



# Tidyverse

VERÓNICA JIMÉNEZ JACINTO

UNIDAD UNIVERSITARIA DE SECUENCIACIÓN MASIVA Y BIOINFORMATICA

INSTITUTO DE BITECNOLOGIA

UNAM

# Objetivos

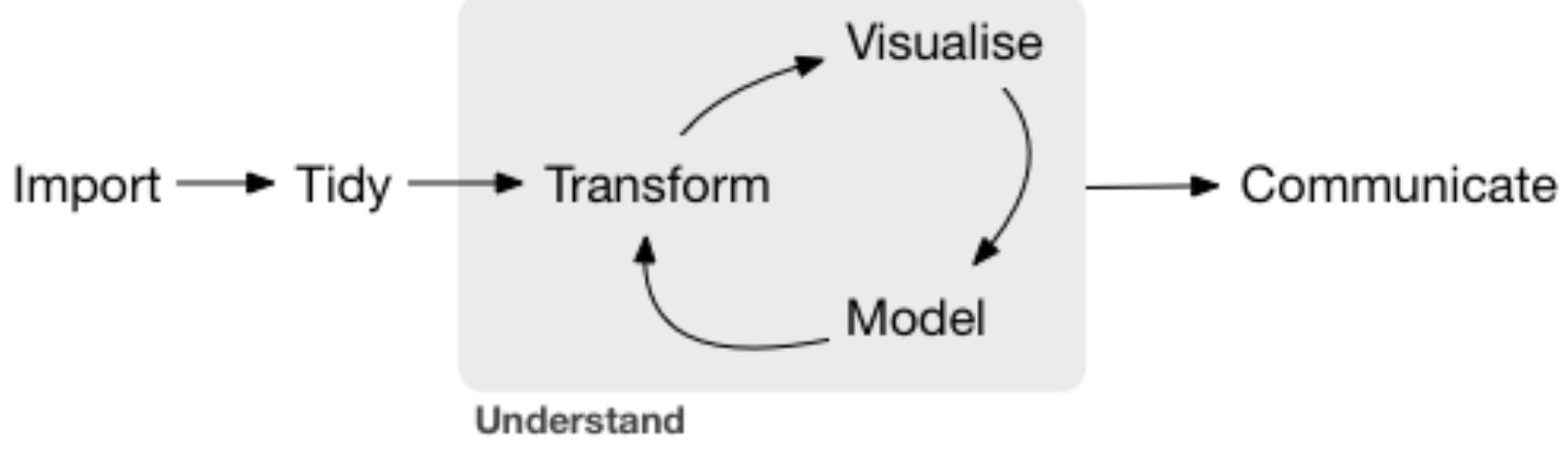
La ciencia de datos es una disciplina emocionante que permite convertir los datos sin procesar en conocimiento.

En esta sesión daremos una mirada a Tidyverse, y específicamente a dplyr para ordenar y trabajar con datos con miras a analizarlos.

# INDICE

- ▶ Modelo de herramientas necesarias para la ciencia de datos. Y el Big Data
- ▶ Una mirada rápida a los paquetes de tidyverse
- ▶ El paquete dplyr
  - ▶ Creando Tibbles
    - ▶ Tipos de columnas en tibble
  - ▶ Filtrando renglones filter()
  - ▶ Ordenando valores
  - ▶ Seleccionando columnas
  - ▶ Agregando nuevas variables
  - ▶ Summary
  - ▶ Group by
- ▶ Uso de pipes
- ▶ Estilos de programacion

# Modelo de herramientas necesarias



Fuente: <http://r4ds.had.co.nz/introduction.html>

# Tidy

- ▶ La información ordenada es importante porque la estructura coherente nos permite centrar el esfuerzo en las preguntas sobre los datos, no luchando para obtener los datos en la forma correcta para las diferentes funciones...
- ▶ Por ejemplo considere un formato Genebank como fuente de nuestra información de entrada:

# GeneBank

## Copies and Copyright-Notice

User is not entitled to change or erase data sets of the RegulonDB database or to eliminate copyright notices from RegulonDB. Furthermore, User is not entitled to expand RegulonDB or to integrate RegulonDB or as a part of other databases or bank systems, without prior written permission from CIBER-BBN.

gene

190..255

/gene="thrL"

/locus\_tag="b0001"

/note="synonym: EG11277"

/db\_xref="GeneID:944742"

CDS

190..255

/gene="thrL"

/locus\_tag="b0001"

/function="leader; Amino acid biosynthesis: Threonine"

/note="go\_process: threonine biosynthesis [goid 0009088]"

/codon\_start=1

/transl\_table=11

/product="thr operon leader peptide"

/protein\_id="NP\_414542.1"

/db\_xref="ASAP:6"

/db\_xref="GI:16127995"

/db\_xref="GeneID:944742"

/translation="MKRISTTITTTITTTGNGAG"

## Citation

User is committed to cite properly the use of RegulonDB. The complete update publication concerning RegulonDB is:

Sergio Gama-Castro, Verónica Jiménez-Jacinto, Martín Peralta-Gil, Alberto Santos-Zavala, Mónica I. Peñalva-Sprinkle, Bruno Contreras-Moreira, Juan Segura-Salazar, Luis Muñoz-Rascado, Irma Martínez-Flores, Saúl Sánchez, César Benítez-Martínez, César Abreu-Goodger, Carlos Rodríguez-Páramo, Juan Miranda-Ríos, Enrique Morett, Arcelia M. Huerta, Luis Treviño-Gómez, and Julio Collado-Vides.

"RegulonDB (Version 6.0): gene regulation model of Escherichia coli integrating transcription active (experimental) annotated promoters and expression navigation".

Nucleic Acids Research, 2008, vol 36, D920-D924

Release 6.4 Date: 10-AUG-08

## Transcription Factor Matrix and Alignments

The consensus and palser programs were used to create the matrix and alignment.

Transcription Factor NameAdA  
Total of 189 binding sites3

## Matrix

A	2	3	1	1	3	2	3	0	0	1	1	2	0	0	1	0
C	1	0	0	1	0	1	0	0	1	2	0	0	2	0	1	1
G	0	0	1	0	0	0	1	0	2	0	0	3	1	2	0	0
T	0	0	1	0	0	0	2	2	0	0	1	0	0	2	0	1

## Alignment Score

AAGCAAGCCACCGCTCTGAATACGTT20.66  
AAAAATAATTAAAGCGCAAGATGTGGT121.42  
CATTACATTGCTGGATAAAGATGTTTAG19.78

- ▶ El primer esfuerzo será ordenar la información de manera que en cada renglon tengamos la descripción de un objeto y en cada columna los atributos para describir los objetos

Gene_id	Gene Name	Poslef	PosRigth	Prod_id	Prod_name	Prod_sym	W_molec
ECK120001	thrL	190	255	ECK12P01	thrL	ThrL	2.138
ECK120002	thrA	337	2799	ECK12P02	ThrA	ThrD	89.12
ECK120003	thrB	2801	3733	ECK12P03	ThrB		33.62

# Transformaciones

- ▶ La transformación incluye resumir las observaciones de interés (como todas las personas en una ciudad, o todos los datos del último año), crear nuevas variables que son funciones de variables existentes (como calculo de la velocidad, a partir de la distancia y tiempo) y calcular un conjunto de resumen estadístico.

<u>Gene_id</u>	<u>Gene Name</u>	<u>Posleft</u>	<u>PosRigth</u>	<u>Size</u>
ECK120001	<u>thrL</u>	190	255	65
ECK120002	<u>thrA</u>	337	2799	2463
ECK120003	<u>thrB</u>	2801	3733	932

# Visualización

- ▶ La visualización es una actividad fundamentalmente humana.
- ▶ Una buena visualización muestra cosas que no esperabas, o plantea nuevas preguntas sobre los datos.
- ▶ Una buena visualización también podría indicar que estas haciendo la pregunta incorrecta, o necesita recolectar datos diferentes.
- ▶ Las visualizaciones pueden sorprenderte, pero no escalan particularmente bien porque requieren un ser humano para interpretarlas.

# Visualización



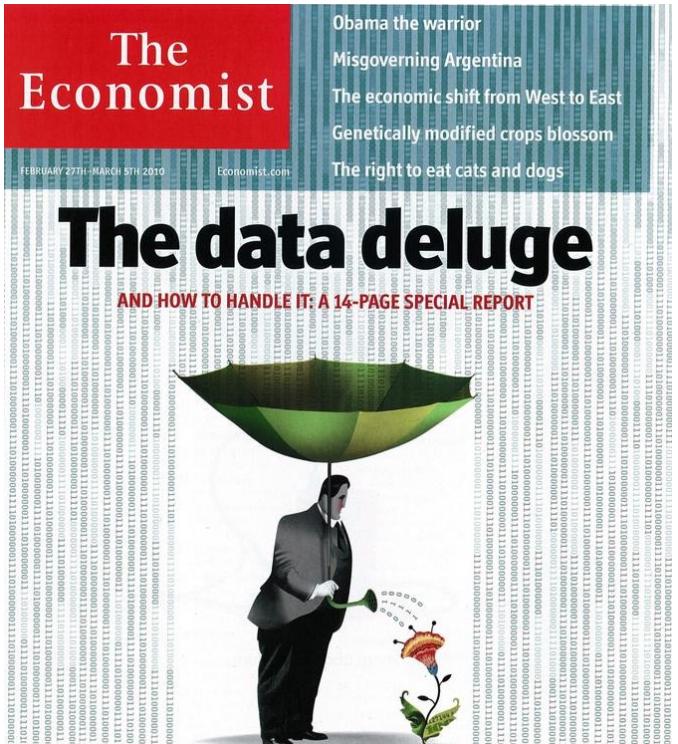
# Modelado

- ▶ Los modelos son herramientas complementarias para la visualización.
- ▶ Una vez que hayas hecho las preguntas lo suficientemente precisas, puedes usar un modelo para responderlas.
- ▶ Los modelos son una herramienta fundamentalmente matemática o computacional, por lo que generalmente se escalan bien.
- ▶ Incluso cuando no lo hacen, generalmente es más barato comprar más computadoras que comprar más cerebros

# Comunicación

- ▶ No importa qué tan bien tus modelos y visualización te hayan llevado a comprender los datos a menos que también puedas comunicar tus resultados a otros.
- ▶ La programación es una herramienta transversal que se utiliza en cada parte del proyecto.
- ▶ No necesitas ser un programador experto para ser un científico de datos, pero... aprender más acerca de la programación vale la pena porque convertirse en un mejor programador te permite automatizar tareas comunes y resolver nuevos problemas con mayor facilidad.

# BIG DATA





dplyr

R Studio

R

C

FORTRA  
N

OS

# Paquete tidyverse

- ✓ **GGPLOT2:** GRAFICOS
- ✓ **PURRR :** PROGRAMACION FUNCIONAL
- ✓ **TIBBLE :** ITERADOR SOBRE DATAFRAME
- ✓ **DPLYR:** ORDENA, SELECCIONA, MODIFICA
- ✓ **TIDYR :** REARREGLAR DATOS
- ✓ **STRINGR :** CADENAS DE CARACTERES
- ✓ **READR :** IMPORTAR DATOS DE ARCHIVOS
- ✓ **FORCATS:** VARIABLES CATEGORICAS

# Paquete dplyr

## INTRODUCCIÓN

# Prerequisitos

- ▶ Debemos tener instalado y cargado el paquete tidyverse

```
install.packages('tidyverse')
```

```
library(tidyverse)
```

```
— Attaching packages ————— tidyverse 1.2.1 —
✓ ggplot2 3.0.0    ✓ purrr   0.2.5
✓ tibble  1.4.2    ✓ dplyr   0.7.6
✓ tidyr   0.8.1    ✓ stringr 1.3.1
✓ readr   1.1.1    ✓forcats 0.3.0
— Conflicts ————— tidyverse_conflicts() —
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()   masks stats::lag()
```

# dplyr

- ▶ dplyr es un poderoso paquete de R para transformar y sumarizar datos tabulares en R
- ▶ Tiene un conjunto de funciones (o verbos) que realizan las operaciones mas comunes en el análisis de datos como son :
  - ▶ Filtrar por renglones
  - ▶ Seleccionar columnas específicas
  - ▶ Reordenar renglones
  - ▶ Adicionar nuevas columnas
  - ▶ Sumarizar datos

# Tipos de columnas en tibble

- ▶ **int** enteros.
- ▶ **dbl** para doubles, o numeros real.
- ▶ **chr** para vectores de caracteres vectors, o strings.
- ▶ **date** para fechas.
- ▶ **dttm** para date-times (una fecha + un tiempo).
- ▶ **lgl** para valores logicos, vectores de TRUE/FALSE.
- ▶ **fctr** Para *factors*, (R representa variables categóricas –con un numero fijo de valres posibles–)

# Data: mammals sleep

- ▶ Contienen los tiempos de sueño y peso de un conjunto de 83 mamíferos, descritos con las siguientes 11 columnas

- ▶ name common name
- ▶ genus taxonomic rank
- ▶ vore carnivore, omnivore or herbivore?
- ▶ order taxonomic rank
- ▶ conservation the conservation status of the mammal
- ▶ sleep\_total total amount of sleep, in hours
- ▶ sleep\_rem rem sleep, in hours (rem=rapid eye movement)
- ▶ sleep\_cycle length of sleep cycle, in hours
- ▶ awake amount of time spent awake, in hours
- ▶ brainwt brain weight in kilograms
- ▶ bodywt body weight in kilograms

# Preparando los datos

```
# Leemos la tabla en formato csv  
msleep <-  
read.csv("Data/msleep_ggplot2.csv")  
  
head(msleep)  
# transformamos el data.frame a tibble  
tmsleep<-as.tibble(msleep)
```

# dplyr

Existen 5 funciones claves para transformar datos con dplyr que permiten resolver la mayoría de los retos en el proceso de manipulación de datos:

- ▶ Elegir las observaciones por sus valores (filter())
- ▶ Reordenar los renglones (arrange())
- ▶ Elegir las variables por sus nombres (select())
- ▶ Crear nuevas variables con funciones sobre las variables existentes (mutate())
- ▶ Colapsar muchos valores en un resumen de los mismos (summarise())

# dplyr

dplyr puede ser vista como una gramática de manipulación de datos, donde cada función es un verbo y cada verbo trabaja de manera similar:

1. El primer argumento es el nombre del data.frame o tibbles
2. Los siguientes argumentos describen qué se hace con el data.frame usando los nombres de variables.
3. El resultado es un nuevo data.frame o tibbles

# Filtrando renglones con filter()

- ▶ Filter() permite filtrar observaciones a partir de sus valores. El primer argumento es el nombre de data frame. El segundo y los siguientes argumentos son las expresiones de filtro. Por ejemplo: Filtremos a todos los mamíferos carnívoros que duerman mas de 10 horas

```
filter(tmsleep, vore=='carni', sleep_total>10)
# A tibble: 11 x 11
   name           genus     vore   order   conservation sleep_total
   <fct>          <fct>    <fct>  <fct>      <fct>        <dbl>
 1 Cheetah        Acinonyx  carni Carnivora  lc            12.1
 2 Dog             Canis     carni Carnivora  domesticated 10.1
 3 Long-nosed armadillo Dasypus  carni Cingulata  lc            17.4
 4 Domestic cat    Felis     carni Carnivora  domesticated 12.5
 5 Thick-tailed opossum Lutreoli... carni Didelphimo... lc            19.4
 6 Slow loris      Nyctibeus carni Primates   <NA>           11
 ...

```

# Ordenando renglones con arrange()

- ▶ `Arrange()` trabaja similar a `filter()` excepto que en lugar de seleccionar renglones, cambia su orden.
- ▶ Ordena los renglones de un `data.frame` a partir de un conjunto de columnas especificadas (o una expresión mas complicada)
- ▶ Si seleccionas mas de una columna, cada columna adicional será ordenada en dependencia de los valores de la columna anterior

# Ordenando

- ▶ Ordenamos en función del peso del cerebro, el peso del cuerpo, el total de horas que se duerme...

```
arrange(tmsleep, slllep_total)
# A tibble: 83 x 11
  name          genus ...sleep_total sleep_rem sleep_cycle awake
brainwt bodywt
<fct>      <fct>    ... <dbl><dbl><dbl> <dbl> <dbl> <dbl>
1 Lesser short-tailed... Cryptotis ... 9.1 1.4 0.15 14.9 1.40e-4 0.005
2 Little brown bat     Myotis   ... 19.9 2   0.2   4.1  2.50e-4 0.01 
3 Greater short-tailed... Blarina  ... 14.9 2.3  0.133 9.1  2.90e-4 0.019
4 Big brown bat       Eptesicus ... 19.7 3.9  0.117 4.3  3.00e-4 0.023
...
...
```

# Seleccionando columnas con select()

- ▶ No es poco común tener conjuntos de datos con miles o varios cientos de variables. En este caso, lo primero que queremos cambiar es enfocarnos en las variables en las que estamos interesados en este momento.
- ▶ Select() nos permite hacer un “zoom” en un subconjunto usando operaciones sobre los nombres de las variables

# Select()

29

- ▶ Seleccionemos del nombre a la conservación:

```
select(tmsleep, name:conservation)
```

```
# A tibble: 83 x 5
```

	name	genus	vore	order
	conservation			
1	<fct>	<fct>	<fct>	<fct>
2	Cheetah	Acinonyx	carni	Carnivora
3	Owl monkey	Aotus	omni	Primates
4	Mountain beaver	Aplodontia	herbi	Rodentia
5	Greater short-tailed shrew	Blarina	omni	Soricomorp
6	Cow	Bos	herbi	Artiodacty
7	Three-toed sloth	Bradypus	herbi	Pilosa
8	Northern fur seal	Callorhinus	carni	Carnivora
9	Vesper mouse	Calomys	<NA>	Rodentia
10	Dog	Canis	carni	Carnivora
11	Roe deer	Capreolus	herbi	Artiodacty
	# ... with 73 more rows			

# Agregando nuevas variables con mutate()

- ▶ Además de seleccionar conjuntos de columnas existentes, es frecuentemente útil adicionar nuevas columnas que son funciones de las columnas existentes. Ese es el trabajo de mutate()
- ▶ mutate() permite adicionar nuevas columnas al final de tu dataset así que empezaremos creando un conjunto de datos más estrecho para que podamos ver las nuevas variables
- ▶ Recuerda que en R Studio la manera más fácil de ver todas las columnas es con View()

# Mutate()

```
sleep_sml<-select(tmsleep, name,starts_with("sleep"),awake)
# A tibble: 83 x 5
  name          sleep_total sleep_rem sleep_cycle awake
  <fct>        <dbl>      <dbl>      <dbl>      <dbl>
1 Cheetah      12.1       NA         NA         11.9
2 Owl monkey   17          1.8        NA          7
3 Mountain beaver 14.4      2.4        NA         9.6
4 Greater short-ta. 14.9      2.3        0.133     9.1
5 Cow           4           0.7        0.667     20
6 Three-toed sloth 14.4      2.2        0.767     9.6
7 Northern fur seal 8.7       1.4        0.383    15.3
8 Vesper mouse  7           NA         NA         17
9 Dog            10.1       2.9        0.333    13.9
10 Roe deer     3           NA         NA         21
# ... with 73 more rows
```

```
mutate(sleep_sml, total= sleep_total+awake,
prop=sleep_rem/sleep_total)
```

# Mutate()

Y te puedes referir a las variables que justo acabas de crear:

```
mutate(sleep_sm1,prop=sleep_rem/sleep_total, porcent=prop*100)
# A tibble: 83 x 7
  name sleep_total sleep_rem sleep_cycle awake   prop porcent
  <fct>     <dbl>      <dbl>       <dbl> <dbl>  <dbl>    <dbl>
1 Cheetah    12.1        NA        NA      11.9  NA      NA
2 Owl monkey 17          1.8        NA        7     0.106  10.6
3 Mountain b.14.4        2.4        NA        9.6   0.167  16.7
4 Greater sh.14.9        2.3       0.133      9.1   0.154  15.4
5 Cow         4           0.7       0.667     20    0.175  17.5
6 Three-toed 14.4        2.2       0.767      9.6   0.153  15.3
7 Northern fu 8.7         1.4       0.383     15.3   0.161  16.1
8 Vesper mouse 7          NA        NA        17    NA      NA
9 Dog         10.1        2.9       0.333     13.9   0.287  28.7
10 Roe deer    3           NA        NA        21    NA      NA
# ... with 73 more rows
```

# Resumenes agrupados con summarise()

- ▶ El ultimo verbo es summarize(). Colapsa un data.frame a un simple renglón:

```
summarize(sleep_sml,  
sleep=mean(sleep_total),brain=mean(sleep_rem))  
# A tibble: 1 x 2  
#   sleep brain  
#   <dbl> <dbl>  
# 1 10.4    NA  
  
summarize(sleep_sml,  
sle=mean(sleep_total),br=mean(sleep_rem,na.rm=TRUE))  
# A tibble: 1 x 2  
#   sleep brain  
#   <dbl> <dbl>  
# 1 10.4  1.88
```

# dplyr

Estas funciones pueden ser usadas en conjunción con `group_by()` la cual cambia el ámbito de cada función para operar sobre el conjunto completo de datos u operando solo grupo por grupo.

state	city	revenue
Maharashtra	Achalpur	10000
Uttar Pradesh	Achhnera	20000
Gujarat	Adalaj	10000
Uttar Pradesh	Agra	30000
Gujarat	Ahmedabad	40000
Maharashtra	Ahmednagar	10000
Maharashtra	Akola	40000



state	sum(revenue)
Maharashtra	60000
Uttar Pradesh	50000
Gujarat	50000

# Resumenes agrupados con group\_by()

- ▶ summarise() es mucho mas util cuando se usa a lapar de group\_by. Esto cambia la unidad de análisis del grupo completo de datos a grupos individuales.

```
vore<-group_by(msleep,vore)
summarize(vore,mean(sleep_total),mean(sleep_rem,na.rm=TRUE))
# A tibble: 5 x 3
#> #>   `mean(sleep_total)` `mean(sleep_rem, na.rm = TRUE)` 
#> #>   <dbl>                  <dbl>
#> 1 carni                 10.4                2.29
#> 2 herbi                 9.51                1.37
#> 3 insecti                14.9                3.52
#> 4 omni                  10.9                1.96
#> 5 <NA>                  10.2                1.88
```

# UsO de PIPES

# Pipes

Las tuberías (pipes) son una poderosa herramienta que hace mas claro y fácil aplicar una secuencia de operaciones. Ahora las examinaremos mas a detalle para aprender muchas otras ventajas de trabajar con tuberías (pipes)

Las tuberías %>% vienen en el paquete **magrittr**, pero cuando cargas el paquete tidyverse se carga automaticamente %>%, sino fuera el caso, puedes hacer explicitamente:

```
library (magrittr)
```

# Combinando múltiples operaciones con el pipe

Por ejemplo:

1. Tome la tabla msleep
2. Selecciona el nombre y el peso del cerebro y del cuerpo
3. Aplique un head para ver el resultado.

```
msleep %>% select(name,brainwt,bodywt) %>% head  
  name brainwt bodywt  
1 Cheetah      NA 50.000  
2 Owl monkey  0.01550 0.480  
3 Mountain beaver      NA 1.350  
4 Greater short-tailed shrew 0.00029 0.019  
5 Cow          0.42300 600.000  
6 Three-toed sloth     NA 3.850
```

## Pipe %>%

El concepto de pipe (tuberías) es el mismo que en Unix. En este caso %> % representa el pipe, el cual indica que queremos usar el comando de la izquierda como entrada al comando de la derecha del pipe.

# Combinando múltiples operaciones con el pipe

- ▶ Hay otra forma de abordar el mismo problema con tuberías (pipes) %>%

```
(mean_sleep_vore<-msleep %>%
  group_by(vore) %>%
  summarize(
    count=n(),
    sleep=mean(sleep_total),
    rem=mean(sleep_rem,na.rm=TRUE)
  ) %>%
  filter(count>10))

# A tibble: 3 x 4
  vore   count       sleep`       rem
  <fct> <int>      <dbl>      <dbl>
1 carni     19      10.4       2.29
2 herbi     32      9.51       1.37
3 omni     20      10.9       1.96
```

# Combinando múltiples operaciones con el pipe

Entre bastidores,

`x %>% f(y)` se convierte en `f(x, y)`

`x %>% f(y) %>% g(z)` se convierte en `g(f(x, y), z)`

# Alternativas de las tuberías

- ▶ El punto de las tuberías es ayudarte a escribir código de una manera fácil para leer y entender. . Para terminar de ver porque las tuberías son también útiles, exploraremos un numero de maneras de escribir el mismo código. Contemos la historia de un pequeño conejo llamado Foo:

Little bunny Foo Foo

Went hopping through the forest

Scooping up the field mice

And bopping them on the head

# Alternativas de las tuberías

Pimpón es un muñeco muy guapo y de  
Cartón

Se lava la carita con agua y con jabón

Se desenreda el pelo con peine de marfil

Y aunque se de estirones

No llora, ni hace así.

# Alternativas de las tuberías

- ▶ Empezamos por crear un objeto que represente a Pimpón.

```
Pimpón<-es_un_muneco(muy_guapo, carton)
```

- ▶ Y necesitamos funciones para cada verbo:

```
lava(), desenreda(), estiron(),  
NoLlora(), HaceAsi()
```

Usando este objeto y sus verbos hay (al menos) 4 maneras de recountar la historia en código:

1. Salvar cada paso intermedio como un nuevo objeto
2. Sobrescribir el objeto original
3. Composición de funciones
4. Uso de pipes

# 1. Pasos intermedios

- ▶ La mas simple aproximación es salvar cada paso como un nuevo objeto:

```
Pimpon_1<-lava(Pimpon,  
que=Carita,conque=Aqua,Jabon)  
Pimpom_2<-desenreda(Pimpom_1, Que=Pelo,  
conque=PeineMarfil)  
Pimpom_3<-NoLlora(Pimpom_2)  
Pimpom_4<-NiHaceAsi(Pimpom_3)
```

- ▶ ¿cuáles son las desventajas de esto?

# 1. Pasos intermedios

```
diamonds <- ggplot2::diamonds
diamonds2 <- diamonds %>%
  dplyr::mutate(price_per_carat = price / carat)

pryr::object_size(diamonds)
3.46 MB
pryr::object_size(diamonds2)
3.89 MB
pryr::object_size(diamonds, diamonds2)
3.89 MB
```

# 1. Pasos intermedios

```
diamonds$caract[1] <-NA  
  
pryr::object_size(diamonds)  
3.46 MB  
pryr::object_size(diamonds2)  
3.89 MB  
pryr::object_size(diamonds, diamonds2)  
4.32 MB
```

## 2. Sobreescribir el original

- ▶ En lugar de crear objetos intermedios en casa pasó, sobreescribimos en el original:

```
Pimpon <- lava(Pimpon,  
que=Carita,conque=Agua,Jabon)  
Pimpom <-desenreda(Pimpom, Que=Pelo,  
conque=PeineMarfil)  
Pimpom <-NoLlora(Pimpon)  
Pimpom <-NiHaceAsi(Pimpom)
```

- ▶ ¿cuáles son las ventajas y desventajas de esto?

### 3. Composición de Funciones

- ▶ Otra aproximación es abandonar la asignación y solo encadenar el llamado de las funciones:

```
NiHaceAsi(  
    NoLlora(  
        desenreda(  
            lava(Pimpon,  
                que=Carita,conque=Agua,Jabon),  
                que=Pelo, conque=PeineMarfil)  
    )  
)
```

- ▶ ¿Cuáles son las desventajas de esto?

## 4. Uso de pipe

Pimpon %>%

lava(que=Carita,conque=Aqua,Jabon) %>%

desenreda(Que=Pelo, conque=PeineMarfil) %>%

NoLlora() %>%

NiHaceAsi()

- ▶ ¿Cuáles son las desventajas de esto?

# Cuando no usar Pipes

- ▶ Cuando tus pipes son muy largos (mas de 10 paso). En este caso crea objetos intermedio s con nombres útiles...para facilitar checar los errores y hacerlo mas fácil de entender.
- ▶ Cuando tienes múltiples entradas o salidas.
- ▶ Tu estas pensando en dirigirte hacia una grafica con dependencias complejas de la estructura. Recuerda.: los pipelines son fundamentalmente lineales



# Otras herramientas de magrittr

- ▶ Cuando trabajas con tuberías mas complejas, algunas veces el llamado a una función por sus efectos secundarios, como imprimir un objeto, o graficarlo o escribirlo a un archivo y ese tipo de acciones puede no regresar nada y terminar el pipe.
- Para darle la vuelta a este problema, puedes usar el “tee pie” `%T%` porque literalmente funciona con esa forma de tubería



# Tee pie

```
rnorm(100) %>%
  matrix(ncol = 2) %>%
  plot() %>%
  str()
# NULL

rnorm(100) %>%
  matrix(ncol = 2) %T>%
  plot() %>%
  str()
# num [1:50, 1:2] -0.877 -0.673 -1.642 -2.037 0.665 ...
```

# %\$%

- ▶ Si estas trabajando con funciones que no están basados en un data frame sino en vectores , por ejemplo, puedes encontrar %\$% útil:

```
mtcars %$%
  cor(disp, mpg)
# [1] -0.8475514
```

- ▶ Para asignar a la misma variable de entrada:

```
mtcars <- mtcars %>%
  transform(cyl = cyl * 2)

## o simplemente
mtcars %<>% transform(cyl = cyl * 2)
```