrow =
               dimnames = list(c("row1", "row2"),
                               c("C.1", "C.2", "C.3")))
mdat
```

```
##      C.1 C.2 C.3
## row1   1   2   3
## row2  11  12  13
```

Matrices

A las matrices se les puede asignar un número de operaciones respetando sus reglas[1]

```
a<-matrix(c(1:10),nrow = 2)
a+a # Suma
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    6   10   14   18
## [2,]    4    8   12   16   20
```

```
a*2 # Multiplicación por si misma
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    6   10   14   18
## [2,]    4    8   12   16   20
```

[1]Para conocer las propiedades y las reglas de las matrices,por favor visite el siguiente link https://books.google.es/books?hl=es&lr=&id=el34KBt0tTwC&oi=fnd&pg=PR5&ots=MJIVJKr48y&sig=UkJ10vFbvmfeoMzrDNW06HIT3ac#v=onepage&q&f=false

Matrices

```
# Multiplicación entre matrices
a<-matrix(c(1,2,3,4,5,6),2,3)
b<-matrix(c(7,8,9))
a%*%b
```

```
##      [,1]
## [1,]   76
## [2,]  100
```

Matrices

```
a<-matrix(c(1,2,3,4),2,2)
#Promedio
mean(a)
```

```
## [1] 2.5
```

Matrices

```
# Resolver una matriz
solve(a)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

Matrices

```
# Hallar el determinante de una matriz
det(a)
```

```
## [1] -2
```

```
# trasponer una matriz

t(a)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

Matrices

Cuando veamos Machine Learning, la parte de valores propios sera muy importante, por ello solo se presentan en este momento.

```
eigen(a)
```

```
## eigen() decomposition
## $values
## [1]  5.3722813 -0.3722813
##
## $vectors
##            [,1]       [,2]
## [1,] -0.5657675 -0.9093767
## [2,] -0.8245648  0.4159736
```