

Introducción a R

Lucía Babino

Introducción a R

Basado en la [clase de Juan Barriola y Sofía Perini](#)

Les dejo un [curso online \(mío\) introductorio de R](#) para quienes necesiten más detalles.

Modalidad

- Clases expositivas
- Ejercicios para el hogar

Material

- Presentaciones que uso durante las clases
- Apuntes de 2024
- Ejercicios para practicar (junto con los datos)
- Ejercicios resueltos

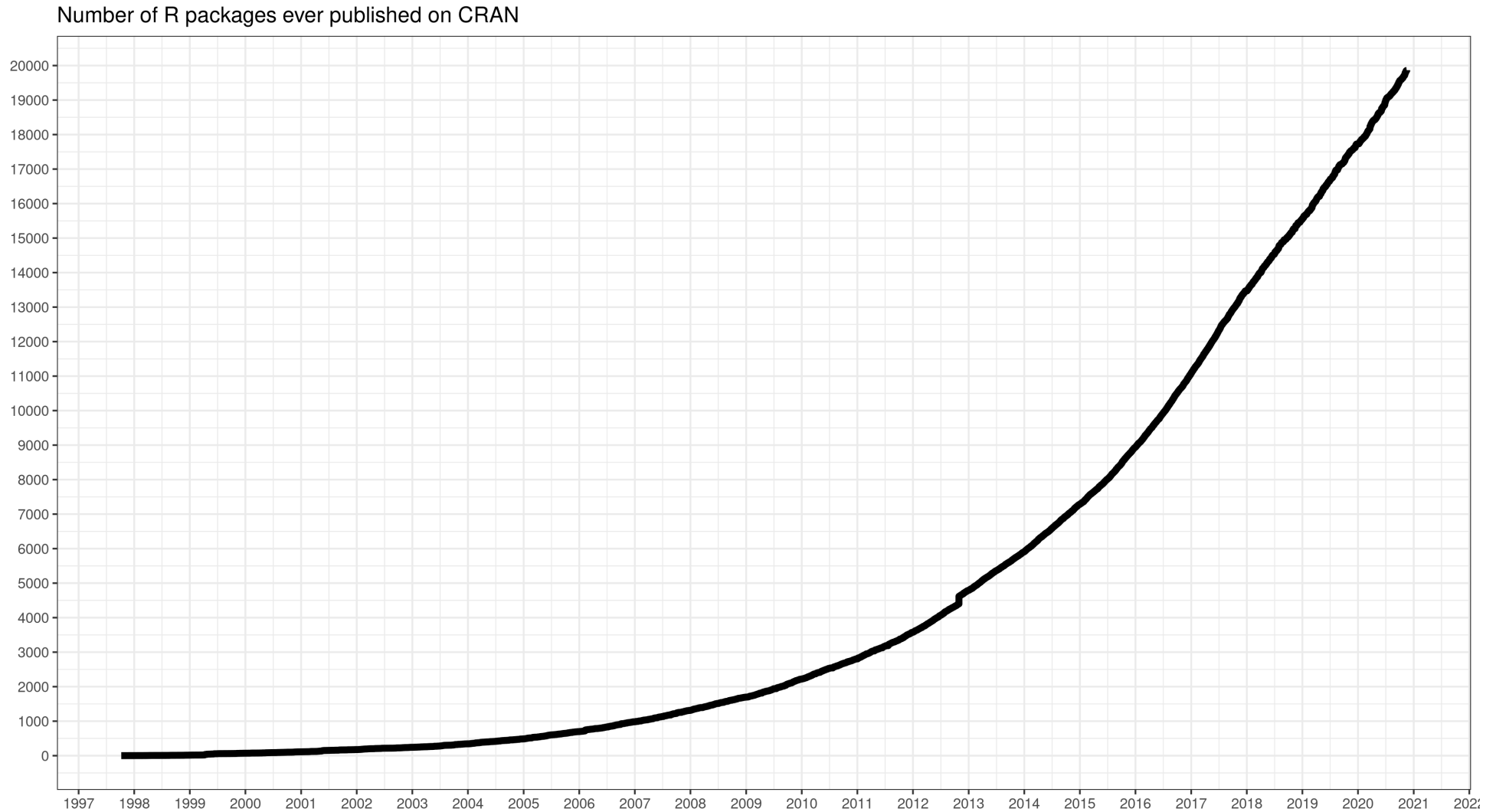
Temas de clases 1 y 2

- Clase 1:
 - Introducción a R y RStudio
 - Introducción a objetos
 - Manejo básico de bases de datos con R
 - Loops, estructuras condicionales y funciones
- Clase 2:
 - BLA

¿Qué es R?

- Lenguaje para el análisis estadístico de datos
- Software Libre
- Sintaxis incremental: paquete Base y paquetes que suben los distintos usuarios/as (universidades, empresas, etc)
- Es de código abierto
- Comunidad muy grande para realizar preguntas:
[RenBaires](#) y [R-Ladies Buenos Aires](#)

Ecosistema de R



Fuente: <https://gist.github.com/daroczig/3cf06d6db4be2bbe3368>

¿Qué es RStudio?



- Entorno cómodo para trabajar con R
- Producto de la empresa Posit
- Gratuito
- Descargable (junto con R) desde <https://posit.co/download/rstudio-desktop/>

Interfaz de RStudio

Scripts

```

226 ingreso_total = ingreso_laboral + ingreso_no_laboral,
227 CH04
228   case_when(CH04 == 1 ~ "Varon",
229             CH04 == 2 ~ "Mujer",
230             FALSE ~ "otro") ) %>%
231   group_by(DECINDR, CH04) %>%
232   summarise('ingreso laboral' = sum(ingreso_laboral*PONDII)/sum(ingreso_total*PONDII),
233             'ingreso no laboral' = sum(ingreso_no_laboral*PONDII)/sum(ingreso_total*PONDII)) %>%
234   gather(tipo_ingreso, monto, 3:4 )
235
236 ggplot(Grafico_2, aes(CH04, monto, fill = tipo_ingreso,
237                       label = sprintf("%1.1f%", 100*monto))) +
238   geom_col(position = "stack", alpha=0.6) +
239   geom_text(position = position_stack(vjust = 0.5), size=3) +
240   labs(x="", y="Porcentaje") +
241   theme_tufte() +
242   scale_fill_fivethirtyeight() +
243   scale_y_continuous(labels = percent) +
244   theme(legend.position = "bottom",
245         legend.title=element_blank()) +
246   facet_grid(.~DECINDR)
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Consola

```

> ggplot(Grafico_2, aes(CH04, monto, fill = tipo_ingreso,
+                       label = sprintf("%1.1f%", 100*monto))) +
+   geom_col(position = "stack", alpha=0.6) +
+   geom_text(position = position_stack(vjust = 0.5), size=3) +
+   labs(x="", y="Porcentaje") +
+   theme_tufte() +
+   scale_fill_fivethirtyeight() +
+   scale_y_continuous(labels = percent) +
+   theme(legend.position = "bottom",
+         legend.title=element_blank()) +
+   facet_grid(.~DECINDR)
+

```

Tablas

Environment	Presentation
Global Environment	
Hogar_t117	18477 obs. of 88 variables
Individual_t117	58675 obs. of 177 variables
Individual_t216	59811 obs. of 4 variables
Individual_t316	59550 obs. of 4 variables
Individual_t416	58154 obs. of 4 variables

Valores

values	
bases.dir	"G:/Bases/EPH/"
data.dir	"G:/Análisis/Otras publicaciones INDEC/..."
directorio	"G:/Análisis/Otras publicaciones INDEC/..."
resultados.dir	"G:/Análisis/Otras publicaciones INDEC/..."

Plots

Porcentaje

1 2 3 4 5 6 7 8 9 10

ingreso laboral ingreso no laboral

Pantalla Rstudio

R como calculadora

```
1 3 + 5
```

```
[1] 8
```

Para ejecutar (correr) un comando, presionar...

- En consola: **Enter**
- En script: **Ctrl + Enter** o botón **Run** (con el cursor ubicado en el renglón)

Objetos

Objetos

- Ingrediente fundamental de R
- “Caja” que guarda algo adentro, por ejemplo:
 - número
 - lista de números
 - texto
 - tabla de datos
 - gráfico
 - función, etc.

Objetos: ejemplo básico

```
1  a <- 3
2  b <- 5
3  suma <- a + b
4  suma
```

```
[1] 8
```

Environment (entorno de trabajo)

- lugar donde se guardan los objetos que creamos.
- está en el panel superior derecho de Rstudio.

Scripts

Tablas

Valores

Consola

```

> ggplot(Grafico_2, aes(CH04, monto, fill = tipo_ingreso,
+   label = sprintf("%1.1f%%", 100*monto))) +
+   geom_col(position = "stack", alpha=0.6) +
+   geom_text(position = position_stack(vjust = 0.5), size=3) +
+   labs(x="", y="porcentaje") +
+   theme_tufte() +
+   scale_fill_fiftythreeeight() +
+   scale_y_continuous(labels = percent) +
+   theme(legend.position = "bottom",
+   legend.title=element_blank()) +
+   facet_grid(.~DECINDR)
  
```

Eliminar objetos del entorno de trabajo

- Con `rm()` eliminamos un objeto del entorno de trabajo (environment)

```
1 rm(suma)
2 suma
```

Error in eval(expr, envir, enclos): objeto 'suma' no encontrado

- Con `rm(list = ls())` eliminamos **todos los objetos** del entorno de trabajo.

Es común utilizar esta instrucción al comenzar una nueva sesión, para asegurarnos de no trabajar con objetos que hayan quedado de sesiones

Operadores

Operadores aritméticos

```
1  5+6 # suma
```

```
[1] 11
```

```
1  6-8 # resta
```

```
[1] -2
```

```
1  6/2.5 # división
```

```
[1] 2.4
```

```
1  6*2.5 # multiplicación
```

```
[1] 15
```

```
1  3^2 # potenciación
```

```
[1] 9
```

Operadores lógicos

Más adelante veremos que son muy útiles, por ejemplo, para filtrar datos de una base de datos.

```
1 A <- 10
```

```
2 B <- 20
```

```
3 A > B
```

```
[1] FALSE
```

```
1 A >= B
```

```
[1] FALSE
```

Operadores lógicos

```
1 A < B
```

```
[1] TRUE
```

¿A es igual a B?

```
1 A == B
```

```
[1] FALSE
```

¿A es distinto a B?

```
1 A != B
```

```
[1] TRUE
```

```
1 C <- A != B
```

```
2 C
```

```
[1] TRUE
```

Funciones

- Scripts “enlatados”
- Pueden estar
 - en paquete Base (como veremos ahora)
 - en paquetes que debemos instalar (clase que viene)
 - programadas por el usuario (en un rato)
- Toman **inputs** (*argumentos*) como entrada y devuelven un **output** (*resultado*)

Algunas funciones del paquete Base

```
1 1:5
```

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

```
1 sum(1:5)
```

```
[1] 15
```

```
1 mean(1:5)
```

```
[1] 3
```

```
1 paste("Hola", "mundo", sep = " ")
```

```
[1] "Hola mundo"
```

```
1 paste("Hola", "mundo", sep = "-")
```

```
[1] "Hola-mundo"
```

Caracteres especiales en R

Mayúsculas y minúsculas

R es sensible a mayúsculas y minúsculas.

```
1 a <- 3
```

```
2 A <- 5
```

```
1 a
```

```
[1] 3
```

```
1 A
```

```
[1] 5
```

a y **A** son objetos distintos para **R**.

Espacios en blanco

No son considerados por R, se utilizan para emprolijar el código.

```
1  A<-1
```

Es lo mismo que...

```
1  A <- 1
```

Pero lo segundo es más prolijo.

Espacios en blanco

```
1 paste("Hola", "mundo", sep=" ")
```

```
[1] "Hola mundo"
```

Es lo mismo que...

```
1 paste("Hola", "mundo", sep = " ")
```

```
[1] "Hola mundo"
```

Pero lo segundo es más prolijo.

Mas recomendaciones de estilo en

<https://style.tidyverse.org/>

Numeral

Se utiliza para hacer comentarios. Todo lo que se escribe después del # no es interpretado por R.

```
1 mean(1:5) # calcula el promedio
```

```
[1] 3
```

```
1 # calcula el promedio
```

```
2 mean(1:5)
```

```
[1] 3
```

->

Tipos de objetos

En R, hay distintos tipos de objetos (estructuras de datos) que permiten organizar la información de distintas maneras. Los más básicos son:

1. Valores
2. Vectores
3. Data Frames
4. Listas

1. Valores:

Están compuestos por un único elemento.

Ejemplos:

```
1 A <- 1
```

```
1 B <- "hola"
```

Clases de objetos

Los valores (y vectores) pueden ser a su vez de distintas clases

1. Numeric

```
1  A <- 1  
2  class(A)
```

```
[1] "numeric"
```

Clases de objetos

Los valores (y vectores) pueden ser a su vez de distintas clases

2. Character

```
1 B <- "Hola mundo"  
2 B
```

```
[1] "Hola mundo"
```

```
1 class(B)
```

```
[1] "character"
```


Clases de objetos

Los valores (y vectores) pueden ser a su vez de distintas clases

3. Logical

```
1 C <- 3 < 5  
2 C
```

```
[1] TRUE
```

```
1 class(C)
```

```
[1] "logical"
```

Tipos de objetos

1. Valores
2. Vectores
3. Data Frames
4. Listas

Vectores

- Un vector es una *concatenación* de valores de la misma clase.
- Se crean con el comando `c()`.

```
1 x <- c(1, 3, 4)
```

```
2 x
```

```
[1] 1 3 4
```

```
1 class(x)
```

Vector de caracteres

```
1 nombres <- c("Carlos", "Federico", "Pedro")  
2 nombres
```

```
[1] "Carlos"      "Federico"    "Pedro"
```

```
1 class(nombres)
```

```
[1] "character"
```

Vector lógico

```
1 menor_a_5 <- c(3 < 5, 4 < 5, 6 < 5)  
2 menor_a_5
```

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

```
1 class(menor_a_5)
```

```
[1] "logical"
```

Vectores (más ejemplos)

```
1 x <- c(1, "hola")  
2 x
```

```
[1] "1"      "hola"
```

```
1 class(x)
```

```
[1] "character"
```

Vectores (más ejemplos)

```
1 y <- c(1, TRUE)
```

```
2 y
```

```
[1] 1 1
```

```
1 class(y)
```

```
[1] "numeric"
```

Vectores (más ejemplos)

```
1 z <- c(TRUE, "hola")  
2 z
```

```
[1] "TRUE" "hola"
```

```
1 class(z)
```

```
[1] "character"
```


Vectores (más ejemplos)

```
1 w <- c(1, TRUE, "hola")  
2 w
```

```
[1] "1"      "TRUE"   "hola"
```

```
1 class(w)
```

```
[1] "character"
```

Suma de vectores

Podemos sumar dos vectores numéricos.

```
1 x <- c(1, 3, 4)
```

```
2 y <- c(2, 5, 3)
```

```
3 x + y
```

```
[1] 3 8 7
```

Suma de vectores

Podemos sumar el mismo valor a todos los elementos de un vector

```
1 x <- c(1, 3, 4)
```

```
2 x + 2
```

```
[1] 3 5 6
```

¿Cómo ejecuta R internamente esta operación?

Suma de vectores

```
1 x <- c(1, 3, 4)
```

```
2 x + 2
```

```
[1] 3 5 6
```

es equivalente a

```
1 x <- c(1, 3, 4)
```

```
2 x + c(2, 2, 2)
```

```
[1] 3 5 6
```

Reciclado de vectores

Para sumar vectores de distinta longitud R *recicla* el vector más corto tantas veces como sea necesario para igualar la longitud del más largo.

Reciclado de vectores (ejemplo)

```
1 # sumamos dos vectores de distinta longi
2 x <- 1:10
3 y <- 1:2
```

```
1 x
```

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

```
1 y
```

```
[1] 1 2
```

```
1 x + y
```

```
[1] 2 4 4 6 6 8 8 10 10 12
```

Elementos de un vector

Veamos cómo acceder a un elemento de un vector.

```
1 nombres <- c("Carlos", "Federico", "Pedro")
```

Accedemos al segundo elemento

```
1 nombres[2]
```

```
[1] "Federico"
```

Podemos almacenarlos en otro objeto

```
1 elemento2 <- nombres[2]  
2 elemento2
```

```
[1] "Federico"
```

Elementos de un vector

Podemos cambiar el contenido de un elemento de un vector

```
1 nombres <- c("Carlos", "Federico", "Pedro")  
2 nombres
```

```
[1] "Carlos" "Federico" "Pedro"
```

```
1 nombres[2] <- "Pablo"  
2 nombres
```

```
[1] "Carlos" "Pablo" "Pedro"
```


Tipos de objetos

1. Valores
2. Vectores
3. Data Frames
4. Listas

Data Frames

Un **data frame** es una tabla de datos en R donde cada *columna* representa una *variable* y cada *fila* una observación.

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1
8	May-20	Público	264.0

Data frames

Los **data frames** son el tipo de objeto que usamos en R para importar y trabajar con bases de datos externas.

Leer datos en R

Funciones comunes para importar datos:

- `read.table()`
 - Paquete Base R
 - Para archivos de texto en general (`.txt`, `.csv`, `.tsv`)
- `read.csv()`
 - Paquete Base R
 - Para archivos separados por comas (`.csv`)
- `read_excel()`
 - Paquete `readxl`
 - Para archivos de Excel (`.xls`, `.xlsx`)

Importar datos con `read.table()`

Tenemos un dataset con algunos datos del índice de salarios del INDEC en un `salarios.txt` en la carpeta “./datos”.

```
"FECHA" "GRUPO" "INDICE"  
"Mar-20" "Privado_Registrado" 286.4  
"Mar-20" "Público" 262.1  
"Mar-20" "Privado_No_Registrado" 248.5  
"Abr-20" "Privado_Registrado" 285.7  
"Abr-20" "Público" 263.5  
"Abr-20" "Privado_No_Registrado" 250.2  
"May-20" "Privado_Registrado" 285.1  
"May-20" "Público" 264.9  
"May-20" "Privado_No_Registrado" NA
```

Importar datos con `read.table()`

```
1 Datos <- read.table("./datos/salarios.tx")
```

```
1 head(Datos)
```

	V1	V2	V3
1	FECHA	GRUPO	INDICE
2	Mar-20	Privado_Registrado	286.4
3	Mar-20	Público	262.1
4	Mar-20	Privado_No_Registrado	248.5
5	Abr-20	Privado_Registrado	285.7
6	Abr-20	Público	263.5

```
1 names(Datos)
```

```
[1] "V1" "V2" "V3"
```

Importar datos con `read.table()`

```
1 head(Datos)
```

	V1	V2	V3
1	FECHA	GRUPO	INDICE
2	Mar-20	Privado_Registrado	286.4
3	Mar-20	Público	262.1
4	Mar-20	Privado_No_Registrado	248.5
5	Abr-20	Privado_Registrado	285.7
6	Abr-20	Público	263.5

La primera fila del .txt tiene los nombres de las variables, pero **R** la interpretó como una observación.

Importar datos con `read.table()`

```
1 Datos <- read.table("./datos/salarios.tx  
2                      header = TRUE)
```

```
1 head(Datos)
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2

```
1 names(Datos)
```

Elementos de un data frame

Para acceder a determinado elemento de un data frame podemos usar los corchetes `[]` como en los vectores pero especificando dos valores (*fila* y *columna*).

```
1 Datos[3, 1]
```

```
[1] "Mar-20"
```

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7

5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285 1

Fila de un data frame

Si queremos todos los elementos de una fila (ej. la fila 3)

```
1 Datos[3, ]
```

	FECHA	GRUPO	INDICE
3	Mar-20	Privado_No_Registrado	248.5

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2

7 May-20

Privado_Registrado

285.1

0 00 00

0000

0000

Columna de un data frame

Si queremos todos los elementos de una columna (ej. la columna 1)

```
1 Datos[, 1]
```

```
[1] "Mar-20" "Mar-20" "Mar-20" "Abr-20"
"Abr-20" "Abr-20" "May-20" "May-20"
[9] "May-20"
```

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7

5	Abr-20		Público	263.5
6	Abr-20	Privado_No_Registrado		250.2
7	May-20	Privado_Registrado		285.1
8	May-20		Público	264.0

Columna de un data frame

Otra forma de acceder a la primera columna

```
1 Datos$FECHA
```

```
[1] "Mar-20" "Mar-20" "Mar-20" "Abr-20"
"Abr-20" "Abr-20" "May-20" "May-20"
[9] "May-20"
```

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5

6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1
8	Jun-20	Privado_Registrado	264.2

Elemento de un data frame

Otra forma de acceder a un elemento de un data frame (ej. observación 3 de la variable FECHA)

```
1 Datos$FECHA[3]
```

```
[1] "Mar-20"
```

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2

7 May-20	Privado_Registrado	285.1
8 May-20	Público	264.0

Filtros en data frames

Supongamos que queremos quedarnos con todas las observaciones de abril de 2020.

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1
8	May-20	Público	264.0

Filtros en data frames

```
1 Datos[4:6, ]
```

	FECHA	GRUPO	INDICE
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2

Filtros con condiciones lógicas

```
1 Datos[Datos$FECHA == "Abr-20", ]
```

	FECHA	GRUPO	INDICE
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2

Veamos por qué funciona

Filtros con condiciones lógicas

Veamos qué realiza este comando

```
1 Datos$FECHA == "Abr-20"
```

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

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2

7 May-20

Privado_Registrado

285.1

0 10 20

0 10 20

0 10 20

Datos faltantes

```
1 tail(Datos)
```

	FECHA	GRUPO	INDICE
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1
8	May-20	Público	264.9
9	May-20	Privado_No_Registrado	NA

“NA” (Not Available) es la forma en que R representa los datos faltantes

Eliminar datos faltantes

Queremos quedarnos con un data frame que tenga únicamente los registros sin datos faltantes en **INDICE**.

Para ello debemos usar la función **is.na()**

```
1 is.na(Datos$INDICE)
```

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

Para quedarnos con los datos distintos de NA, necesitamos un vector que tenga **TRUE** si el valor de **INDICE** **No** es NA.

Para eso, usamos el operador **!** al principio para negar **is.na()** y obtener **TRUE** cuando el valor **No** es faltante.

```
1 !is.na(Datos$INDICE)
```

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

Eliminar datos faltantes

```
1 Datos[!is.na(Datos$INDICE), ]
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1
8	May-20	Público	264.0

Manejo de datos faltantes

Podemos aplicar el filtro sobre una columna

```
1 Datos$INDICE[Datos$FECHA == "Abr-20"]
```

```
[1] 285.7 263.5 250.2
```

```
1 Datos
```

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1

Filtros en data frames

Calculamos la media de los índices de abril de 2020

```
1 mean(Datos$INDICE[Datos$FECHA == "Abr-2020"])  
[1] 266.4667
```

o la de los índices de mayo de 2020

```
1 mean(Datos$INDICE[Datos$FECHA == "May-2020"])  
[1] NA
```

Devuelve **NA** porque uno de los valores del vector que está promediando es **NA**. Si queremos calcular el promedio de los valores no faltantes...

```
1 mean(Datos$INDICE[Datos$FECHA == "May-2020"],  
2       na.rm = TRUE)
```

Tipos de objetos

1. Valores
2. Vectores
3. Data Frames
4. Listas

Listas

1. Valores
2. Vectores: concatenación de valores de la misma clase
3. Data Frames: concatenación de vectores
4. **Listas**: concatenación de objetos de cualquier tipo

Listas (ejemplo)

```
1 mi_lista <- list(nombre = "Lucía",
2                   numeros = 1:10,
3                   indices_df = Datos)
4 mi_lista
```

\$nombre

[1] "Lucía"

\$numeros

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

\$indices_df

FECHA

GRUPO INDICE

1 Mar 20 Desierto Desertado 206 1

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista[2]
```

```
$numeros
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista[2])
```

```
[1] "list"
```

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista[[2]]
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista[[2]])
```

```
[1] "integer"
```

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista$numeros
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista$numeros)
```

```
[1] "integer"
```

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista[["numeros"]]
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista[["numeros"]])
```

```
[1] "integer"
```

Loops, estructuras condicionales y funciones

¿Qué es un loop?

Loop (o bucle): estructura que nos permite **repetir una misma acción varias veces**, variando el valor de una variable en cada repetición.

Ejemplo: imprimimos el cuadrado de los números del 1 al 5.

```
1  for(i in 1:5) {  
2      print(i^2)  
3  }
```

[1] 1

[1] 4

[1] 9

[1] 16

[1] 25

Otro ejemplo de loop y buenas prácticas

```
1 for(valor in 1:5) {  
2     print(valor^2)  
3 }
```

[1] 1

[1] 4

[1] 9

[1] 16

[1] 25

Podemos usar el nombre que querramos para la variable del `for`.

Estructuras condicionales

- permiten ejecutar diferentes instrucciones según si se cumple o no una condición lógica.
- Comandos principales: `if`, `else` y `else if`.

Comando `if`

Sintaxis básica:

```
1  if (condición) {  
2      # código a ejecutar si la cond. es ver  
3  }
```

Ejemplo:

```
1  x <- 5  
2  
3  if (x > 0) {  
4      print("x es positivo")  
5  }
```

```
[1] "x es positivo"
```

Comando **if** (condición no cumplida)

```
1  x <- -3
2
3  if (x > 0) {
4      print("x es positivo")
5  }
```

Comando `else`

```
1  x <- 5
2
3  if (x > 0) {
4    print("x es positivo")
5  } else {
6    print("x no es positivo")
7  }
```

```
[1] "x es positivo"
```

Comandos `if` y `else`

```
1  x <- 0
2
3  if (x > 0) {
4    print("x es positivo")
5  } else {
6    print("x no es positivo")
7  }
```

```
[1] "x no es positivo"
```

Comando `else if`

```
1  x <- 0
2
3  if (x > 0) {
4    print("x es positivo")
5  } else if (x == 0) {
6    print("x es cero")
7  } else {
8    print("x es negativo")
9  }
```

```
[1] "x es cero"
```

Comando `else if`

```
1  x <- -3
2
3  if (x > 0) {
4    print("x es positivo")
5  } else if (x == 0) {
6    print("x es cero")
7  } else {
8    print("x es negativo")
9  }
```

```
[1] "x es negativo"
```

Funciones

- Función: script “enlatado”
 - Inputs: argumentos
 - Output: resultado
- Pueden estar
 - en paquete Base (como ya vimos)
 - en paquetes que debemos instalar (clase que viene)
 - programadas por el usuario (en un rato)

Funciones en paquete Base (repaso)

```
1 x <- c(1, 3, 2)
2 sum(x)
```

```
[1] 6
```

```
1 sum(c(1, 3, 2))
```

```
[1] 6
```

```
1 x <- c(1, 3, 2)
2 mean(x)
```

```
[1] 2
```

```
1 paste("Hola", "mundo", sep = " ")
```

```
[1] "Hola mundo"
```

Help

Podemos ver la **ayuda** de una función con

?nombre_función

o en la pestaña “Help” del panel inferior derecho.

The screenshot shows the RStudio interface with the following components and annotations:

- Scripts:** The main editor window displays R code for data analysis and visualization.
- Consola:** The console window at the bottom shows the execution of the R code.
- Tablas:** The Environment pane on the right lists the objects in the global environment, including 'Hogar_t117', 'Individual_t117', 'Individual_t216', 'Individual_t316', and 'Individual_t416'.
- Valores:** The Values pane on the right shows the values of the selected object, including 'bases.dir', 'data.dir', 'directorio', and 'resultados.dir'.
- Help:** The 'Help' tab is selected in the bottom-right pane, showing a bar chart with a legend for 'ingreso laboral' and 'ingreso no laboral'.

Pantalla Rstudio

Crear nuestra propia función

Podemos definir nuestras propias funciones con la función `function()`:

```
1  cuadrado <- function(x) {  
2    return(x^2)  
3  }  
4  
5  cuadrado(5)
```

```
[1] 25
```

`return()` no es obligatorio, pero hace el código más claro.

Función sin return

```
1  cuadrado <- function(x) {  
2    x^2  
3  }  
4  
5  cuadrado(5)
```

```
[1] 25
```

Otra función sin return

```
1 polinomio <- function(x) {  
2     y <- x + 2  
3     y^2  
4 }  
5  
6 polinomio(5)
```

```
[1] 49
```

Otra función sin return

```
1 polinomio <- function(x) {  
2     y <- x + 2  
3     return(y^2)  
4 }  
5  
6 polinomio(5)
```

```
[1] 49
```

`return()` hace el código más claro especialmente cuando hay más de una línea de código en la función.

Funciones en Environment

Las funciones que creamos nosotros permanecen en el entorno de trabajo de R (Environment) temporariamente.

Para poder usar una funcion primero debemos ejecutarla, para que quede guardada en el entorno de trabajo y R la pueda usar.

Funciones con múltiples argumentos

Podemos definir funciones que tomen más de un argumento.

```
1 potencia <- function(base, exponente) {  
2   return(base ^ exponente)  
3 }
```

```
4  
5 potencia(3, 4)
```

```
[1] 81
```


Funciones con múltiples argumentos

Podemos aclarar explícitamente qué valor toma cada argumento

```
1 potencia <- function(base, exponente) {  
2   return(base ^ exponente)  
3 }  
4  
5 potencia(base = 3, exponente = 4)
```

```
[1] 81
```

Funciones con múltiples argumentos

En ese caso, podemos cambiar el orden de los argumentos

```
1 potencia <- function(base, exponente) {  
2   return(base ^ exponente)  
3 }
```

```
4
```

```
5 potencia(exponente = 4, base = 3)
```

```
[1] 81
```

Funciones con múltiples argumentos

pero, ¿qué pasa si cambiamos el orden de los argumentos sin nombrarlos?

```
1 potencia <- function(base, exponente) {  
2   return(base ^ exponente)  
3 }  
4  
5 potencia(4, 3)
```

```
[1] 64
```

Valores por defecto

También podemos establecer **valores por defecto** para que no siempre sea necesario especificarlos.

```
1 potencia <- function(base, exponente = 2)
2   return(base ^ exponente)
3 }
4
5 potencia(3, 4)      # Usa exponente = 4
```

```
[1] 81
```

Valores por defecto

```
1 potencia <- function(base, exponente = 2
2   return(base ^ exponente)
3 }
4
5 potencia(3) # Usa exponente = 2 por defe
```

```
[1] 9
```

