Lecturas Importantes

1. Tutorial de gráficos en R, usando `ggplot2`
   `http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html`.

Preguntas

1. (a) Almacena el siguiente vector:

```
> f1<- c(13563,-14156,-14319,16981,12921,11979,9568,8833,-12968, 8133)
```

   Entonces realiza lo siguiente:

   - Muestra la salida todos los elementos de `f1` que, cuando se eleva a una potencia de 75, no son infinitos.
   - Devuelve los elementos de `f1`, excluyendo aquellos que resultan en infinito negativo cuando se eleva a una potencia de 75.

   (b) Almacena la siguiente matriz 3×4, como el objeto `varMatriz`

$$\begin{pmatrix} 77875.40 & 27551.45 & 23764.30 & -36478.88 \\ -35466.25 & -73333.85 & 36599.69 & -70585.69 \\ -39803.81 & 55976.34 & 76694.82 & 47032.00 \end{pmatrix}$$

   Entonces realiza lo siguiente:

   - Identifica los índices específicos de las entradas de `varMatriz` que son `NaN` cuando se eleva `varMatriz` a una potencia de 65 y se divide por infinito.
   - Devuelve los valores en `varMatriz` que NO son `NaN` cuando se eleva `varMatriz` a una potencia de 67 y se añade infinito al resultado. Confirma que esto es idéntico a identificar aquellos valores en `varMatriz` que, cuando aumentan a una potencia de 67, no son iguales al infinito negativo.
   - Identifique los valores en `varMatriz` que sean infinito negativo o finito cuando eleva `varMatriz` a una potencia de 67.

   (c) Considere la siguiente línea de código:

```
> f2 <- c(4.3,2.2,NULL,2.4,NaN,3.3,3.1,NULL,3.4,NA)
```

   Decide cuál de las siguientes afirmaciones son verdaderas y cuáles son falsas y luego use R para confirmar:

   - La longitud de `f2` es 8.
   - Llamando a `which(x=is.na(x=f2))`, no resultaá en 4 y 8.
   - Verificando `is.null(x=f2)`, proporciona la localización de dos valores NULL, presentes.

2. (a) Dada una lista cuyos miembros son vectores de cadena de caracteres de diferentes longitudes, utilice una función con `lapply` para pegar un signo de exclamación al final de cada elemento de cada miembro, con una cadena vacía como carácter de separación (tenga en cuenta que el comportamiento predeterminado de `paste` cuando se aplica a los vectores de caracteres). Ejecuta tus líneas de código en la lista dada por:

```
> f3 <- list("a",c("b","c","d","e"),"f",c("g","h","i"))
```

   (b) Escribe una función llamada `geolista` que puede buscar a través de una lista especificada y calcula las medias geométricas de cada miembro según las siguientes pautas:

- La función debe definir y utilizar una función de ayuda interna que devuelve la media geométrica de un argumento vectorial.
- Suponga que la lista sólo puede tener vectores numéricos o matrices numéricas como sus miembros. La función debe contener un bucle apropiado para inspeccionar a cada miembro de la lista a la vez.
- Si el miembro de la lista es un vector, calcula la media geométrica de ese vector, sobrescribiendo el miembro con el resultado, que debe ser un solo número.
- Si el miembro de la lista es una matriz, utilice un bucle implícito para calcular la media geométrica de cada fila de la matriz, sobrescribiendo el miembro con los resultados.
- La lista final debe ser devuelta al usuario.

Ahora, como una prueba rápida, compruebe que tu función coincida con las siguientes dos llamadas:

```
f4 <- list(1:3,matrix(c(3.3,3.2,2.8,2.1,4.6,4.5,3.1,9.4),4,2),
           matrix(c(3.3,3.2,2.8,2.1,4.6,4.5,3.1,9.4),2,4))
geolista(f4)
[[1]]
[1] 1.817121
[[2]]
[1] 3.896152 3.794733 2.946184 4.442972
[[3]]
[1] 3.388035 4.106080

f5 <- list(1:9,matrix(1:9,1,9),matrix(1:9,9,1),matrix(1:9,3,3))
geolista(f5)
[[1]]
[1] 4.147166
[[2]]
[1] 4.147166[[3]]
[1] 1 2 3 4 5 6 7 8 9
[[4]]
[1] 3.036589 4.308869 5.451362
```

(c) Escribe dos funciones de R: una que toma una función como argumento de entrada y otra que devuelve una función como salida.

3. (a) Supongamos que representamos el triángulo de Pascal como una lista, donde el elemento n es la fila n del triángulo. Por ejemplo, el triángulo de Pascal a la profundidad cuatro seria dado por:

```
> list(c(1), c(1, 1), c(1, 2, 1), c(1, 3, 3, 1))
```

La n-ésima fila se puede obtener de la fila $n-1$ añadiendo todos los pares adyacentes de números, luego prefijando y sufijando a 1. Escribe una función que, dado el triángulo de Pascal a la profundidad n, devuelve el triángulo de Pascal a la profundidad $n+1$. Comprueba que la undécima fila da los coeficientes binomiales $\binom{10}{i}$ para $i = 0, 1, \ldots 10$.

(b) Supongamos que necesitamos todas las 2^n secuencias binarias de longitud n. Una forma de generarlas es con bucles anidados. Por ejemplo, el siguiente código genera una matriz `binseq`, donde cada fila es una secuencia binaria diferente de longitud tres.

```
> binseq <- matrix(nrow = 8, ncol = 3)
> r <- 0 # fila actual de  binseq
> for (i in 0:1) {
+   for (j in 0:1) {
+     for (k in 0:1) {
+       r <- r + 1
```

```
+       binseq[r,] <- c(i, j, k)
+     }
+   }
+ }
```

```
binseq
[,1] [,2] [,3]
[1,] 0 0 0
[2,] 0 0 1
[3,] 0 1 0
[4,] 0 1 1
[5,] 1 0 0
[6,] 1 0 1
[7,] 1 1 0
[8,] 1 1 1
```

Claramente este enfoque será un poco tedioso para grandes valores de n. Una alternativa es utilizar la recursión. Supongamos que A es una matriz de tamaño $2^n \times n$, donde cada fila es una secuencia binaria diferente de longitud n. Entonces la siguiente matriz contiene todas las secuencias binarias de longitud $n+1$:

$$C = \left[\begin{array}{c|c} O & A \\ \hline 1 & A \end{array}\right]$$

Aquí **0** es un vector de ceros y **1** es un vector de unos.

Utiliza esta idea para escribir una función recursiva `binseq`, que toma como entrada un entero n y devuelve una matriz que contiene todas las secuencias binarias de longitud n, como filas de la matriz. En tu programa debes encontrar las funciones `cbind` y `rbind` particularmente útil.

4. (a) Corre

```
> ggplot(data = mpg)
```

¿Qué ves?.

(b) ¿Cuántas filas hay en `mpg`?. ¿ Cuántas columnas?.

(c) ¿Qué describe la variable `drv`?. Lee la ayuda `?mpg` para averiguarlo.

(d) Realiza un diagrama de dispersión de `hwy` vs `cyl`.

(e) ¿Qué sucede si haces un diagrama de dispersión de `class` vs drv?. ¿Por qué el gráfico no es útil?.

(f) ¿Qué ha salido mal con este código?. ¿Por qué los puntos no son azules?

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, color = "blue"))
```

(g) ¿Qué variables en `mpg` son categóricas?. ¿Qué variables son continuas?. ¿Cómo puedes ver esta información cuando se ejecuta `mpg`?.

(h) Asigne una variable continua a `color`, `size` y `shape`. ¿Cómo se comportan estas estéticas de manera diferente para variables categóricas y continuas?.

(i) ¿Qué sucede si se asigna la misma variable a estéticas múltiples?.

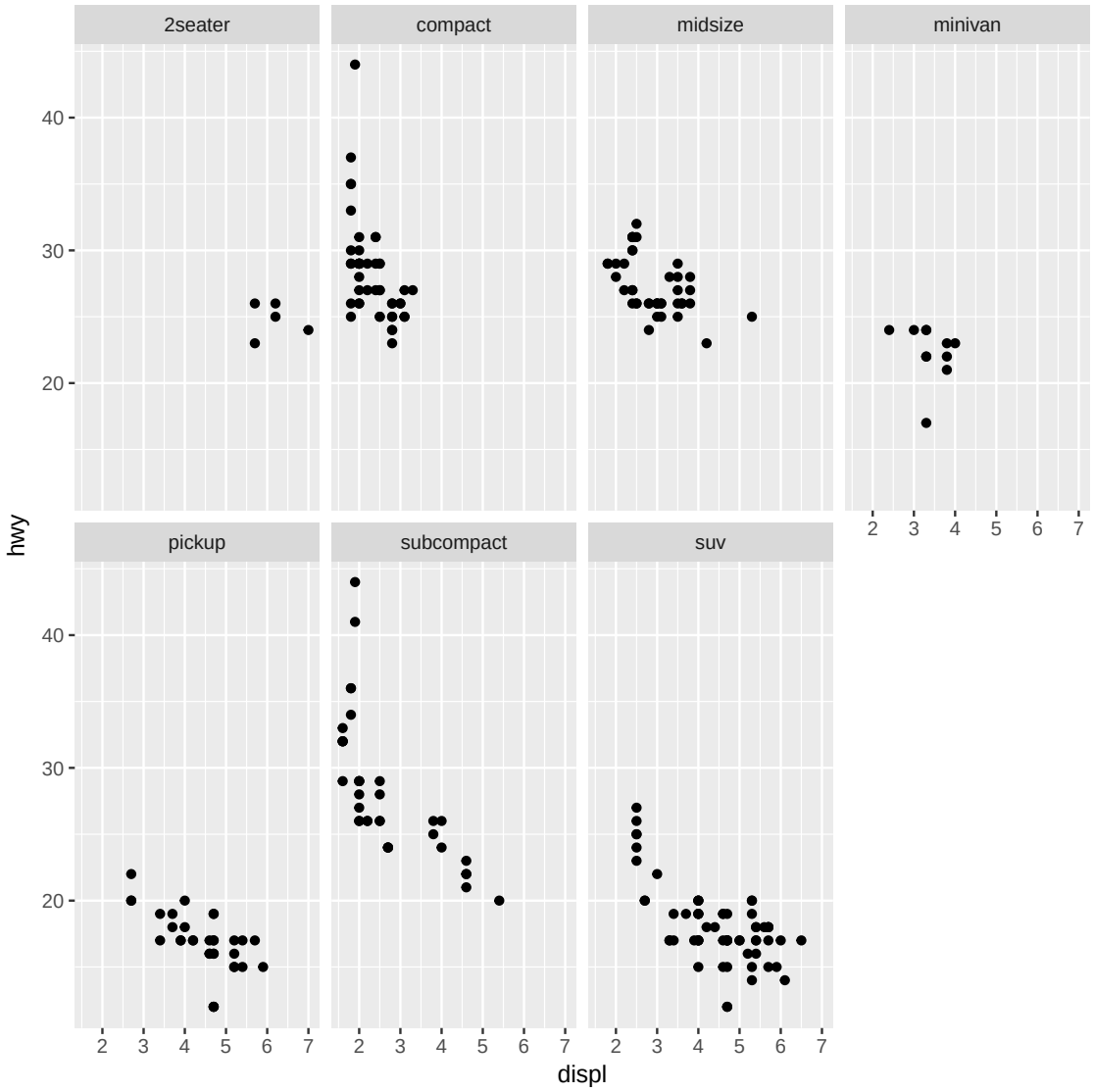
(j) ¿Qué hace la estética `stroke`?. ¿ Con qué formas trabaja?.

(k) ¿Qué sucede si se asigna una estética a algo que no sea un nombre de variable, como `aes(color = displ <5)`?.

(l) Una forma de agregar variables adicionales es con `aesthetics (estética)`. Otra forma, particularmente útil para las variables categóricas, es dividir el gráfico que resulta en `facets`, subgráficos donde cada uno muestra un subconjunto de los datos.

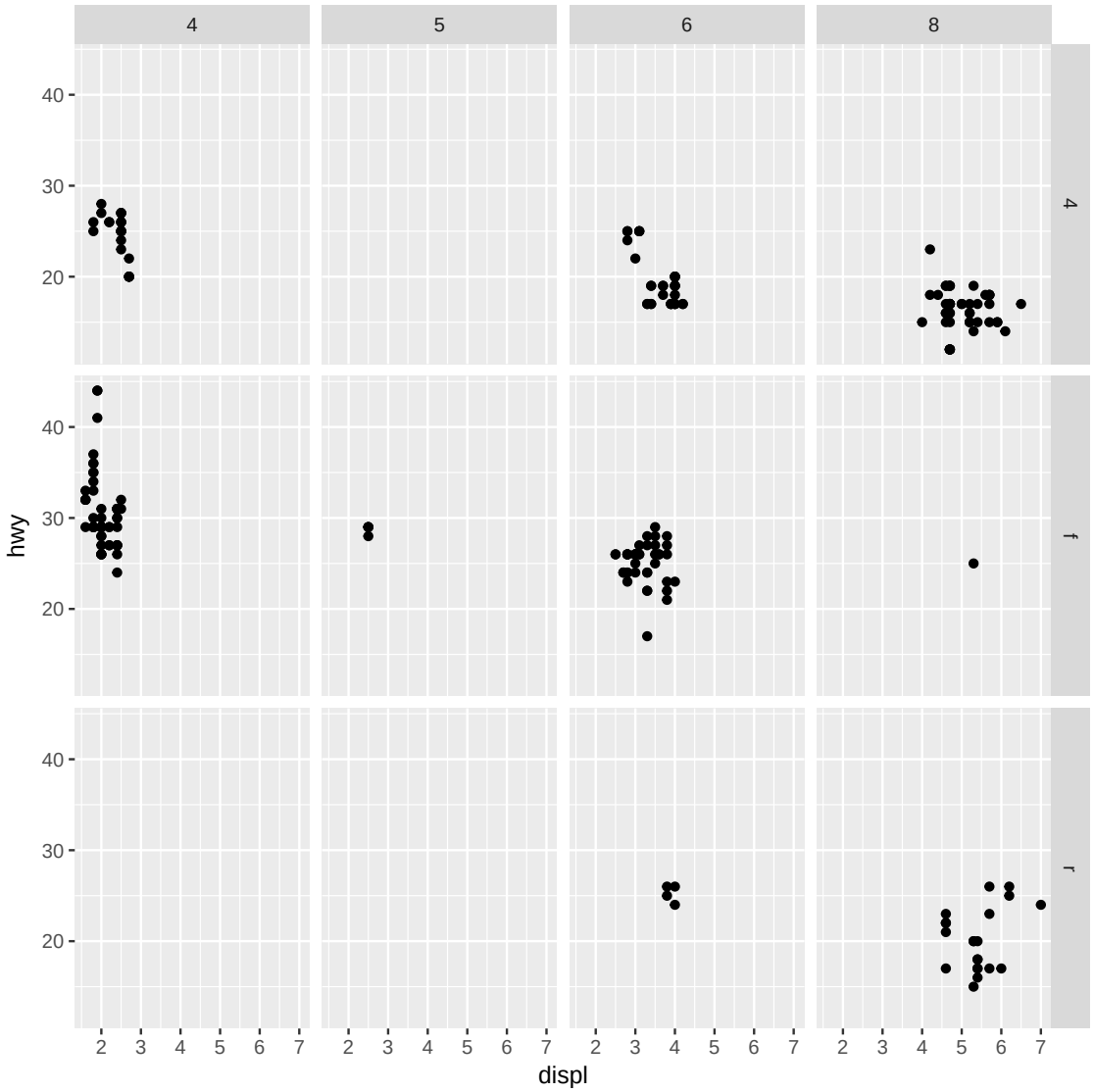
Para realizar `facets` a tu gráfico por una sola variable, usa `facet_wrap()`. El primer argumento de `facet_wrap()` debe ser una fórmula, que se crea con ∼ seguido de un nombre de variable (aquí "fórmula" es el nombre de una estructura de datos en R, no un sinónimo de "ecuación"). La variable que se pasa a `facet_wrap()` debe ser discreta.

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, nrow = 2)
```



Para realizar `facets` en la combinación de dos variables, agrega `facet_grid()` a la llamar al gráfico. El primer argumento de `facet_grid()` es también una fórmula. Esta vez la fórmula debe contener dos nombres de variables separados por un ∼.

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_grid(drv ~ cyl)
```



- ¿ Qué sucede si hace `facets` en una variable continua.
- ¿ Qué significan las celdas vacías en el gráfico con facet_grid(drv ~ cyl)?. ¿Cómo se relacionan con el gráfico?.

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = drv, y = cyl))
```

- ¿Qué gráficos hace el siguiente código?. ¿Qué hace el punto . ?.

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_grid(drv ~ .)
>
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_grid(. ~ cyl)
```

(m) Lee `?facet_wrap`.¿Qué hace `nrow`?. ¿ Qué hace `ncol`?.¿Qué otras opciones controlan la disposición gráfica?. ¿Qué no hace `facet_grid`, tiene los argumentos `nrow` y `ncol`?.

(n) Cuando se utiliza `facet_grid()` normalmente se debe colocar la variable con más niveles únicos en las columnas. ¿Por qué?.

5. Resuelve las siguientes preguntas:

   (a) ¿Qué ocurre a un factor, cuando se modifica sus niveles?

```
> f1 <- factor(letters)
> levels(f1) <- rev(levels(f1))
```

   (b) ¿Qué hace este código?. ¿Cómo f2 y f3 difieren de f1?.

```
> f2 <- rev(factor(letters))
> f3 <- factor(letters, levels = rev(letters))
```

   (c) ¿Qué atributos posee un data frames?.

   (d) ¿Qué hace `as.matrix()` cuando se aplica a un data frame con columnas de diferentes tipos?.

   (e) ¿ Se puede tener un data frame de datos con 0 filas?. ¿Qué hay de 0 columnas?.

   (f) ¿Qué devuelve `upper.tri()`?. ¿Cómo funciona los subconjuntos de una matriz con esta función?. ¿Necesitamos reglas de subconjunto adicionales para describir su comportamiento?.

   (g) Implemente tu propia función que extrae las entradas diagonales de una matriz (debe comportarse como `diag(x)` donde x es una matriz).

   (h) ¿Cómo cambiarías aleatoriamente las columnas de un data frame? (Esta es una técnica importante en los bosques aleatorios). ¿Puedes permutar las filas y las columnas simultáneamente en un solo paso?.

   (i) ¿Cómo seleccionarias una muestra aleatoria de m filas de un data frame de datos?. ¿Qué pasa si la muestra tiene que ser contigua (es decir, con una fila inicial, una fila final y cada fila entre ellas)?.

   (j) ¿Cómo podrías colocar las columnas en un data frame de datos en orden alfabético?.

   (k) Enumera tres formas en que un entorno difiere de una lista.

   (l) Usando `parent.env()` y un bucle (o una función recursiva), verifica que los predecesores de `globalenv ()` incluyan `baseenv()` y `emptyenv()`. Utiliza la misma idea básica para implementar tu propia versión de `search()`.