Introduction to R

Code

Bibek Sapkota

Tibbles

Task 1:Loading the tidyverse package.

Hide

```
library(tidyverse)
```

Task 2:Converting the iris dataset to a tibble.

Hide

```
as_tibble(iris)
```

Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fctr>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

1-10 of 150 rows Previous **1** 2 3 4 5 6 … 15 Next

Task 3: Creating a tibble with columns “x,” “y,” and “z,” where “x” ranges from 1 to 5, “y” is 1 for all rows, and “z” is calculated as the square of “x” plus “y” for each row.

Hide

```
tibble(
  x = 1:5,
  y = 1,
  z = x ^ 2 + y
)
```

x <int>	y <dbl>	z <dbl>
1	1	2
2	1	5
3	1	10
4	1	17
5	1	26

5 rows

Task 4:Creating a tibble with columns named ":)" (representing "smile"), " " (representing "space"), and "2000" (representing "number").

Hide

```
tb <- tibble(
  `:)` = "smile",
  ` ` = "space",
  `2000` = "number"
)
tb
```

:) <chr>	<chr>	2000 <chr>
smile	space	number

1 row

Task 5:Creating a tibble with columns "x," "y," and "z," containing the values "a," 2, 3.6 and "b," 1, 8.5 respectively.

Hide

```
tribble(
  ~x, ~y, ~z,

  "a", 2, 3.6,
  "b", 1, 8.5
)
```

x <chr>	y <dbl>	z <dbl>
a	2	3.6
b	1	8.5

2 rows

Tibbles vs. data.frame

Task-1:Creating a tibble with columns “a,” “b,” “c,” “d,” and “e,” containing 1000 randomly generated values for each column, representing dates, numbers, and letters.

Hide

```
tibble(
  a = lubridate::now() + runif(1e3) * 86400,
  b = lubridate::today() + runif(1e3) * 30,
  c = 1:1e3,
  d = runif(1e3),
  e = sample(letters, 1e3, replace = TRUE)
)
```

a <S3: POSIXct>	b <date>	c <int>	d <dbl>	e <chr>
2024-05-05 06:49:22	2024-05-16	1	0.175926707	t
2024-05-05 00:01:55	2024-05-16	2	0.267888714	b
2024-05-04 23:23:49	2024-05-12	3	0.785106206	i
2024-05-05 17:26:29	2024-05-17	4	0.362381005	q
2024-05-04 23:34:31	2024-05-30	5	0.294068429	a
2024-05-05 10:53:15	2024-05-22	6	0.723301471	h
2024-05-05 17:18:41	2024-05-17	7	0.502085239	e
2024-05-05 09:11:30	2024-05-21	8	0.319334059	p
2024-05-05 18:13:51	2024-05-15	9	0.290362461	e
2024-05-04 21:14:18	2024-05-12	10	0.433118793	q

1-10 of 1,000 rows Previous **1** 2 3 4 5 6 … 100 Next

Task 2: Tnstalling the package

Hide

```
package_to_install <- c("nycflights13")

for (package_name in package_to_install) {
  if (!requireNamespace(package_name, quietly = TRUE)) {
    install.packages(package_name)
  }
}
library(nycflights13)
```

Task 3: Printing the first 10 rows of the nycflights13::flights dataset with unlimited width.

Hide

```
nycflights13::flights %>%
  print(n = 10, width = Inf)
```

y… <int>	mo… <int>	d.. <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	c <
2013	1	1	517	515	2	830	819	11	U
2013	1	1	533	529	4	850	830	20	U
2013	1	1	542	540	2	923	850	33	A
2013	1	1	544	545	-1	1004	1022	-18	B
2013	1	1	554	600	-6	812	837	-25	D
2013	1	1	554	558	-4	740	728	12	U
2013	1	1	555	600	-5	913	854	19	B
2013	1	1	557	600	-3	709	723	-14	E
2013	1	1	557	600	-3	838	846	-8	B
2013	1	1	558	600	-2	753	745	8	A

1-10 of 336,776 rows | 1-10 of 19 columns Previous **1** 2 3 4 5 6 … 100 Next

Task 4: Viewing the nycflights13::flights dataset in a separate window for interactive exploration.

Hide

```
nycflights13::flights %>%
  View()
```

Subsetting

Task 1: Creating a tibble named “df” with columns “x” and “y,” then accessing the “x” column using different methods:

Hide

```
df <- tibble(
  x = runif(5),#function that generates random numbers from a uniform distribution
  y = rnorm(5) # function that generates random numbers from a normal (Gaussian) distribution
)

df$x
```

```
[1] 0.4134781 0.3841133 0.5761670 0.6047906 0.8490257
```

Hide

```
df[["x"]]
```

```
[1] 0.4134781 0.3841133 0.5761670 0.6047906 0.8490257
```

Hide

```
df[[1]]
```

```
[1] 0.4134781 0.3841133 0.5761670 0.6047906 0.8490257
```

Hide

```
df %>% .$x
```

```
[1] 0.4134781 0.3841133 0.5761670 0.6047906 0.8490257
```

Interacting with older code

Task-1: Determining the class of the object “tb” after converting it to a data frame.

Hide

```
class(as.data.frame(tb))
```

```
[1] "data.frame"
```

Exercises

Task-1: How can you tell if an object is a tibble? (Hint: try printing mtcars, which is a regular data frame).

Hide

```
mtcars
```

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>	am <dbl>
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0

1-10 of 32 rows | 1-10 of 11 columns Previous **1** 2 3 4 Next

Task-2

Hide

```
# In a data.frame, extracting a non-existent column returns NULL,
# whereas in a tibble, it raises an error, providing immediate feedback.
# Other operations, such as extracting existing columns and subsets of columns,
# behave similarly across both data frames and tibbles.
# The default behavior of data.frames may lead to frustration
# due to the lack of error feedback for non-existent columns,
# potentially causing unnoticed mistakes and difficulty in debugging.
# In contrast, tibbles offer more robust behavior, enhancing data integrity
# and debugging efficiency.

df <- data.frame(abc = 1, xyz = "a")

# Extracting non-existent column in a data.frame
df$x  # Returns NULL
```

```
[1] "a"
```

Hide

```
# Extracting existing column in a data.frame
df[, "xyz"]  # Returns a data frame with one column containing the values of the "xyz" column
```

```
[1] "a"
```

Hide

```
# Extracting multiple columns in a data.frame
df[, c("abc", "xyz")]  # Returns a data frame containing only the specified columns
```

abc	xyz
<dbl>	<chr>
1	a

1 row

Hide

```
NA
```

Task-3:If you have the name of a variable stored in an object, e.g. var <- “mpg”, how can you extract the reference variable from a tibble?

No pacakages

Hide

```
# heights <- read_csv("data/heights.csv")
```

Task 1: listing several tables: table1, table2, table3, table4a, and table4b.

Hide

```
table1
```

country <chr>	year <dbl>	cases <dbl>	population <dbl>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Hide

```
table2
```

country <chr>	year <dbl>	type <chr>	count <dbl>
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272

1-10 of 12 rows Previous **1** 2 Next

Hide

```
table3
```

country <chr>	year <dbl>	rate <chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362

country <chr>	year <dbl>	rate <chr>
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

6 rows

Hide

```
table4a
```

country <chr>	1999 <dbl>	2000 <dbl>
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

3 rows

Hide

```
table4b
```

country <chr>	1999 <dbl>	2000 <dbl>
Afghanistan	19987071	20595360
Brazil	172006362	174504898
China	1272915272	1280428583

3 rows

Task 2: Calculating the rate by dividing the number of cases by the population and then multiplying by 10,000 for table1.

Hide

```
table1 %>%
  mutate(rate = cases / population * 10000)
```

country <chr>	year <dbl>	cases <dbl>	population <dbl>	rate <dbl>
Afghanistan	1999	745	19987071	0.372741
Afghanistan	2000	2666	20595360	1.294466
Brazil	1999	37737	172006362	2.193930
Brazil	2000	80488	174504898	4.612363

country <chr>	year <dbl>	cases <dbl>	population <dbl>	rate <dbl>
China	1999	212258	1272915272	1.667495
China	2000	213766	1280428583	1.669488

6 rows

Task 3: Counting the occurrences of each year in table1, using the ‘cases’ column as the weight.

Hide

```
table1 %>%
  count(year, wt = cases)
```

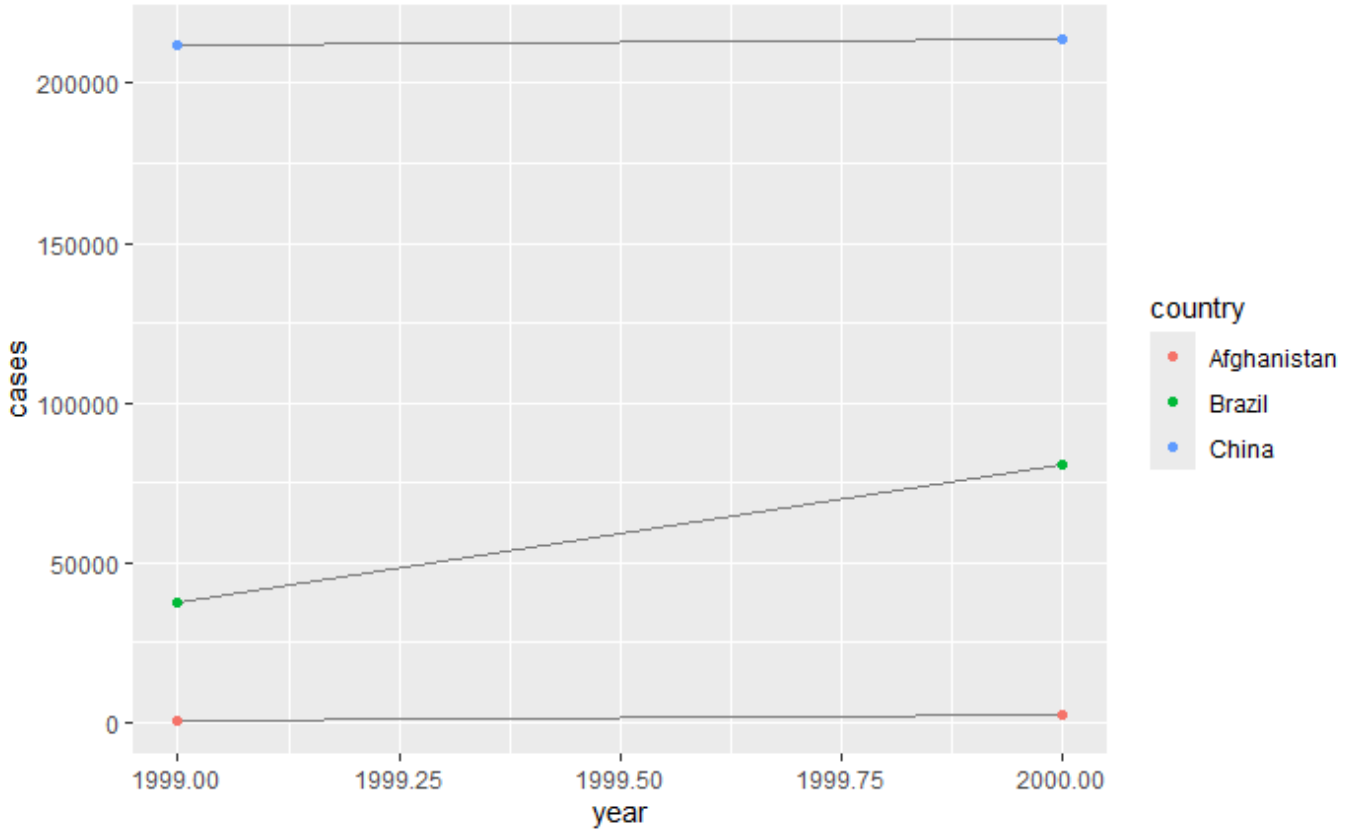
year <dbl>	n <dbl>
1999	250740
2000	296920

2 rows

Task 4: Creating a ggplot using table1, plotting ‘year’ against ‘cases’ with lines grouped by ‘country’ and colored in grey50, along with points colored by ‘country’.

Hide

```
library(ggplot2)
ggplot(table1, aes(year, cases)) +
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country))
```



Pivoting

Longer

Task-1: referring to ‘table4a’

Hide

```
table4a
```

country <chr>	1999 <dbl>	2000 <dbl>
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

3 rows

Task-2: Reshaping table4a using pivot_longer for columns ‘1999’ and ‘2000’ into ‘year’ and ‘cases’.

Hide

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

country <chr>	year <chr>	cases <dbl>
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

6 rows

Task-3: Reshaping table4b with pivot_longer for columns ‘1999’ and ‘2000’ into ‘year’ and ‘population’.

Hide

```
table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")  #function tra
nsforms wide data into long format by stacking multiple columns into two: one for variable na
mes and one for their corresponding values
```

country <chr>	year <chr>	population <dbl>
Afghanistan	1999	19987071

country <chr>	year <chr>	population <dbl>
Afghanistan	2000	20595360
Brazil	1999	172006362
Brazil	2000	174504898
China	1999	1272915272
China	2000	1280428583

6 rows

Task-4: creating tidy datasets tidy4a and tidy4b by using pivot_longer on table4a and table4b to reshape them. Then, performing a left join on tidy4a and tidy4b.

Hide

```
tidy4a <- table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
tidy4b <- table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")
left_join(tidy4a, tidy4b)
```

```
Joining with `by = join_by(country, year)`
```

country <chr>	year <chr>	cases <dbl>	population <dbl>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Wider

Task-1:Displaying table 2

Hide

```
table2
```

country <chr>	year <dbl>	type <chr>	count <dbl>
Afghanistan	1999	cases	745

country <chr>	year <dbl>	type <chr>	count <dbl>
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272

1-10 of 12 rows Previous **1** 2 Next

Task-2: using the pivot_wider function on table2 to transform it from long to wide format, with ‘type’ becoming the new column names and ‘count’ being the corresponding values.

Hide

```
table2 %>%
    pivot_wider(names_from = type, values_from = count)
```

country <chr>	year <dbl>	cases <dbl>	population <dbl>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Separating and uniting

Separate

Task-1:displaying table3

Hide

```
table3
```

country <chr>	year <dbl>	rate <chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

6 rows

Task-2: Using the separate function on table3 splits the ‘rate’ column into two separate columns named ‘cases’ and ‘population’.

Hide

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

country <chr>	year <dbl>	cases <chr>	population <chr>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Task-3:Using the separate function on table3 splits the ‘rate’ column into two separate columns named ‘cases’ and ‘population’, using the ‘/’ character as the separator.

Hide

```
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
```

country <chr>	year <dbl>	cases <chr>	population <chr>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898

country <chr>	year <dbl>	cases <chr>	population <chr>
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Task-4:Using the separate function on table3 splits the ‘rate’ column into two separate columns named ‘cases’ and ‘population’, converting the resulting columns to their appropriate data types.

Hide

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

country <chr>	year <dbl>	cases <int>	population <int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Task-5: Applying the separate function to table3, the ‘year’ column is divided into two separate columns labeled ‘century’ and ‘year’, with the separator defined as the second character.

Hide

```
table3 %>%
  separate(year, into = c("century", "year"), sep = 2)
```

country <chr>	century <chr>	year <chr>	rate <chr>
Afghanistan	19	99	745/19987071
Afghanistan	20	00	2666/20595360
Brazil	19	99	37737/172006362
Brazil	20	00	80488/174504898
China	19	99	212258/1272915272
China	20	00	213766/1280428583

6 rows

Unite

Task-1: The unite function is applied to table5 to merge the ‘century’ and ‘year’ columns into a single column named ‘new’.

Hide

```
table5 %>%
  unite(new, century, year)
```

country <chr>	new <chr>	rate <chr>
Afghanistan	19_99	745/19987071
Afghanistan	20_00	2666/20595360
Brazil	19_99	37737/172006362
Brazil	20_00	80488/174504898
China	19_99	212258/1272915272
China	20_00	213766/1280428583

6 rows

Task-2: unite function is applied to table5 to merge the ‘century’ and ‘year’ columns into a single column named ‘new’, with no separator between them.

Hide

```
table5 %>%
  unite(new, century, year, sep = "")
```

country <chr>	new <chr>	rate <chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

6 rows

Missing values

Task-1: Create a tibble named “stocks” with columns “year”, “qtr” (quarter), and “return”, having data for 2015 and 2016, with quarterly returns specified and some missing entries as NA.

Hide

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

Task-2:Pivoting the “stocks” tibble to widen the data, extracting columns from the “year” variable and values from the “return” variable.

Hide

```
stocks %>%
  pivot_wider(names_from = year, values_from = return)
```

qtr <dbl>	2015 <dbl>	2016 <dbl>
1	1.88	*NA*
2	0.59	0.92
3	0.35	0.17
4	*NA*	2.66

4 rows

Task-3: pivot the data to a wide format with columns for each year’s returns, then reshape it back to a long format, keeping only the non-missing values in the “return” column.

Hide

```
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(
    cols = c(`2015`, `2016`),
    names_to = "year",
    values_to = "return",
    values_drop_na = TRUE
  )
```

qtr <dbl>	year <chr>	return <dbl>
1	2015	1.88
2	2015	0.59
2	2016	0.92
3	2015	0.35
3	2016	0.17
4	2016	2.66

6 rows

Task-4:Filling missing combinations of “year” and “qtr” in the “stocks” dataset.

Hide

```
stocks %>%
  complete(year, qtr)
```

year <dbl>	qtr <dbl>	return <dbl>
2015	1	1.88
2015	2	0.59
2015	3	0.35
2015	4	*NA*
2016	1	*NA*
2016	2	0.92
2016	3	0.17
2016	4	2.66

8 rows

Task-5:Creating a tibble named “treatment” containing information about individuals, their treatment groups, and their responses, with some missing values for the “person” column.

Hide

```
treatment <- tribble(
  ~ person,           ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
)
```

Task-6: Filling the missing values in the “person” column of the “treatment” tibble.

Hide

```
treatment %>%
  fill(person)
```

person <chr>	treatment <dbl>	response <dbl>
Derrick Whitmore	1	7
Derrick Whitmore	2	10
Derrick Whitmore	3	9
Katherine Burke	1	4

4 rows

Hide

```
NA
```

Case Study

Task-1: Loading data set

Hide

```
who
```

country <chr>	is… <chr>	is… <chr>	y… <dbl>	new_sp_m… <dbl>	new_sp_m1… <dbl>	new_sp_m2… <dbl>	new_sp_m3… <dbl>	new_s
Afghanistan	AF	AFG	1980	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1981	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1982	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1983	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1984	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1985	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1986	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1987	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1988	*NA*	*NA*	*NA*	*NA*	
Afghanistan	AF	AFG	1989	*NA*	*NA*	*NA*	*NA*	

1-10 of 7,240 rows | 1-9 of 60 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-2:Pivoting the “who” dataset from wide to long format, condensing columns into “cases” and capturing the original column names in “key”.

Hide

```
who1 <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  )
who1
```

country <chr>	iso2 <chr>	iso3 <chr>	year <dbl>	key <chr>	cases <dbl>
Afghanistan	AF	AFG	1997	new_sp_m014	0
Afghanistan	AF	AFG	1997	new_sp_m1524	10
Afghanistan	AF	AFG	1997	new_sp_m2534	6

country <chr>	iso2 <chr>	iso3 <chr>	year <dbl>	key <chr>	cases <dbl>
Afghanistan	AF	AFG	1997	new_sp_m3544	3
Afghanistan	AF	AFG	1997	new_sp_m4554	5
Afghanistan	AF	AFG	1997	new_sp_m5564	2
Afghanistan	AF	AFG	1997	new_sp_m65	0
Afghanistan	AF	AFG	1997	new_sp_f014	5
Afghanistan	AF	AFG	1997	new_sp_f1524	38
Afghanistan	AF	AFG	1997	new_sp_f2534	36

1-10 of 76,046 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-3:Counting the occurrences of each “key” in the “who1” dataset.

Hide

```
who1 %>%
  count(key)
```

key <chr>	n <int>
new_ep_f014	1032
new_ep_f1524	1021
new_ep_f2534	1021
new_ep_f3544	1021
new_ep_f4554	1017
new_ep_f5564	1017
new_ep_f65	1014
new_ep_m014	1038
new_ep_m1524	1026
new_ep_m2534	1020

1-10 of 56 rows Previous **1** 2 3 4 5 6 Next

Task-4:Replacing “newrel” with “new_rel” in the “key” column of the “who1” dataset to create “who2.”

Hide

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who2
```

country <chr>	iso2 <chr>	iso3 <chr>	year <dbl>	key <chr>	cases <dbl>
Afghanistan	AF	AFG	1997	new_sp_m014	0
Afghanistan	AF	AFG	1997	new_sp_m1524	10
Afghanistan	AF	AFG	1997	new_sp_m2534	6
Afghanistan	AF	AFG	1997	new_sp_m3544	3
Afghanistan	AF	AFG	1997	new_sp_m4554	5
Afghanistan	AF	AFG	1997	new_sp_m5564	2
Afghanistan	AF	AFG	1997	new_sp_m65	0
Afghanistan	AF	AFG	1997	new_sp_f014	5
Afghanistan	AF	AFG	1997	new_sp_f1524	38
Afghanistan	AF	AFG	1997	new_sp_f2534	36

1-10 of 76,046 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-5:Separating the “key” column in the “who2” dataset into “new,” “type,” and “sexage” columns using “_” as the separator to create “who3.”

Hide

```
who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
who3
```

country <chr>	iso2 <chr>	iso3 <chr>	year <dbl>	new <chr>	type <chr>	sexage <chr>	cases <dbl>
Afghanistan	AF	AFG	1997	new	sp	m014	0
Afghanistan	AF	AFG	1997	new	sp	m1524	10
Afghanistan	AF	AFG	1997	new	sp	m2534	6
Afghanistan	AF	AFG	1997	new	sp	m3544	3
Afghanistan	AF	AFG	1997	new	sp	m4554	5
Afghanistan	AF	AFG	1997	new	sp	m5564	2
Afghanistan	AF	AFG	1997	new	sp	m65	0
Afghanistan	AF	AFG	1997	new	sp	f014	5
Afghanistan	AF	AFG	1997	new	sp	f1524	38
Afghanistan	AF	AFG	1997	new	sp	f2534	36

1-10 of 76,046 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-6:Counting the occurrences of each unique value in the “new” column of the “who3” dataset.

Hide

```
who3 %>%
  count(new)
```

new	n
<chr>	<int>
new	76046

1 row

Task-7:Removing the “new”, “iso2”, and “iso3” columns from the “who3” dataset and assigning the result to “who4”.

Hide

```
who4 <- who3 %>%
  select(-new, -iso2, -iso3)
```

Task-8:Splitting the “sexage” column of the “who4” dataset into “sex” and “age” columns, separated by the first character, and assigning the result to “who5”.

Hide

```
who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)
who5
```

country	year	type	sex	age	cases
<chr>	<dbl>	<chr>	<chr>	<chr>	<dbl>
Afghanistan	1997	sp	m	014	0
Afghanistan	1997	sp	m	1524	10
Afghanistan	1997	sp	m	2534	6
Afghanistan	1997	sp	m	3544	3
Afghanistan	1997	sp	m	4554	5
Afghanistan	1997	sp	m	5564	2
Afghanistan	1997	sp	m	65	0
Afghanistan	1997	sp	f	014	5
Afghanistan	1997	sp	f	1524	38
Afghanistan	1997	sp	f	2534	36

1-10 of 76,046 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-9:Transforming the “who” dataset from wide to long format, adjusting column names, extracting meaningful variables, dropping unnecessary columns, and splitting the “sexage” column into “sex” and “age”.

Hide

```
who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  ) %>%
  mutate(
    key = stringr::str_replace(key, "newrel", "new_rel")
  ) %>%
  separate(key, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

country <chr>	year <dbl>	var <chr>	sex <chr>	age <chr>	cases <dbl>
Afghanistan	1997	sp	m	014	0
Afghanistan	1997	sp	m	1524	10
Afghanistan	1997	sp	m	2534	6
Afghanistan	1997	sp	m	3544	3
Afghanistan	1997	sp	m	4554	5
Afghanistan	1997	sp	m	5564	2
Afghanistan	1997	sp	m	65	0
Afghanistan	1997	sp	f	014	5
Afghanistan	1997	sp	f	1524	38
Afghanistan	1997	sp	f	2534	36

1-10 of 76,046 rows Previous **1** 2 3 4 5 6 … 100 Next

Hide

```
NA
```

CH-13: Relational data

Task-1:Loding the libraries

Hide

```
library(tidyverse)
library(nycflights13)
```

nycflights13

Task-1: airlines data

Hide

```
airlines
```

carrier <chr>	name <chr>
9E	Endeavor Air Inc.
AA	American Airlines Inc.
AS	Alaska Airlines Inc.
B6	JetBlue Airways
DL	Delta Air Lines Inc.
EV	ExpressJet Airlines Inc.
F9	Frontier Airlines Inc.
FL	AirTran Airways Corporation
HA	Hawaiian Airlines Inc.
MQ	Envoy Air

1-10 of 16 rows Previous **1** 2 Next

Task-2: airports data

Hide

```
airports
```

f…	name <chr>	lat <dbl>	lon <dbl>	alt <dbl>	tz <dbl>	d.. <chr>
04G	Lansdowne Airport	41.13047	-80.61958	1044	-5	A
06A	Moton Field Municipal Airport	32.46057	-85.68003	264	-6	A
06C	Schaumburg Regional	41.98934	-88.10124	801	-6	A
06N	Randall Airport	41.43191	-74.39156	523	-5	A
09J	Jekyll Island Airport	31.07447	-81.42778	11	-5	A
0A9	Elizabethton Municipal Airport	36.37122	-82.17342	1593	-5	A
0G6	Williams County Airport	41.46731	-84.50678	730	-5	A
0G7	Finger Lakes Regional Airport	42.88356	-76.78123	492	-5	A
0P2	Shoestring Aviation Airfield	39.79482	-76.64719	1000	-5	U
0S9	Jefferson County Intl	48.05381	-122.81064	108	-8	A

1-10 of 1,458 rows | 1-7 of 8 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-3: planes data

Hide

```
planes
```

tailnum <chr>	y… <int>	type <chr>	manufacturer <chr>	model <chr>	engines <int>	se… <int>	sp… <int>
N10156	2004	Fixed wing multi engine	EMBRAER	EMB-145XR	2	55	*NA*
N102UW	1998	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N103US	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N104UW	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N10575	2002	Fixed wing multi engine	EMBRAER	EMB-145LR	2	55	*NA*
N105UW	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N107US	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N108UW	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N109UW	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*
N110UW	1999	Fixed wing multi engine	AIRBUS INDUSTRIE	A320-214	2	182	*NA*

1-10 of 3,322 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-4: weather data

Hide

```
weather
```

origin <chr>	year <int>	month <int>	day <int>	hour <int>	temp <dbl>	dewp <dbl>	humid <dbl>	wind_dir <dbl>	wind_speed <dbl>
EWR	2013	1	1	1	39.02	26.06	59.37	270	10.35702
EWR	2013	1	1	2	39.02	26.96	61.63	250	8.05546
EWR	2013	1	1	3	39.02	28.04	64.43	240	11.50780
EWR	2013	1	1	4	39.92	28.04	62.21	250	12.65858
EWR	2013	1	1	5	39.02	28.04	64.43	260	12.65858
EWR	2013	1	1	6	37.94	28.04	67.21	240	11.50780
EWR	2013	1	1	7	39.02	28.04	64.43	240	14.96014
EWR	2013	1	1	8	39.92	28.04	62.21	250	10.35702
EWR	2013	1	1	9	39.92	28.04	62.21	260	14.96014
EWR	2013	1	1	10	41.00	28.04	59.65	260	13.80936

1-10 of 26,115 rows | 1-10 of 15 columns Previous **1** 2 3 4 5 6 … 100 Next

Keys

Task-1Counting the occurrences of each tail number in the “planes” table and filtering for those with more than one occurrence.

Hide

```
planes %>%
  count(tailnum) %>%
  filter(n > 1)
```

0 rows

Task-2:Counting the occurrences of each combination of year, month, day, hour, and origin in the “weather” table and filtering for those with more than one occurrence.

Hide

```
weather %>%
  count(year, month, day, hour, origin) %>%
  filter(n > 1)
```

year <int>	month <int>	day <int>	hour <int>	origin <chr>	n <int>
2013	11	3	1	EWR	2
2013	11	3	1	JFK	2
2013	11	3	1	LGA	2

3 rows

Task-3:Counting the occurrences of each combination of year, month, day, and flight in the “flights” table and filtering for those with more than one occurrence.

Hide

```
flights %>%
  count(year, month, day, flight) %>%
  filter(n > 1)
```

year <int>	month <int>	day <int>	flight <int>	n <int>
2013	1	1	1	2
2013	1	1	3	2
2013	1	1	4	2
2013	1	1	11	3
2013	1	1	15	2
2013	1	1	21	2
2013	1	1	27	4

year <int>	month <int>	day <int>	flight <int>	n <int>
2013	1	1	31	2
2013	1	1	32	2
2013	1	1	35	2

1-10 of 29,768 rows | Previous **1** 2 3 4 5 6 … 100 Next

Task-4:Counting the occurrences of each combination of year, month, day, and tail number in the “flights” table and filtering for those with more than one occurrence.

Hide

```
flights %>%
  count(year, month, day, tailnum) %>%
  filter(n > 1)
```

year <int>	month <int>	day <int>	tailnum <chr>	n <int>
2013	1	1	N0EGMQ	2
2013	1	1	N11189	2
2013	1	1	N11536	2
2013	1	1	N11544	3
2013	1	1	N11551	2
2013	1	1	N12540	2
2013	1	1	N12567	2
2013	1	1	N13123	2
2013	1	1	N13538	3
2013	1	1	N13566	3

1-10 of 64,928 rows | Previous **1** 2 3 4 5 6 … 100 Next

Mutating joins

Task-1: Creating a subset of the “flights” table named “flights2” containing columns from “year” to “day”, “hour”, “origin”, “dest”, “tailnum”, and “carrier”.

Hide

```
flights2 <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
flights2
```

year <int>	month <int>	day <int>	hour <dbl>	origin <chr>	dest <chr>	tailnum <chr>	carrier <chr>
2013	1	1	5	EWR	IAH	N14228	UA
2013	1	1	5	LGA	IAH	N24211	UA
2013	1	1	5	JFK	MIA	N619AA	AA
2013	1	1	5	JFK	BQN	N804JB	B6
2013	1	1	6	LGA	ATL	N668DN	DL
2013	1	1	5	EWR	ORD	N39463	UA
2013	1	1	6	EWR	FLL	N516JB	B6
2013	1	1	6	LGA	IAD	N829AS	EV
2013	1	1	6	JFK	MCO	N593JB	B6
2013	1	1	6	LGA	ORD	N3ALAA	AA

1-10 of 336,776 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-2:Removing the “origin” and “dest” columns from “flights2” table and then performing a left join with the “airlines” table, using the “carrier” column as the key for matching.

Hide

```
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
```

year <int>	month <int>	day <int>	hour <dbl>	tailnum <chr>	carrier <chr>	name <chr>
2013	1	1	5	N14228	UA	United Air Lines Inc.
2013	1	1	5	N24211	UA	United Air Lines Inc.
2013	1	1	5	N619AA	AA	American Airlines Inc.
2013	1	1	5	N804JB	B6	JetBlue Airways
2013	1	1	6	N668DN	DL	Delta Air Lines Inc.
2013	1	1	5	N39463	UA	United Air Lines Inc.
2013	1	1	6	N516JB	B6	JetBlue Airways
2013	1	1	6	N829AS	EV	ExpressJet Airlines Inc.
2013	1	1	6	N593JB	B6	JetBlue Airways
2013	1	1	6	N3ALAA	AA	American Airlines Inc.

1-10 of 336,776 rows Previous **1** 2 3 4 5 6 … 100 Next

Task-3:Shortening the command by removing “selecting” and directly “mutating” the “name” column with the corresponding airline names from the “airlines” table based on the “carrier” column.

Hide

```
flights2 %>%
  select(-origin, -dest) %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])
```

year <int>	month <int>	day <int>	hour <dbl>	tailnum <chr>	carrier <chr>	name <chr>
2013	1	1	5	N14228	UA	United Air Lines Inc.
2013	1	1	5	N24211	UA	United Air Lines Inc.
2013	1	1	5	N619AA	AA	American Airlines Inc.
2013	1	1	5	N804JB	B6	JetBlue Airways
2013	1	1	6	N668DN	DL	Delta Air Lines Inc.
2013	1	1	5	N39463	UA	United Air Lines Inc.
2013	1	1	6	N516JB	B6	JetBlue Airways
2013	1	1	6	N829AS	EV	ExpressJet Airlines Inc.
2013	1	1	6	N593JB	B6	JetBlue Airways
2013	1	1	6	N3ALAA	AA	American Airlines Inc.

1-10 of 336,776 rows Previous **1** 2 3 4 5 6 … 100 Next

Understanding joins

Task-1:Creating two tibbles, “x” and “y”, each with a “key” column and an associated “val_x” or “val_y” column, respectively.

Hide

```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     3, "x3"
)
y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2",
     4, "y3"
)

x
```

key <dbl>	val_x <chr>
1	x1
2	x2

key	val_x
<dbl>	<chr>
3	x3

3 rows

Hide

```
y
```

key	val_y
<dbl>	<chr>
1	y1
2	y2
4	y3

3 rows

Inner join

Task-1:Joining tibbles `x` and `y` using an inner join operation based on the “key” column.

Hide

```
x %>%
  inner_join(y, by = "key")
```

key	val_x	val_y
<dbl>	<chr>	<chr>
1	x1	y1
2	x2	y2

2 rows

Duplicate keys

Task-1: Joining tibble x with tibble y using the common column “key”.

Hide

```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     1, "x4"
)
y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2"
)
```

Task-2:Performing a left join between tibble `x` and tibble `y` based on the common column “key”.

Hide

```
left_join(x, y, by = "key")
```

key <dbl>	val_x <chr>	val_y <chr>
1	x1	y1
2	x2	y2
2	x3	y2
1	x4	y1

4 rows

Task-3:Creating two tibbles, `x` and `y`, with columns “key”, “val_x”, and “val_y”, populated with corresponding values.

Hide

```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     3, "x4"
)
y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2",
     2, "y3",
     3, "y4"
)
```

Task-4:Performing a left join on tibbles `x` and `y` using the “key” column as the join key.

Hide

```
left_join(x, y, by = "key")
```

```
Warning: Detected an unexpected many-to-many relationship between `x` and `y`.
```

key <dbl>	val_x <chr>	val_y <chr>
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

6 rows

Defining the key columns

Task-1:Performing a left join between the `flights2` tibble and the `weather` tibble.

Hide

```
flights2 %>%
  left_join(weather)
```

```
Joining with `by = join_by(year, month, day, hour, origin)`
```

year <int>	month <int>	day <int>	hour <dbl>	origin <chr>	dest <chr>	tailnum <chr>	carrier <chr>	temp <dbl>	dewp <dbl>
2013	1	1	5	EWR	IAH	N14228	UA	39.02	28.04
2013	1	1	5	LGA	IAH	N24211	UA	39.92	24.98
2013	1	1	5	JFK	MIA	N619AA	AA	39.02	26.96
2013	1	1	5	JFK	BQN	N804JB	B6	39.02	26.96
2013	1	1	6	LGA	ATL	N668DN	DL	39.92	24.98
2013	1	1	5	EWR	ORD	N39463	UA	39.02	28.04
2013	1	1	6	EWR	FLL	N516JB	B6	37.94	28.04
2013	1	1	6	LGA	IAD	N829AS	EV	39.92	24.98
2013	1	1	6	JFK	MCO	N593JB	B6	37.94	26.96
2013	1	1	6	LGA	ORD	N3ALAA	AA	39.92	24.98

1-10 of 336,776 rows | 1-10 of 18 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-2:Performing a left join between the `flights2` tibble and the `planes` tibble using the “tailnum” column as the key.

Hide

```
flights2 %>%
  left_join(planes, by = "tailnum")
```

year.x <int>	mo… <int>	d.. <int>	h… <dbl>	origin <chr>	d… <chr>	tailnum <chr>	carrier <chr>	year.y <int>	type <chr>
2013	1	1	5	EWR	IAH	N14228	UA	1999	Fixed wing multi engine
2013	1	1	5	LGA	IAH	N24211	UA	1998	Fixed wing multi engine
2013	1	1	5	JFK	MIA	N619AA	AA	1990	Fixed wing multi engine
2013	1	1	5	JFK	BQN	N804JB	B6	2012	Fixed wing multi engine
2013	1	1	6	LGA	ATL	N668DN	DL	1991	Fixed wing multi engine
2013	1	1	5	EWR	ORD	N39463	UA	2012	Fixed wing multi engine
2013	1	1	6	EWR	FLL	N516JB	B6	2000	Fixed wing multi engine
2013	1	1	6	LGA	IAD	N829AS	EV	1998	Fixed wing multi engine
2013	1	1	6	JFK	MCO	N593JB	B6	2004	Fixed wing multi engine
2013	1	1	6	LGA	ORD	N3ALAA	AA	*NA*	*NA*

1-10 of 336,776 rows | 1-10 of 16 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-3:Performing a left join between the `flights2` tibble and the `airports` tibble, matching the “dest” column from `flights2` with the “faa” column from `airports` .

Hide

```
flights2 %>%
  left_join(airports, c("dest" = "faa"))
```

y… <int>	mo… <int>	d.. <int>	h… <dbl>	origin <chr>	d… <chr>	tailnum <chr>	carrier <chr>	name <chr>	
2013	1	1	5	EWR	IAH	N14228	UA	George Bush Intercontinental	2
2013	1	1	5	LGA	IAH	N24211	UA	George Bush Intercontinental	2
2013	1	1	5	JFK	MIA	N619AA	AA	Miami Intl	2
2013	1	1	5	JFK	BQN	N804JB	B6	*NA*	
2013	1	1	6	LGA	ATL	N668DN	DL	Hartsfield Jackson Atlanta Intl	3
2013	1	1	5	EWR	ORD	N39463	UA	Chicago Ohare Intl	4
2013	1	1	6	EWR	FLL	N516JB	B6	Fort Lauderdale Hollywood Intl	2
2013	1	1	6	LGA	IAD	N829AS	EV	Washington Dulles Intl	3
2013	1	1	6	JFK	MCO	N593JB	B6	Orlando Intl	2
2013	1	1	6	LGA	ORD	N3ALAA	AA	Chicago Ohare Intl	4

1-10 of 336,776 rows | 1-10 of 15 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-4:Performing a left join between the `flights2` tibble and the `airports` tibble, matching the “origin” column from `flights2` with the “faa” column from `airports` .

Hide

```
flights2 %>%
  left_join(airports, c("origin" = "faa"))
```

year <int>	mo… <int>	d.. <int>	h… <dbl>	origin <chr>	dest <chr>	tailnum <chr>	carrier <chr>	name <chr>	lat <dbl>
2013	1	1	5	EWR	IAH	N14228	UA	Newark Liberty Intl	40.69250
2013	1	1	5	LGA	IAH	N24211	UA	La Guardia	40.77725
2013	1	1	5	JFK	MIA	N619AA	AA	John F Kennedy Intl	40.63975
2013	1	1	5	JFK	BQN	N804JB	B6	John F Kennedy Intl	40.63975
2013	1	1	6	LGA	ATL	N668DN	DL	La Guardia	40.77725
2013	1	1	5	EWR	ORD	N39463	UA	Newark Liberty Intl	40.69250
2013	1	1	6	EWR	FLL	N516JB	B6	Newark Liberty Intl	40.69250
2013	1	1	6	LGA	IAD	N829AS	EV	La Guardia	40.77725
2013	1	1	6	JFK	MCO	N593JB	B6	John F Kennedy Intl	40.63975
2013	1	1	6	LGA	ORD	N3ALAA	AA	La Guardia	40.77725

1-10 of 336,776 rows | 1-10 of 15 columns Previous **1** 2 3 4 5 6 … 100 Next

Filtering joins

Task-1: Calculating the top 10 destinations by counting the occurrences in the “dest” column of the `flights` tibble, sorted in descending order, and then displaying the result.

Hide

```
top_dest <- flights %>%
  count(dest, sort = TRUE) %>%
  head(10)
top_dest
```

dest <chr>	n <int>
ORD	17283
ATL	17215
LAX	16174
BOS	15508
MCO	14082
CLT	14064

dest <chr>	n <int>
SFO	13331
FLL	12055
MIA	11728
DCA	9705

1-10 of 10 rows

Task-2: Filtering the `flights` tibble to include only rows where the destination (`dest`) matches any of the top 10 destinations identified in the previous step.

Hide

```
flights %>%
  filter(dest %in% top_dest$dest)
```

y… <int>	mo… <int>	d.. <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	c <
2013	1	1	542	540	2	923	850	33	A
2013	1	1	554	600	-6	812	837	-25	D
2013	1	1	554	558	-4	740	728	12	U
2013	1	1	555	600	-5	913	854	19	B
2013	1	1	557	600	-3	838	846	-8	B
2013	1	1	558	600	-2	753	745	8	A
2013	1	1	558	600	-2	924	917	7	U
2013	1	1	558	600	-2	923	937	-14	U
2013	1	1	559	559	0	702	706	-4	B
2013	1	1	600	600	0	851	858	-7	B

1-10 of 141,145 rows | 1-10 of 19 columns Previous **1** 2 3 4 5 6 … 100 Next

Hide

```
#%in% operator in R is used to check if elements in one vector are present in another vector
```

Task-3: Selecting rows from the `flights` dataset where the destination airport matches one of the top 10 destinations previously identified.

Hide

```
flights %>%
  semi_join(top_dest)
```

```
Joining with `by = join_by(dest)`
```

y… <int>	mo… <int>	d.. <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	c <
2013	1	1	542	540	2	923	850	33	A
2013	1	1	554	600	-6	812	837	-25	D
2013	1	1	554	558	-4	740	728	12	U
2013	1	1	555	600	-5	913	854	19	B
2013	1	1	557	600	-3	838	846	-8	B
2013	1	1	558	600	-2	753	745	8	A
2013	1	1	558	600	-2	924	917	7	U
2013	1	1	558	600	-2	923	937	-14	U
2013	1	1	559	559	0	702	706	-4	B
2013	1	1	600	600	0	851	858	-7	B

1-10 of 141,145 rows | 1-10 of 19 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-4: Filtering out flights with tail numbers present in the planes dataset and counting the occurrences of each unique tail number, sorting the result.

Hide

```
flights %>%
  anti_join(planes, by = "tailnum") %>%
  count(tailnum, sort = TRUE)
```

tailnum <chr>	n <int>
NA	2512
N725MQ	575
N722MQ	513
N723MQ	507
N713MQ	483
N735MQ	396
N0EGMQ	371
N534MQ	364
N542MQ	363
N531MQ	349

1-10 of 722 rows Previous **1** 2 3 4 5 6 … 73 Next

Set operations

Task-1:creating two tibbles, df1 and df2, each with columns x and y, containing sample data.

Hide

```
df1 <- tribble(
  ~x, ~y,
   1,  1,
   2,  1
)
df2 <- tribble(
  ~x, ~y,
   1,  1,
   1,  2
)
```

Task-2:performing set operations on the tibbles df1 and df2, including intersection, union, and set differences.

Hide

```
intersect(df1, df2)
```

x <dbl>	y <dbl>
1	1

1 row

Hide

```
union(df1, df2)
```

x <dbl>	y <dbl>
1	1
2	1
1	2

3 rows

Hide

```
setdiff(df1, df2)
```

x <dbl>	y <dbl>
2	1

1 row

Hide

```
setdiff(df2, df1)
```

x <dbl>	y <dbl>
1	2

1 row

CH-14: Strings

Basic Info:string1 (Info:string1) <- “This is a string” string2 <- ‘If I want to include a “quote” inside a string, I use single quotes’

Task-1:To include a literal single or double quote in a string you can use to “escape” it

Hide

```
double_quote <- "\"" # or '"'
single_quote <- '\'' # or "'"
```

Task-2: Understanding the character

Hide

```
x <- c("\"", "\\") #backslash is escape character
x
```

```
[1] "\"" "\\"
```

Hide

```
writeLines(x)
```

```
"
\
```

String length

Task-1:

Hide

```
str_length(c("a", "R for data science", NA))
```

```
[1]  1 18 NA
```

Combining strings

Task-1:Combining the strings

Hide

```
str_c("x", "y")
```

```
[1] "xy"
```

Hide

```
str_c("x", "y", "z")
```

```
[1] "xyz"
```

Task-2:Using the sep argument to control how they’re separated.

Hide

```
str_c("x", "y", sep = ", ")
```

```
[1] "x, y"
```

Task-3:Performing concatenation with “|” and “-” at both ends of each element of vector x, and replacing NA values with empty strings before concatenation.

Hide

```
x <- c("abc", NA)
str_c("|-", x, "-|")
```

```
[1] "|-abc-|" NA
```

Hide

```
str_c("|-", str_replace_na(x), "-|")
```

```
[1] "|-abc-|" "|-NA-|"
```

Task-4: concatenating each element of the vector c(“a”, “b”, “c”) with a prefix “prefix-” and a suffix “-suffix”.

Hide

```
str_c("prefix-", c("a", "b", "c"), "-suffix")
```

```
[1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

Task-5: combining strings

Hide

```
name <- "Hadley"
time_of_day <- "morning"
birthday <- FALSE

str_c(
  "Good ", time_of_day, " ", name,
  if (birthday) " and HAPPY BIRTHDAY",
  "."
)
```

```
[1] "Good morning Hadley."
```

Subsetting strings

Task-1:Extracting the first three characters from each element in the vector `x` using `str_sub`.

Hide

```
x <- c("Apple", "Banana", "Pear")
str_sub(x, 1, 3)
```

```
[1] "App" "Ban" "Pea"
```

Task-2:negative numbers count backwards from end

Hide

```
str_sub(x, -3, -1)
```

```
[1] "ple" "ana" "ear"
```

Task-3:using the assignment form of str_sub() to modify strings

Hide

```
str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
x
```

```
[1] "apple"  "banana" "pear"
```

Locales

Task-1:Changing the case

Hide

```
str_to_upper(c("i", "ı"))
```

```
[1] "I" "I"
```

Hide

```
str_to_upper(c("i", "ı"), locale = "tr")
```

```
[1] "İ" "I"
```

Task-2:Sorting the character vector x alphabetically using the English (en) locale and the Hawaiian (haw) locale.

Hide

```
x <- c("apple", "eggplant", "banana")
str_sort(x, locale = "en")
```

```
[1] "apple"    "banana"   "eggplant"
```

Hide

```
str_sort(x, locale = "haw")
```

```
[1] "apple"    "eggplant" "banana"
```

Matching patterns with regular expressions

Basic matches

Task-1:Searching for the pattern “an” within each element of `x` and displaying the matches.

Hide

```
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

```
[2] │ b<an><an>a
```

Task-2:Displaying elements in `x` where any character is followed by “a” and then any character.

Hide

```
str_view(x, ".a.")
```

```
[2] │ <ban>ana
[3] │ p<ear>
```

Task-3

Hide

```
# To create the regular expression, we need \\
dot <- "\\."

# But the expression itself only contains one:
writeLines(dot)
```

```
\.
```

Hide

```
# And this tells R to look for an explicit .
str_view(c("abc", "a.c", "bef"), "a\\.c")
```

```
[2] │ <a.c>
```

Task-4: Displaying elements in `x` where the sequence “\” occurs.

Hide

```
x <- "a\\b"
writeLines(x)
```

```
a\b
```

Hide

```
str_view(x, "\\\\")
```

```
[1] │ a<\>b
```

Anchors

Task-1: Displaying elements in `x` that start with “a” and end with “a” respectively.

Hide

```
x <- c("apple", "banana", "pear")
str_view(x, "^a")
```

```
[1] │ <a>pple
```

Hide

```
str_view(x, "a$")
```

```
[2] │ banan<a>
```

Task-2: Highlighting “apple” occurrences in `x` and instances where it’s the only content.

Hide

```
x <- c("apple pie", "apple", "apple cake")
str_view(x, "apple")
```

```
[1] │ <apple> pie
[2] │ <apple>
[3] │ <apple> cake
```

Hide

```
str_view(x, "^apple$")
```

```
[2] │ <apple>
```

Character classes and alternatives

Task-1: Visualizing patterns matching “a.c”, “a*c”, and “a c” in the provided character vector.

Hide

```
str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c")
```

```
[2] │ <a.c>
```

Hide

```
str_view(c("abc", "a.c", "a*c", "a c"), ".[*]c")
```

```
[3] │ <a*c>
```

Hide

```
str_view(c("abc", "a.c", "a*c", "a c"), "a[ ]")
```

```
[4] │ <a >c
```

Task-2: Visualizing patterns matching “grey” or “gray” in the provided character vector.

Hide

```
str_view(c("grey", "gray"), "gr(e|a)y")
```

```
[1] │ <grey>
[2] │ <gray>
```

Repetition

Task-1:Identifying patterns “CC” or “C” in the string “1888 is the longest year in Roman numerals

Hide

```
x <- "1888 is the longest year in Roman numerals: MDCCCLXXXVIII"
str_view(x, "CC?")
```

```
[1] │ 1888 is the longest year in Roman numerals: MD<CC><C>LXXXVIII
```

Task-2: Viewing the pattern “CC”

Hide

```
str_view(x, "CC+")
```

```
[1] │ 1888 is the longest year in Roman numerals: MD<CCC>LXXXVIII
```

Task-3: Viewing the pattern “C[LX]+”

Hide

```
str_view(x, 'C[LX]+')
```

```
[1] │ 1888 is the longest year in Roman numerals: MDCC<CLXXX>VIII
```

Task-4:Viewing the pattern “C{2},C{2,},c{2,3}”

Hide

```
str_view(x, "C{2}")
```

```
[1] │ 1888 is the longest year in Roman numerals: MD<CC>CLXXXVIII
```

Hide

```
str_view(x, "C{2,}")
```

```
[1] │ 1888 is the longest year in Roman numerals: MD<CCC>LXXXVIII
```

Hide

```
str_view(x, "C{2,3}")
```

```
[1] │ 1888 is the longest year in Roman numerals: MD<CCC>LXXXVIII
```

Grouping and backreferences

Task-1:Grouping

Hide

```
str_view(fruit, "(..)\\1", match = TRUE)
```

```
 [4] │ b<anan>a
[20] │ <coco>nut
[22] │ <cucu>mber
[41] │ <juju>be
[56] │ <papa>ya
[73] │ s<alal> berry
```

Detect matches

Task-1: Checking for the presence of the letter “e” in each word

Hide

```
x <- c("apple", "banana", "pear")
str_detect(x, "e")
```

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

Task-2:Checking how many common words start with t

Hide

```
sum(str_detect(words, "^t"))
```

```
[1] 65
```

Task-3: Checking proportion of common words end with a vowel

Hide

```
mean(str_detect(words, "[aeiou]$"))
```

```
[1] 0.2765306
```

Task-4:Finding all words containing at least one vowel, and negate

Hide

```
no_vowels_1 <- !str_detect(words, "[aeiou]")
```

Task-5:Finding all words consisting only of consonants (non-vowels)

Hide

```
no_vowels_2 <- str_detect(words, "^[^aeiou]+$")
identical(no_vowels_1, no_vowels_2)
```

```
[1] TRUE
```

Task-6: Filtering words that end with the letter “x” from a list of words.

Hide

```
words[str_detect(words, "x$")]
```

```
[1] "box" "sex" "six" "tax"
```

Hide

```
str_subset(words, "x$")
```

```
[1] "box" "sex" "six" "tax"
```

Task-7: Filtering a tibble for words that end with “x”.

Hide

```
df <- tibble(
  word = words,
  i = seq_along(word)
)
df %>%
  filter(str_detect(word, "x$"))
```

word <chr>	i <int>
box	108
sex	747
six	772
tax	841

4 rows

Task-8:Counting the occurrences of “a” in each element of a character vector.

Hide

```
x <- c("apple", "banana", "pear")
str_count(x, "a")
```

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

Task-9: Seeing average of how many vowels per word

Hide

```
mean(str_count(words, "[aeiou]"))
```

```
[1] 1.991837
```

Task-10: Adding columns to a tibble to count vowels and consonants in each word.

Hide

```
df %>%
  mutate(
    vowels = str_count(word, "[aeiou]"),
    consonants = str_count(word, "[^aeiou]")
  )
```

word <chr>	i <int>	vowels <int>	consonants <int>
a	1	1	0
able	2	2	2
about	3	3	2

word <chr>	i <int>	vowels <int>	consonants <int>
absolute	4	4	4
accept	5	2	4
account	6	3	4
achieve	7	4	3
across	8	2	4
act	9	1	2
active	10	3	3

Task-11:Counting “aba” occurrences in “abababa” and showing all “aba” instances.

Hide

```
str_count("abababa", "aba")
```

```
[1] 2
```

Hide

```
str_view_all("abababa", "aba")
```

```
Warning: `str_view_all()` was deprecated in stringr 1.5.0.
Please use `str_view()` instead.
```

```
[1] │ <aba>b<aba>
```

Extract matches

Task-1: Displaying the length of sentences and showing the first few sentences.

Hide

```
length(sentences)
```

```
[1] 720
```

Hide

```
head(sentences)
```

```
[1] "The birch canoe slid on the smooth planks."  "Glue the sheet to the dark blue backgroun
d."
[3] "It's easy to tell the depth of a well."      "These days a chicken leg is a rare dish."
[5] "Rice is often served in round bowls."        "The juice of lemons makes fine punch."
```

Task-2: Creating a string pattern to match colors by concatenating them with a pipe delimiter.

Hide

```
colours <- c("red", "orange", "yellow", "green", "blue", "purple")
colour_match <- str_c(colours, collapse = "|")
colour_match
```

```
[1] "red|orange|yellow|green|blue|purple"
```

Task-3: Filter sentences for colors and extract matches, showing the first few.

Hide

```
has_colour <- str_subset(sentences, colour_match)
matches <- str_extract(has_colour, colour_match)
head(matches)
```

```
[1] "blue" "blue" "red"  "red"  "red"  "blue"
```

Task-4:Showing all sentences containing multiple colors and highlight the matches.

Hide

```
more <- sentences[str_count(sentences, colour_match) > 1]
str_view_all(more, colour_match)
```

```
[1] │ It is hard to erase <blue> or <red> ink.
[2] │ The <green> light in the brown box flicke<red>.
[3] │ The sky in the west is tinged with <orange> <red>.
```

Task-5:Extracting all color matches from the subset of sentences containing multiple colors.

Hide

```
str_extract(more, colour_match)
```

```
[1] "blue"   "green"  "orange"
```

Task-6:Extracting all occurrences of colors from the subset of sentences containing multiple colors.

Hide

```
str_extract_all(more, colour_match)
```

```
[[1]]
[1] "blue" "red"

[[2]]
[1] "green" "red"

[[3]]
[1] "orange" "red"
```

Task-7: Extracting colors from sentences with multiple colors and simplify, also extract lowercase letters from each element in x and simplify.

Hide

```
str_extract_all(more, colour_match, simplify = TRUE)
```

```
     [,1]     [,2]
[1,] "blue"   "red"
[2,] "green"  "red"
[3,] "orange" "red"
```

Hide

```
x <- c("a", "a b", "a b c")
str_extract_all(x, "[a-z]", simplify = TRUE)
```

```
     [,1] [,2] [,3]
[1,] "a"  ""   ""
[2,] "a"  "b"  ""
[3,] "a"  "b"  "c"
```

Grouped matches

Task-1: Extracting sentences containing nouns defined by a pattern, then extracts the nouns from those sentences.

Hide

```
noun <- "(a|the) ([^ ]+)"

has_noun <- sentences %>%
  str_subset(noun) %>%
  head(10)
has_noun %>%
  str_extract(noun)
```

```
 [1] "the smooth" "the sheet"  "the depth"  "a chicken"  "the parked" "the sun"    "the huge"
"the ball"
 [9] "the woman"  "a helps"
```

Task-2:

Hide

```
has_noun %>%
  str_match(noun)
```

```
      [,1]         [,2]  [,3]
 [1,] "the smooth" "the" "smooth"
 [2,] "the sheet"  "the" "sheet"
 [3,] "the depth"  "the" "depth"
 [4,] "a chicken"  "a"   "chicken"
 [5,] "the parked" "the" "parked"
 [6,] "the sun"    "the" "sun"
 [7,] "the huge"   "the" "huge"
 [8,] "the ball"   "the" "ball"
 [9,] "the woman"  "the" "woman"
[10,] "a helps"    "a"   "helps"
```

Task-3:Creating a tibble with columns ‘article’ and ‘noun’ extracted from sentences based on a pattern.

Hide

```
tibble(sentence = sentences) %>%
  tidyr::extract(
    sentence, c("article", "noun"), "(a|the) ([^ ]+)",
    remove = FALSE
  )
```

sentence <chr>	**article** <chr>	**noun** <chr>
The birch canoe slid on the smooth planks.	the	smooth
Glue the sheet to the dark blue background.	the	sheet
It's easy to tell the depth of a well.	the	depth
These days a chicken leg is a rare dish.	a	chicken
Rice is often served in round bowls.	*NA*	*NA*
The juice of lemons makes fine punch.	*NA*	*NA*
The box was thrown beside the parked truck.	the	parked
The hogs were fed chopped corn and garbage.	*NA*	*NA*
Four hours of steady work faced us.	*NA*	*NA*
A large size in stockings is hard to sell.	*NA*	*NA*

Replacing matches

Task-1: Replacing the first vowel in each word of x with a hyphen. Replacing all vowels in each word of x with a hyphen.

Hide

```
x <- c("apple", "pear", "banana")
str_replace(x, "[aeiou]", "-")
```

```
[1] "-pple"  "p-ar"   "b-nana"
```

Hide

```
str_replace_all(x, "[aeiou]", "-")
```

```
[1] "-ppl-"  "p--r"   "b-n-n-"
```

Task-2: Replacing numeric values in x with their corresponding word representations.

Hide

```
x <- c("1 house", "2 cars", "3 people")
str_replace_all(x, c("1" = "one", "2" = "two", "3" = "three"))
```

```
[1] "one house"    "two cars"     "three people"
```

Task-3:Reordering words in sentences by swapping the second and third word positions.

Hide

```
sentences %>%
  str_replace("([^ ]+) ([^ ]+) ([^ ]+)", "\\1 \\3 \\2") %>%
  head(5)
```

```
[1] "The canoe birch slid on the smooth planks."  "Glue sheet the to the dark blue backgroun
d."
[3] "It's to easy tell the depth of a well."      "These a days chicken leg is a rare dish."
[5] "Rice often is served in round bowls."
```

Splitting

Task-1: Splitting the first five sentences into words.

Hide

```
sentences %>%
  head(5) %>%
  str_split(" ")
```

```
[[1]]
[1] "The"     "birch"   "canoe"   "slid"    "on"      "the"     "smooth"  "planks."

[[2]]
[1] "Glue"        "the"         "sheet"       "to"          "the"         "dark"        "blu
e"
[8] "background."

[[3]]
[1] "It's"  "easy"  "to"    "tell"  "the"   "depth" "of"    "a"     "well."

[[4]]
[1] "These"   "days"    "a"       "chicken" "leg"     "is"      "a"       "rare"    "dish."

[[5]]
[1] "Rice"   "is"     "often"  "served" "in"     "round"  "bowls."
```

Task-2:Splitting the string ‘a|b|c|d’ by ‘|’ into a vector of elements.

Hide

```
"a|b|c|d" %>%
  str_split("\\|") %>%
  .[[1]]
```

```
[1] "a" "b" "c" "d"
```

Task-3:Splitting the first 5 sentences by space into a matrix of words.

Hide

```
sentences %>%
  head(5) %>%
  str_split(" ", simplify = TRUE)
```

```
     [,1]    [,2]    [,3]    [,4]      [,5]  [,6]    [,7]     [,8]          [,9]
[1,] "The"   "birch" "canoe" "slid"    "on"  "the"   "smooth" "planks."     ""
[2,] "Glue"  "the"   "sheet" "to"      "the" "dark"  "blue"   "background." ""
[3,] "It's"  "easy"  "to"    "tell"    "the" "depth" "of"     "a"           "well."
[4,] "These" "days"  "a"     "chicken" "leg" "is"    "a"      "rare"        "dish."
[5,] "Rice"  "is"    "often" "served"  "in"  "round" "bowls." ""            ""
```

Task-4:Splitting each field string into two parts at the first occurrence of ‘:’.

Hide

```
fields <- c("Name: Hadley", "Country: NZ", "Age: 35")
fields %>% str_split(": ", n = 2, simplify = TRUE)
```

```
     [,1]      [,2]
[1,] "Name"    "Hadley"
[2,] "Country" "NZ"
[3,] "Age"     "35"
```

Task-5: Display word boundaries, split by spaces, and split by word boundaries, respectively.

Hide

```
x <- "This is a sentence.  This is another sentence."
str_view_all(x, boundary("word"))
```

```
[1] │ <This> <is> <a> <sentence>.  <This> <is> <another> <sentence>.
```

Hide

```
str_split(x, " ")[[1]]
```

```
[1] "This"      "is"        "a"         "sentence." ""          "This"      "is"        "anot
her"
[9] "sentence."
```

Hide

```
str_split(x, boundary("word"))[[1]]
```

```
[1] "This"     "is"       "a"        "sentence" "This"     "is"       "another"  "sentence"
```

Other types of pattern

Task-1:

Hide

```
# The regular call:
str_view(fruit, "nana")
```

```
[4] │ ba<nana>
```

Hide

```
# Is shorthand for
str_view(fruit, regex("nana"))
```

```
[4] │ ba<nana>
```

Task-2:Visualizing occurrences of “banana” in different case variations.

Hide

```
bananas <- c("banana", "Banana", "BANANA")
str_view(bananas, "banana")
```

```
[1] │ <banana>
```

Hide

```
str_view(bananas, regex("banana", ignore_case = TRUE))
```

```
[1] │ <banana>
[2] │ <Banana>
[3] │ <BANANA>
```

Task-3: Extracting all lines starting with “Line” from the text.

Hide

```
x <- "Line 1\nLine 2\nLine 3"
str_extract_all(x, "^Line")[[1]]
```

```
[1] "Line"
```

Task-4: Extracting all occurrences of lines starting with “Line” from the text, considering each line separately.

Hide

```
str_extract_all(x, regex("^Line", multiline = TRUE))[[1]]
```

```
[1] "Line" "Line" "Line"
```

Task-5:Creating a regular expression pattern for phone numbers, allowing for variations in formatting, and attempting to match it against the provided phone number.

Hide

```
phone <- regex("
  \\(?     # optional opening parens
  (\\d{3}) # area code
  [) -]?   # optional closing parens, space, or dash
  (\\d{3}) # another three numbers
  [ -]?    # optional space or dash
  (\\d{3}) # three more numbers
  ", comments = TRUE)

str_match("514-791-8141", phone)
```

```
     [,1]          [,2]  [,3]  [,4]
[1,] "514-791-814" "514" "791" "814"
```

Task-6:Installling the package and Benchmarking string detection in “sentences” using fixed and regex patterns 20 times each, comparing performance with microbenchmark.

Hide

```
package_to_install <- c("microbenchmark")

for (package_name in package_to_install) {
  if (!requireNamespace(package_name, quietly = TRUE)) {
    install.packages(package_name)
  }
}
library(microbenchmark)

microbenchmark::microbenchmark(
  fixed = str_detect(sentences, fixed("the")),
  regex = str_detect(sentences, "the"),
  times = 20
  )
```

```
Unit: microseconds
```

expr <fctr>	min <dbl>	lq <dbl>	mean <dbl>	median <dbl>	uq <dbl>	max <dbl>	neval <dbl>
fixed	145.901	167.9010	240.9112	223.551	287.0005	550.902	20
regex	379.302	552.1015	594.2611	593.451	631.0510	1100.901	20

2 rows

Task-7:Starting with a1 being “0e1” and a2 being “a301”, both representing the character “á”, they are compared for equality.

Hide

```
a1 <- "\u00e1"
a2 <- "a\u0301"
c(a1, a2)
```

```
[1] "á" "á"
```

Hide

```
a1 == a2
```

```
[1] FALSE
```

Task-8: Checking if `a1` contains the fixed string `a2` returns `FALSE`, whereas using collation rules returns `TRUE`.

Hide

```
str_detect(a1, fixed(a2))
```

```
[1] FALSE
```

Hide

```
str_detect(a1, coll(a2))
```

```
[1] TRUE
```

Task-9:Creating a vector `i` with different forms of the letter “i”, then using `str_subset` to filter them based on collation.

Hide

```
i <- c("I", "İ", "i", "ı")
i
```

```
[1] "I" "İ" "i" "ı"
```

Hide

```
str_subset(i, coll("i", ignore_case = TRUE))
```

```
[1] "I" "i"
```

Hide

```
str_subset(i, coll("i", ignore_case = TRUE, locale = "tr"))
```

```
[1] "İ" "i"
```

Task-10: Fetching locale information.

Hide

```
stringi::stri_locale_info()
```

```
$Language
[1] "en"

$Country
[1] "US"

$Variant
[1] ""

$Name
[1] "en_US"
```

Task-11:Visualizing word boundaries and extracts all words from the string.

Hide

```
x <- "This is a sentence."
str_view_all(x, boundary("word"))
```

```
[1] | <This> <is> <a> <sentence>.
```

Hide

```
str_extract_all(x, boundary("word"))
```

```
[[1]]
[1] "This"     "is"       "a"        "sentence"
```

CH-15: Factors

Creatig factors

Task-1:Adding character vector in variable x1

Hide

```
x1 <- c("Dec", "Apr", "Jan", "Mar")
```

Task-2:Adding character vector in variable x2

Hide

```
x2 <- c("Dec", "Apr", "Jam", "Mar")
```

Task-3:Sorting X1

Hide

```
sort(x1)
```

```
[1] "Apr" "Dec" "Jan" "Mar"
```

Task-4:Adding Character vector in month_levels

Hide

```
month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)
```

Task-5:Assigning the factor levels to the variable x1, using the predefined month_levels.

Hide

```
y1 <- factor(x1, levels = month_levels)
y
```

key	val_y
<dbl>	<chr>
1	y1

key <dbl>	val_y <chr>
2	y2
2	y3
3	y4

4 rows

Task-6:Sorting the factor levels in y1.

Hide

```
sort(y1)
```

```
[1] Jan Mar Apr Dec
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Task-7:creating a factor y2 from x2 with custom levels specified by month_levels.

Hide

```
y2 <- factor(x2, levels = month_levels)
y2
```

```
[1] Dec  Apr  <NA> Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Task-8:parsing the values in x2 as factors

Hide

```
y2 <- parse_factor(x2, levels = month_levels)
```

```
Warning: 1 parsing failure.
row col           expected actual
  3  -- value in level set    Jam
```

Task-9: omitting the levels.

Hide

```
factor(x1)
```

```
[1] Dec Apr Jan Mar
Levels: Apr Dec Jan Mar
```

Task-10:Creating a factor f1 from the values in x1, using the unique values of x1 as levels.

Hide

```
f1 <- factor(x1, levels = unique(x1))
f1
```

```
[1] Dec Apr Jan Mar
Levels: Dec Apr Jan Mar
```

Task-11: creating a factor f2 from the values in x1, ordering them according to their appearance in x1.

Hide

```
f2 <- x1 %>% factor() %>% fct_inorder()
f2
```

```
[1] Dec Apr Jan Mar
Levels: Dec Apr Jan Mar
```

Task-12:Omitting levels2

Hide

```
levels(f2)
```

```
[1] "Dec" "Apr" "Jan" "Mar"
```

General Social Survey

Task-1:Loading datasets

Hide

```
gss_cat
```

y… <int>	marital <fctr>	a.. <int>	race <fctr>	rincome <fctr>	partyid <fctr>	relig <fctr>
2000	Never married	26	White	$8000 to 9999	Ind,near rep	Protestant
2000	Divorced	48	White	$8000 to 9999	Not str republican	Protestant
2000	Widowed	67	White	Not applicable	Independent	Protestant
2000	Never married	39	White	Not applicable	Ind,near rep	Orthodox-christian
2000	Divorced	25	White	Not applicable	Not str democrat	None
2000	Married	25	White	$20000 - 24999	Strong democrat	Protestant
2000	Never married	36	White	$25000 or more	Not str republican	Christian
2000	Divorced	44	White	$7000 to 7999	Ind,near dem	Protestant
2000	Married	44	White	$25000 or more	Not str democrat	Protestant
2000	Married	47	White	$25000 or more	Strong republican	Protestant

1-10 of 21,483 rows | 1-7 of 9 columns Previous **1** 2 3 4 5 6 … 100 Next

Task-2:Seeing levels through count()

Hide

```
gss_cat %>%
  count(race)
```

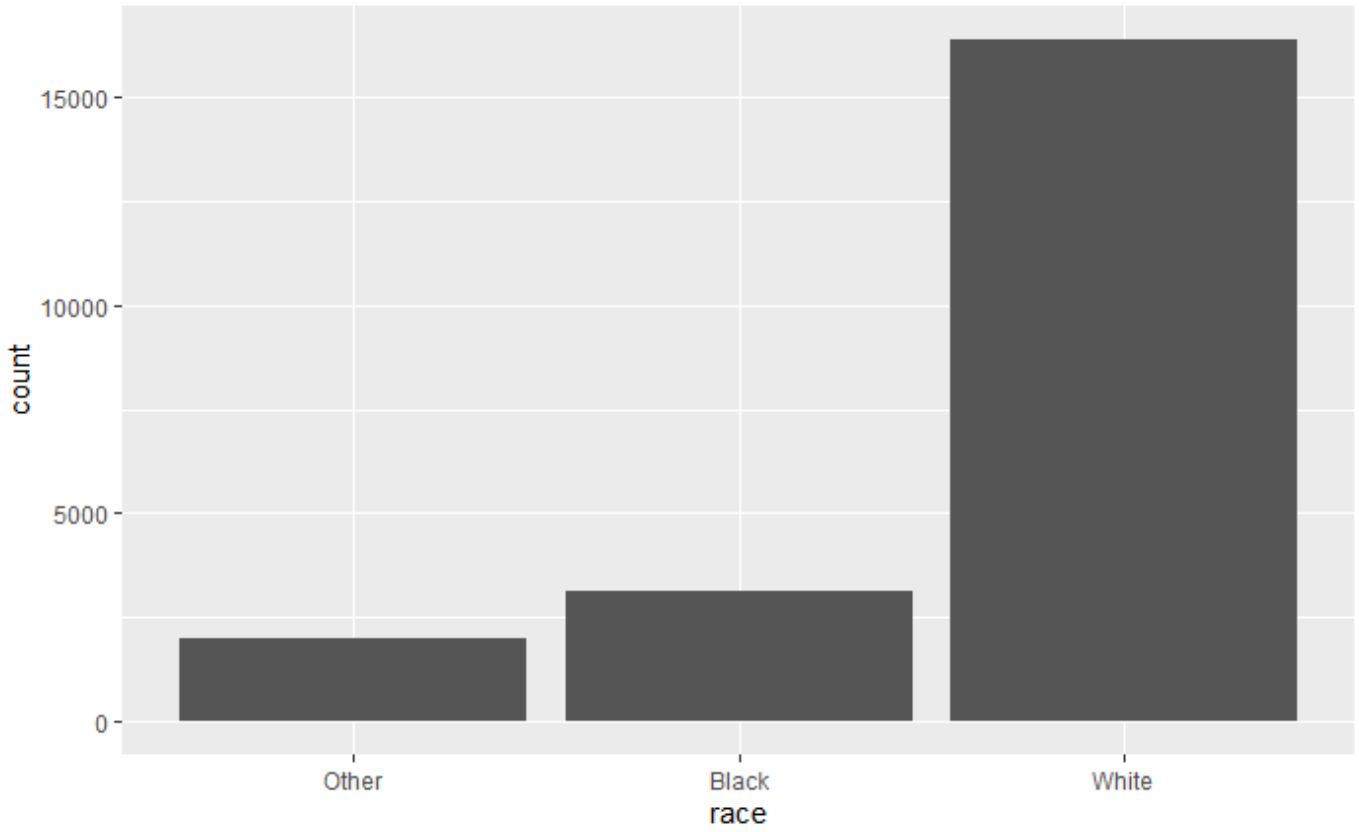
race <fctr>	n <int>
Other	1959
Black	3129
White	16395

3 rows

Task-3:Also seeing through bar()

Hide

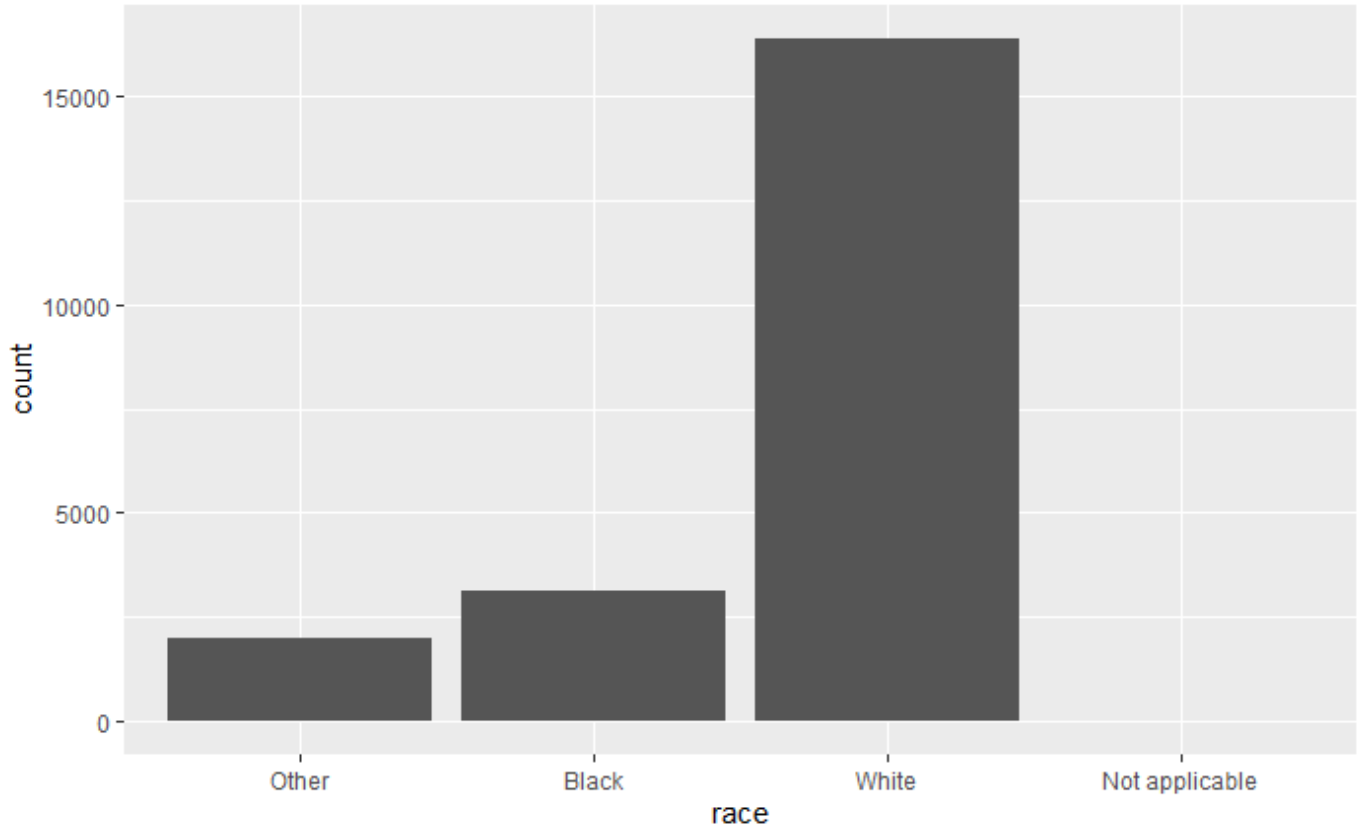
```
ggplot(gss_cat, aes(race)) +
  geom_bar()
```



Task-4:Generating a bar plot using ggplot()

Hide

```
ggplot(gss_cat,aes(race))+geom_bar()+scale_x_discrete(drop=FALSE)
```

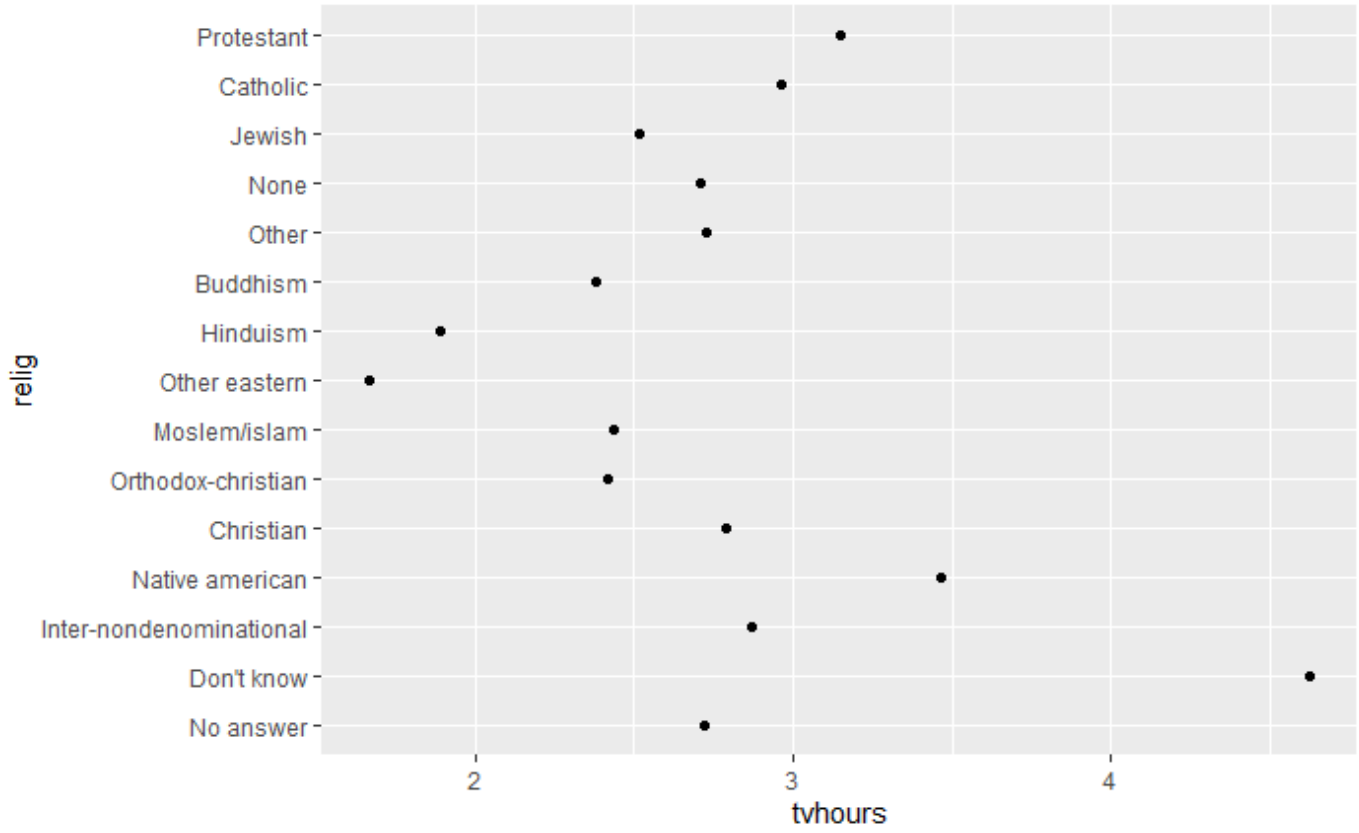


Modifying factor order

Task-1:calculating summary statistics and then creating scatter plot

Hide

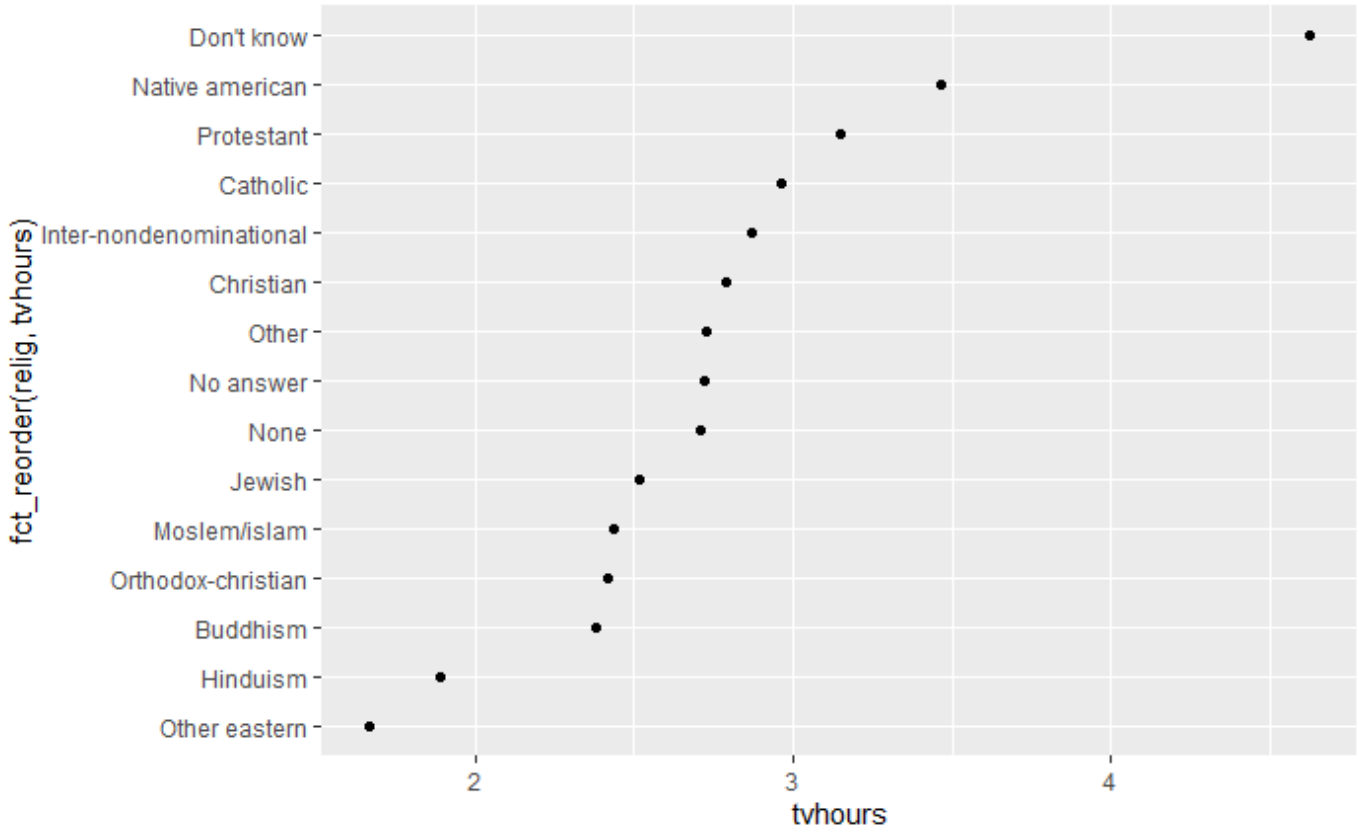
```
relig_summary <- gss_cat %>%
  group_by(relig) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE),
    n = n()
  )

ggplot(relig_summary, aes(tvhours, relig)) + geom_point()
```



Task-2:Generating a scatter plot using `ggplot` , where the x-axis represents the mean TV hours (`tvhours`), and the y-axis represents the `relig` variable reordered by mean TV hours.

Hide

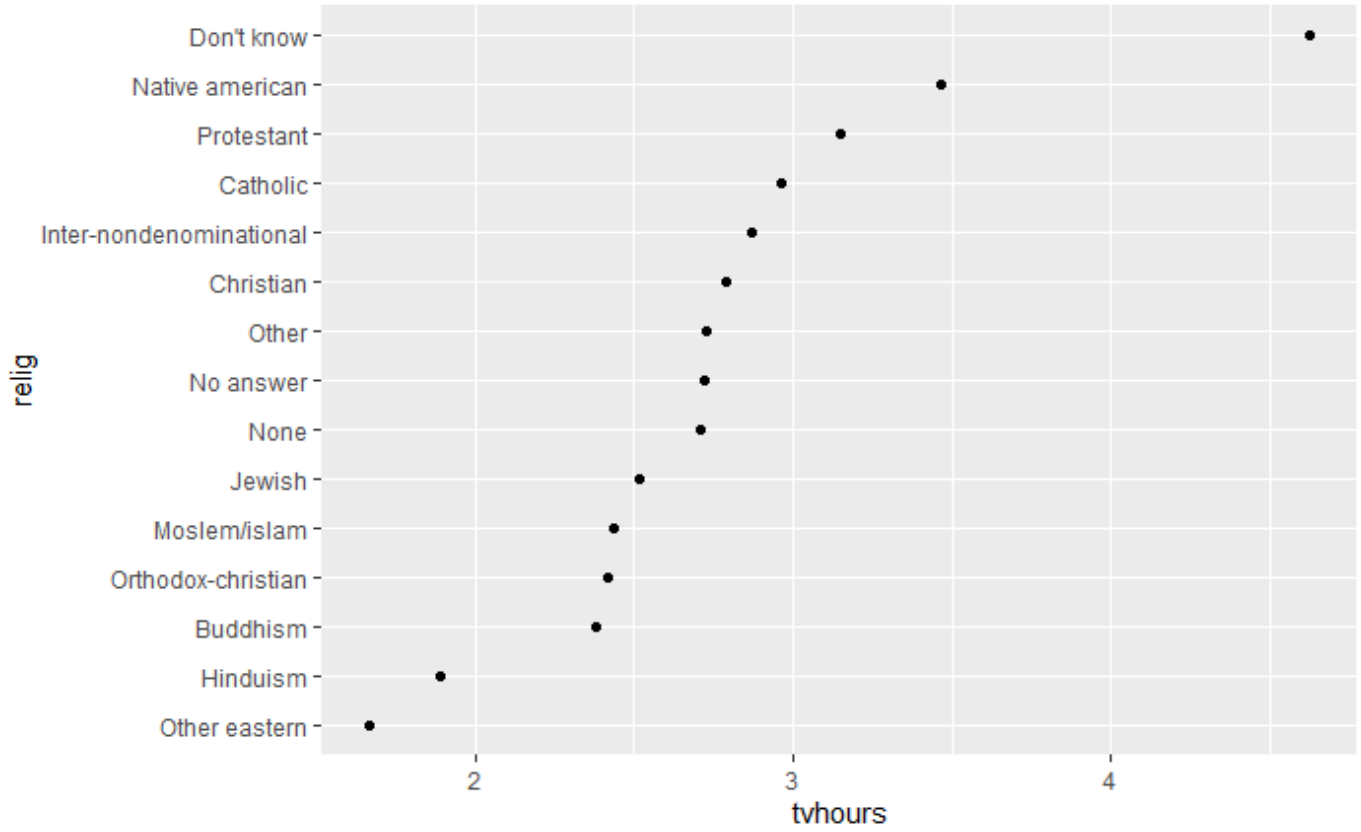
```
ggplot(relig_summary, aes(tvhours, fct_reorder(relig, tvhours))) +
  geom_point()
```



Task-3:Creating a scatter plot using ggplot.

Hide

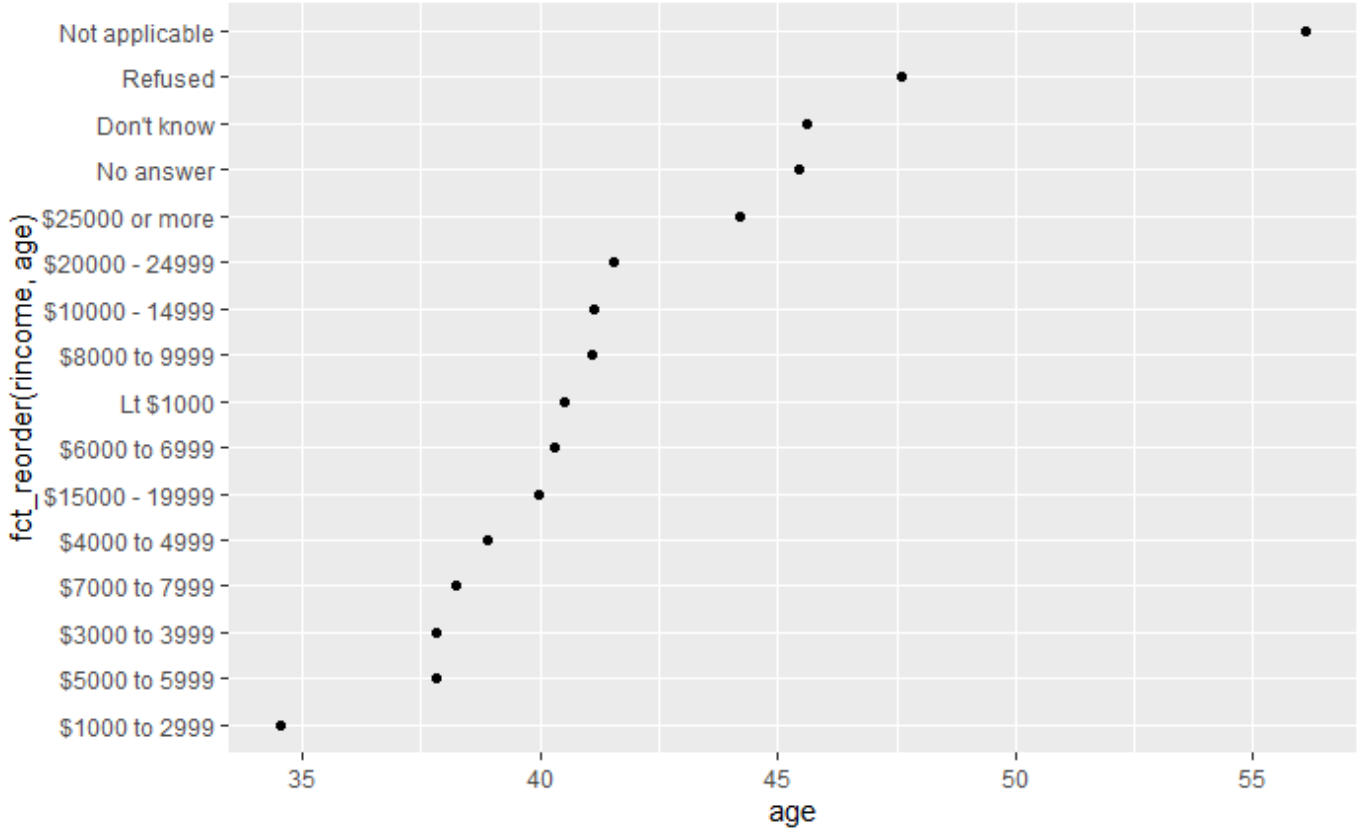
```
relig_summary %>%
  mutate(relig = fct_reorder(relig, tvhours)) %>%
  ggplot(aes(tvhours, relig)) +
    geom_point()
```



Task-4:Generating a scatter plot using ggplot

Hide

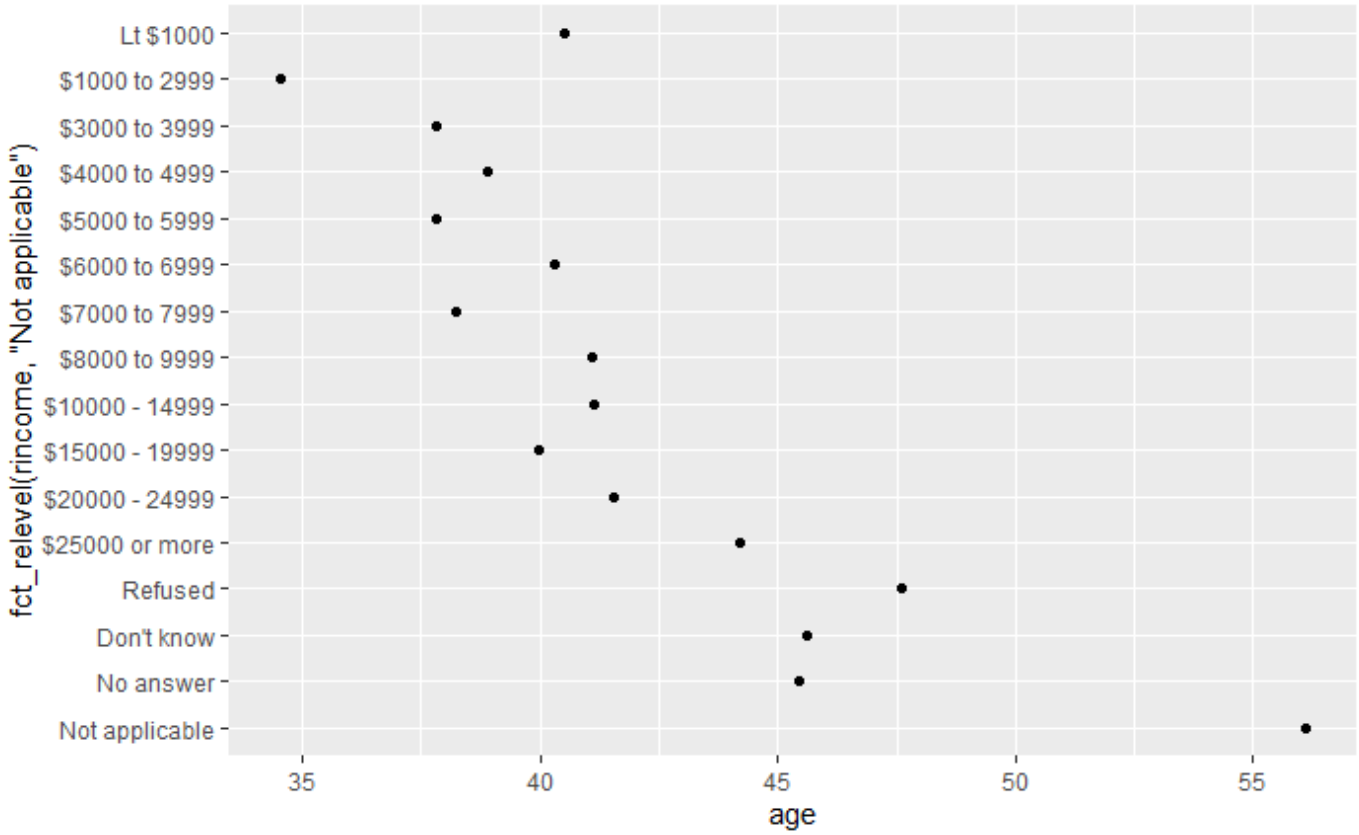
```
rincome_summary <- gss_cat %>%
  group_by(rincome) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE),
    n = n()
  )

ggplot(rincome_summary, aes(age, fct_reorder(rincome, age))) + geom_point()
```



Task-5: creates a scatter plot of the average age by income level, with “Not applicable” as the reference level for income

Hide

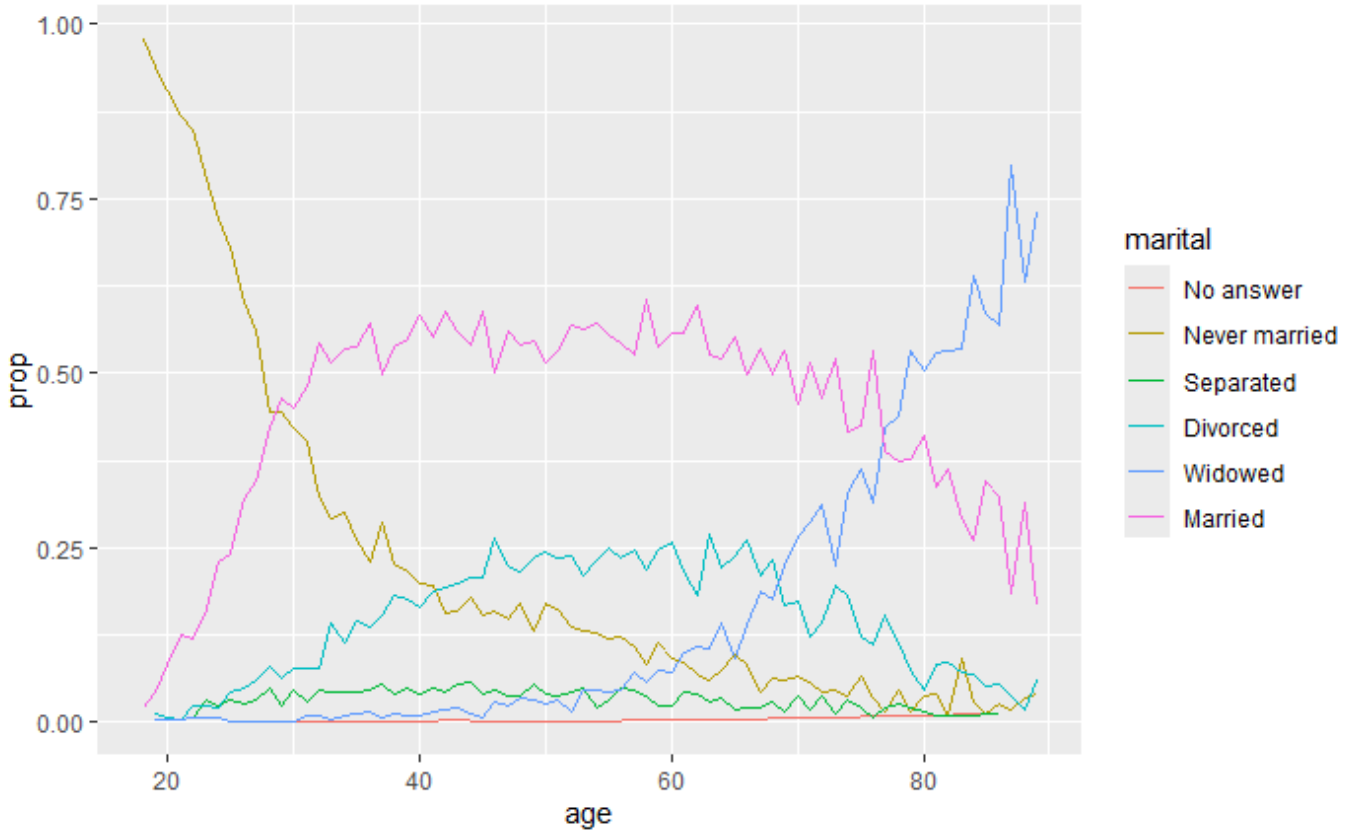
```
ggplot(rincome_summary, aes(age, fct_relevel(rincome, "Not applicable"))) +
  geom_point()
```



Task-6:calculating the proportion of each marital status group across different age groups and creates a line plot showing the distribution of marital status proportions by age.

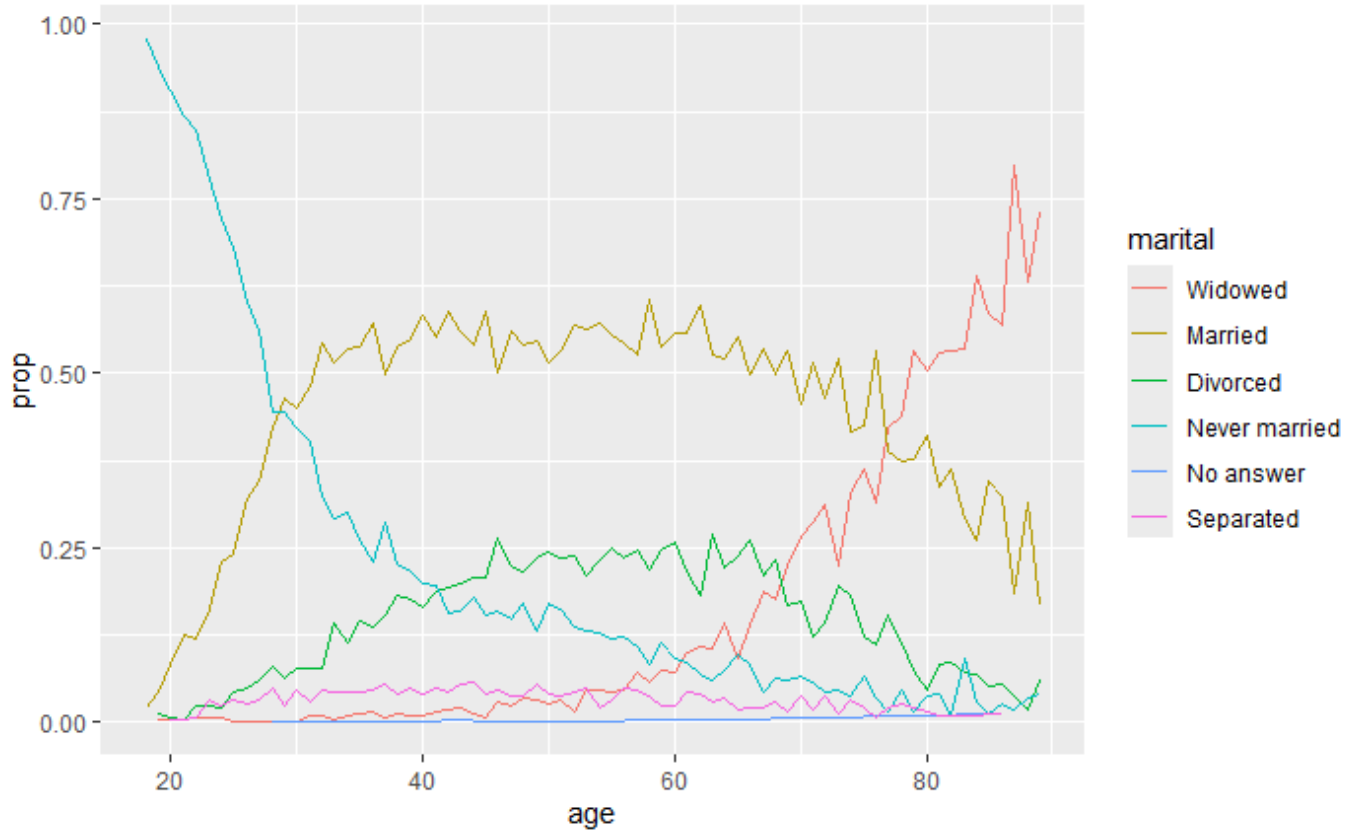
Hide

```
by_age <- gss_cat %>%
  filter(!is.na(age)) %>%
  count(age, marital) %>%
  group_by(age) %>%
  mutate(prop = n / sum(n))

ggplot(by_age, aes(age, prop, colour = marital)) +
  geom_line(na.rm = TRUE)
```



Hide

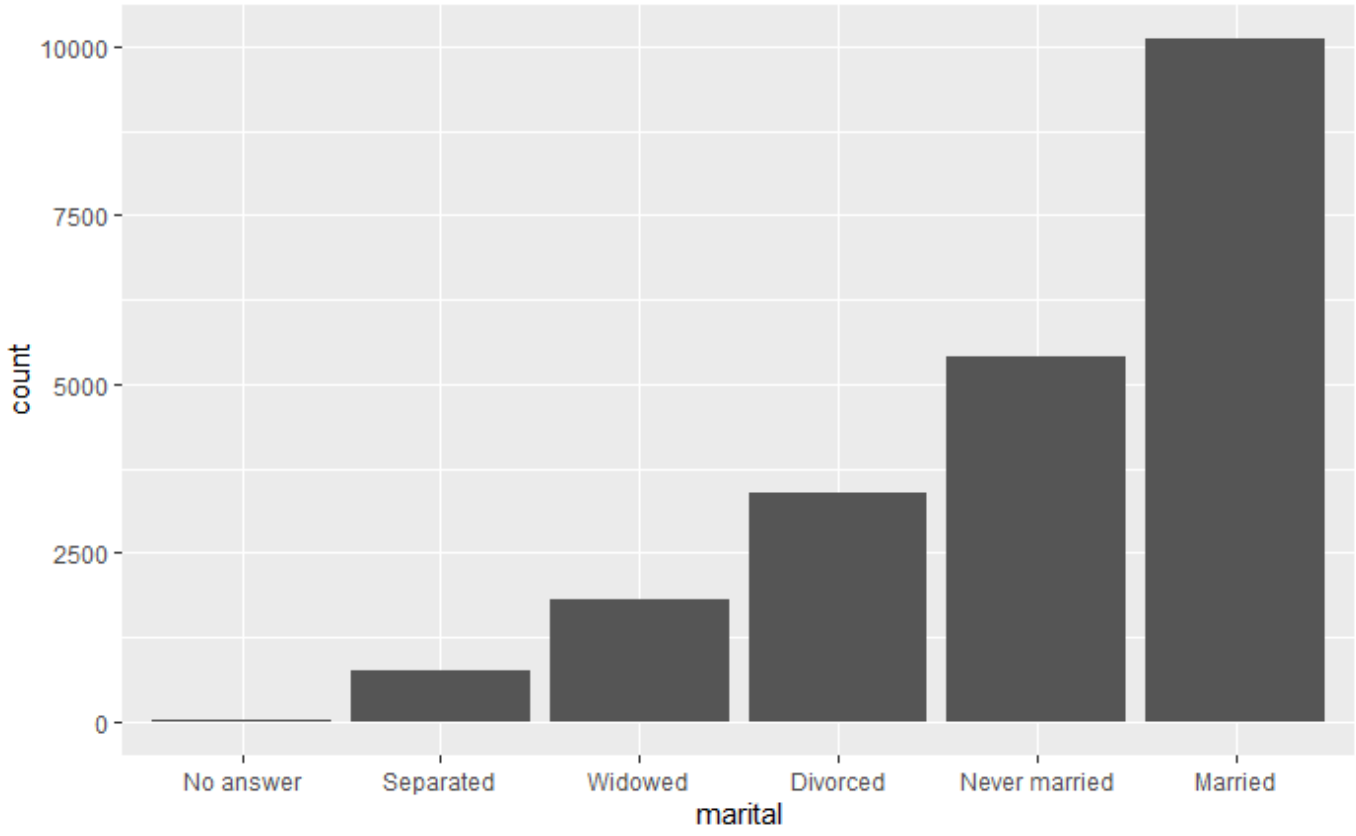
```
ggplot(by_age, aes(age, prop, colour = fct_reorder2(marital, age, prop))) +
  geom_line() +
  labs(colour = "marital")
```



Task-7: Adjusting the order of the “marital” variable based on frequency and then reverses the order before generating a bar plot illustrating the distribution of marital status.

Hide

```
gss_cat %>%
  mutate(marital = marital %>% fct_infreq() %>% fct_rev()) %>%
  ggplot(aes(marital)) +
    geom_bar()
```



Modifying factor levels

Task-1: counting the frequency of each unique value in the “partyid” variable of the “gss_cat” dataset.

Hide

```
gss_cat%>%count(partyid)
```

partyid <fctr>	**n** <int>
No answer	154
Don't know	1
Other party	393
Strong republican	2314
Not str republican	3032
Ind,near rep	1791
Independent	4119
Ind,near dem	2499
Not str democrat	3690
Strong democrat	3490

1-10 of 10 rows

Task-2:Recording the levels of the “partyid” variable in the “gss_cat” dataset and then counts the frequency of each unique recorded value.

Hide

```
gss_cat %>%
  mutate( partyid=fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat"
    ))%>%
  count(partyid)
```

partyid <fctr>	**n** <int>
No answer	154
Don't know	1
Other party	393
Republican, strong	2314

partyid <fctr>	n <int>
Republican, weak	3032
Independent, near rep	1791
Independent	4119
Independent, near dem	2499
Democrat, weak	3690
Democrat, strong	3490

1-10 of 10 rows

Task-3:Recategorizing and counting party affiliations in the “gss_cat” dataset.

Hide

```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat",
    "Other"                 = "No answer",
    "Other"                 = "Don't know",
    "Other"                 = "Other party"
  )) %>%
  count(partyid)
```

partyid <fctr>	n <int>
Other	548
Republican, strong	2314
Republican, weak	3032
Independent, near rep	1791
Independent	4119
Independent, near dem	2499
Democrat, weak	3690
Democrat, strong	3490

8 rows

Task-4: Collapsing categories within the “partyid” variable in the “gss_cat” dataset into broader groups and then counting the frequency of each collapsed category.

Hide

```
gss_cat%>%
  mutate(partyid=fct_collapse(partyid,
                              other=c("No answer", "Don't know", "Other party"),
                              rep=c("Strong republican", "Not str republican"),
                              ind=c("Ind,near rep", "Independent", "Ind,near dem"),
                              dem=c("Not str democrat", "Strong democrat"))) %>%
  count(partyid)
```

partyid <fctr>	**n** <int>
other	548
rep	5346
ind	8409
dem	7180

4 rows

Task-5:Counting and aggregating religious affiliations in the “gss_cat” dataset after lumping together less frequent categories.

Hide

```
gss_cat %>%
  mutate(relig = fct_lump(relig)) %>%
  count(relig)
```

relig <fctr>	**n** <int>
Protestant	10846
Other	10637

2 rows

Task-6:“Summarizing religious affiliations after lumping infrequent categories and sort.”

Hide

```
gss_cat %>%
  mutate(relig = fct_lump(relig, n = 10)) %>%
  count(relig, sort = TRUE) %>%
  print(n = Inf)
```

relig <fctr>	**n** <int>
Protestant	10846
Catholic	5124
None	3523
Christian	689

relig <fctr>	n <int>
Other	458
Jewish	388
Buddhism	147
Inter-nondenominational	109
Moslem/islam	104
Orthodox-christian	95

1-10 of 10 rows

CH-Data and Times

Task-1:Loading library

Hide

```
library(tidyverse)

library(lubridate)
library(nycflights13)
```

Creating dates/times

Task-1: Printing current date or date-time

Hide

```
today()
```

```
[1] "2024-05-04"
```

Hide

```
now()
```

```
[1] "2024-05-04 20:51:16 +0545"
```

Form strings

Task-2:converting date strings to date objects in different formats.

Hide

```
ymd("2017-01-31")
```

```
[1] "2017-01-31"
```

Hide

```
mdy("January 31st, 2017")
```

```
[1] "2017-01-31"
```

Hide

```
dmy("31-Jan-2017")
```

```
[1] "2017-01-31"
```

Hide

```
ymd(20170131)
```

```
[1] "2017-01-31"
```

Hide

```
ymd_hms("2017-01-31 20:11:59")
```

```
[1] "2017-01-31 20:11:59 UTC"
```

Hide

```
mdy_hm("01/31/2017 08:01")
```

```
[1] "2017-01-31 08:01:00 UTC"
```

Hide

```
flights %>%
  select(year, month, day, hour, minute)
```

year <int>	month <int>	day <int>	hour <dbl>	minute <dbl>
2013	1	1	5	15
2013	1	1	5	29
2013	1	1	5	40
2013	1	1	5	45
2013	1	1	6	0
2013	1	1	5	58
2013	1	1	6	0
2013	1	1	6	0

year <int>	month <int>	day <int>	hour <dbl>	minute <dbl>
2013	1	1	6	0
2013	1	1	6	0

1-10 of 336,776 rows Previous **1** 2 3 4 5 6 … 100 Next

Hide

```
flights %>%
  select(year, month, day, hour, minute) %>%
  mutate(departure = make_datetime(year, month, day, hour, minute))
```

year <int>	month <int>	day <int>	hour <dbl>	minute <dbl>	departure <S3: POSIXct>
2013	1	1	5	15	2013-01-01 05:15:00
2013	1	1	5	29	2013-01-01 05:29:00
2013	1	1	5	40	2013-01-01 05:40:00
2013	1	1	5	45	2013-01-01 05:45:00
2013	1	1	6	0	2013-01-01 06:00:00
2013	1	1	5	58	2013-01-01 05:58:00
2013	1	1	6	0	2013-01-01 06:00:00
2013	1	1	6	0	2013-01-01 06:00:00
2013	1	1	6	0	2013-01-01 06:00:00
2013	1	1	6	0	2013-01-01 06:00:00

1-10 of 336,776 rows Previous **1** 2 3 4 5 6 … 100 Next

Task: Creating date-time objects from hour-minute time data in the ‘flights’ dataset and filtering out rows with missing departure or arrival times

Hide

```
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
}

flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(
    dep_time = make_datetime_100(year, month, day, dep_time),
    arr_time = make_datetime_100(year, month, day, arr_time),
    sched_dep_time = make_datetime_100(year, month, day, sched_dep_time),
    sched_arr_time = make_datetime_100(year, month, day, sched_arr_time)
  ) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))

flights_dt
```

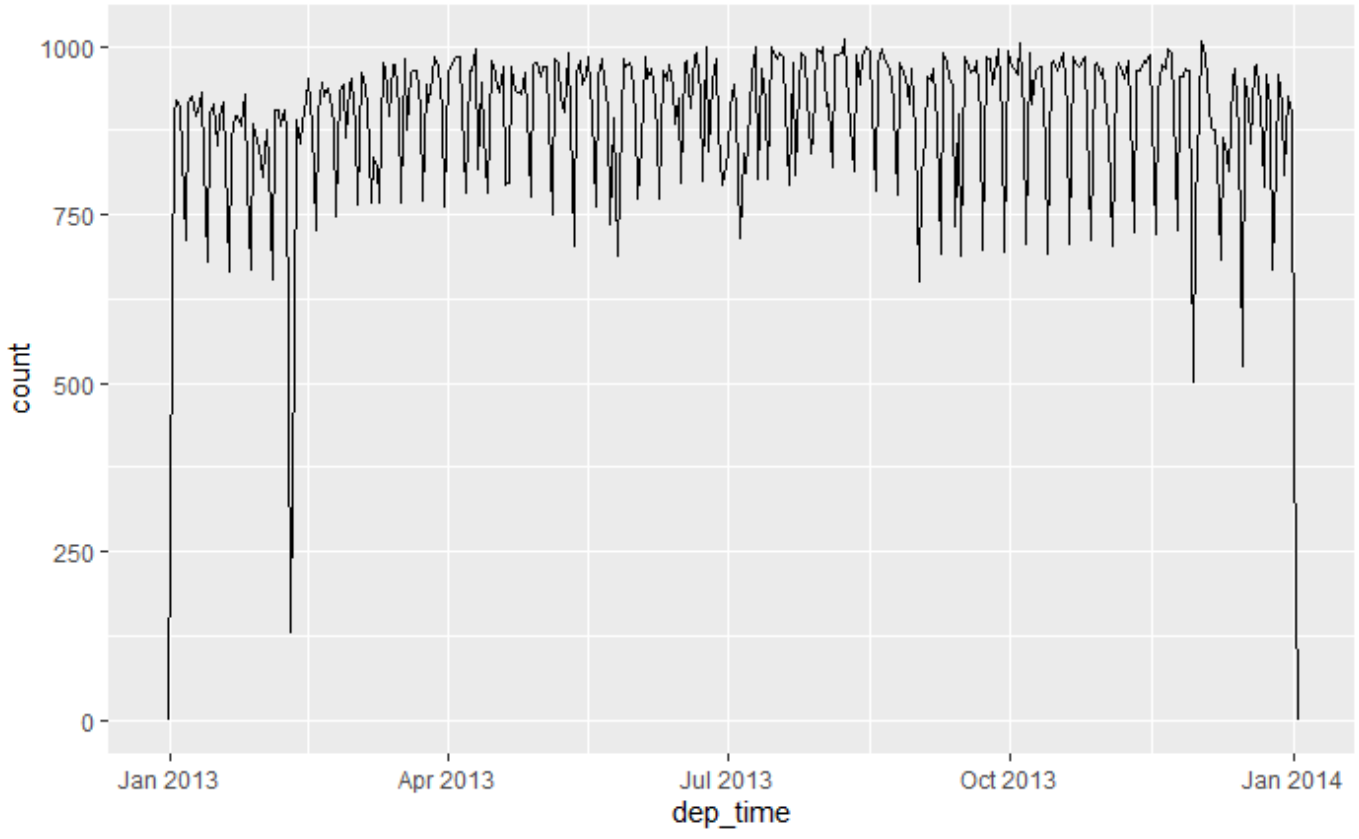
origin <chr>	d… <chr>	dep_delay <dbl>	arr_delay <dbl>	dep_time <S3: POSIXct>	sched_dep_time <S3: POSIXct>	<S
EWR	IAH	2	11	2013-01-01 05:17:00	2013-01-01 05:15:00	2013-01
LGA	IAH	4	20	2013-01-01 05:33:00	2013-01-01 05:29:00	2013-01
JFK	MIA	2	33	2013-01-01 05:42:00	2013-01-01 05:40:00	2013-01
JFK	BQN	-1	-18	2013-01-01 05:44:00	2013-01-01 05:45:00	2013-01
LGA	ATL	-6	-25	2013-01-01 05:54:00	2013-01-01 06:00:00	2013-01
EWR	ORD	-4	12	2013-01-01 05:54:00	2013-01-01 05:58:00	2013-01
EWR	FLL	-5	19	2013-01-01 05:55:00	2013-01-01 06:00:00	2013-01
LGA	IAD	-3	-14	2013-01-01 05:57:00	2013-01-01 06:00:00	2013-01
JFK	MCO	-3	-8	2013-01-01 05:57:00	2013-01-01 06:00:00	2013-01
LGA	ORD	-2	8	2013-01-01 05:58:00	2013-01-01 06:00:00	2013-01

1-10 of 328,063 rows | 1-7 of 9 columns Previous **1** 2 3 4 5 6 … 100 Next

Task: Plotting the frequency of flights over time using departure date-time

Hide

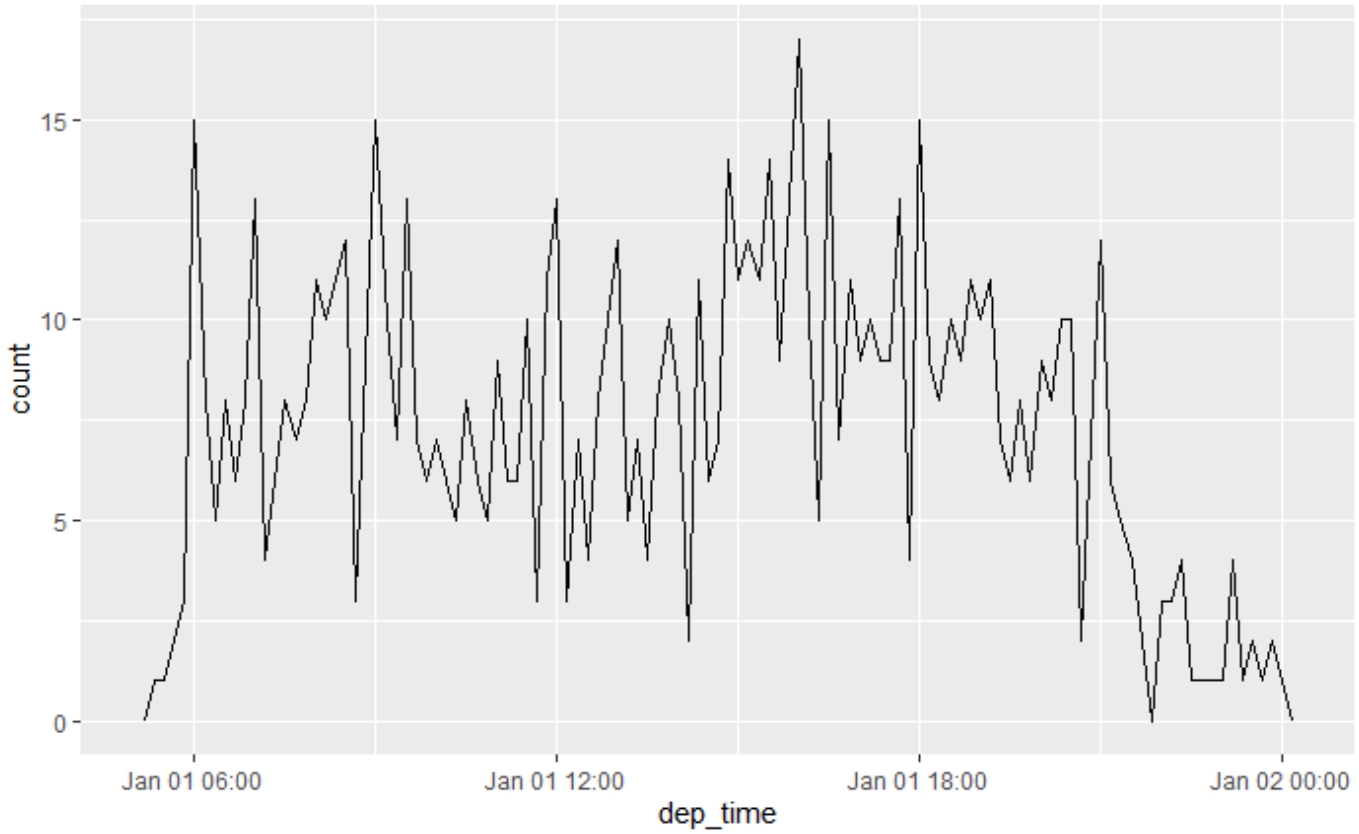
```
flights_dt %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 86400)
```



Task: Plotting the frequency of flights over time for a specific period using departure date-time

Hide

```
flights_dt %>%
  filter(dep_time < ymd(20130102)) %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 600) # 600 s = 10 minutes
```



Task: to convert today's date to date-time object

Hide

```
as_datetime(today())
```

```
[1] "2024-05-04 UTC"
```

Hide

```
as_date(now())
```

```
[1] "2024-05-04"
```

Hide

```
as_date(365 * 10 + 2)
```

```
[1] "1980-01-01"
```

Date-time components Task: Extracting various components of a date-time object

Hide

```
datetime <- ymd_hms("2016-07-08 12:34:56")
year(datetime)
```

```
[1] 2016
```

Hide

```
month(datetime)
```

```
[1] 7
```

Hide

```
mday(datetime)
```

```
[1] 8
```

Hide

```
yday(datetime)
```

```
[1] 190
```

Hide

```
wday(datetime)
```

```
[1] 6
```

Hide

```
month(datetime, label = TRUE)
```

```
[1] Jul
Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < Oct < Nov < Dec
```

Hide

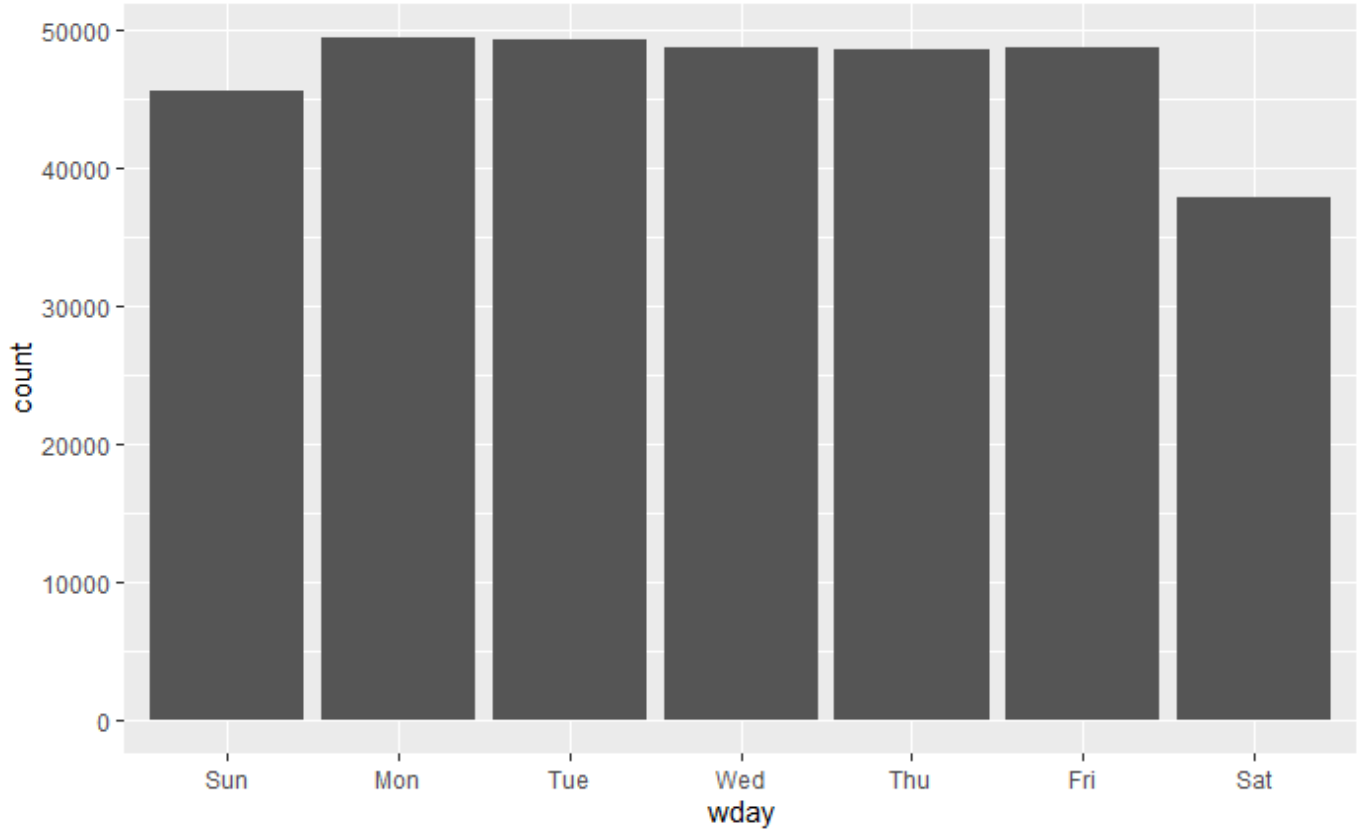
```
wday(datetime, label = TRUE, abbr = FALSE)
```

```
[1] Friday
Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < Friday < Saturday
```

Task: Plotting the frequency of flights by day of the week

Hide

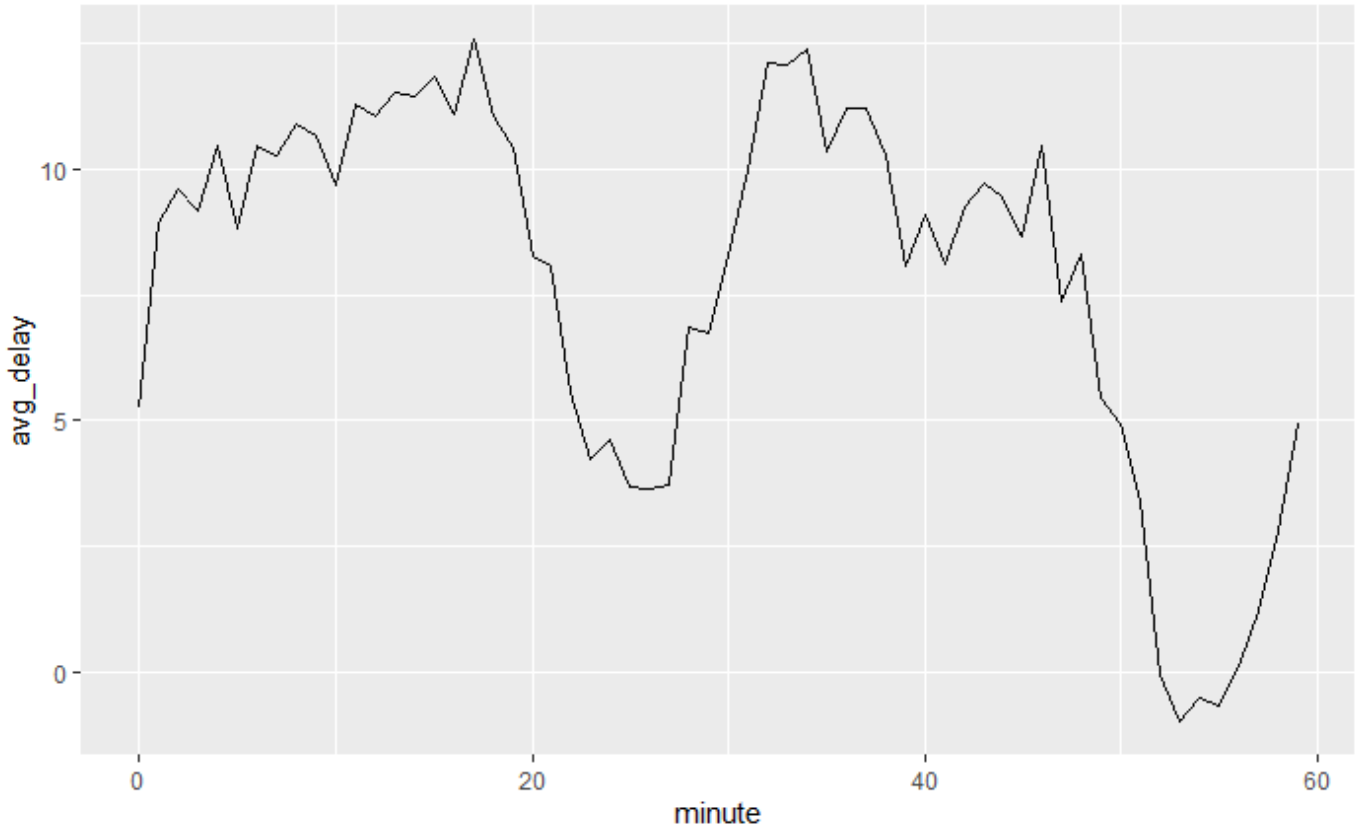
```
flights_dt %>%
  mutate(wday = wday(dep_time, label = TRUE)) %>%
  ggplot(aes(x = wday)) +
    geom_bar()
```



Task: Plotting average delay by minute of departure time

Hide

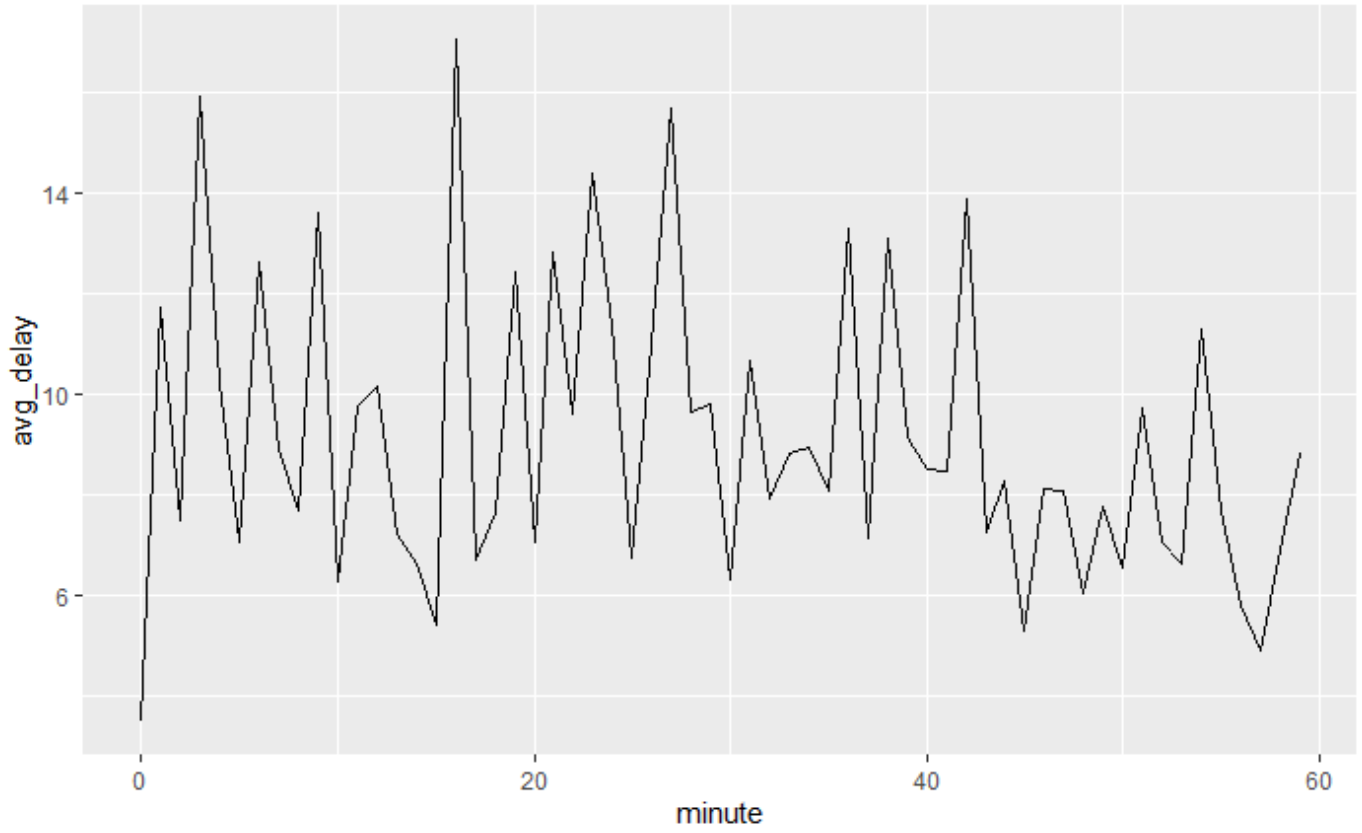
```
flights_dt %>%
  mutate(minute = minute(dep_time)) %>%
  group_by(minute) %>%
  summarise(
    avg_delay = mean(arr_delay, na.rm = TRUE),
    n = n()) %>%
  ggplot(aes(minute, avg_delay)) +
    geom_line()
```



Task: Plotting average delay by minute of scheduled departure time

Hide

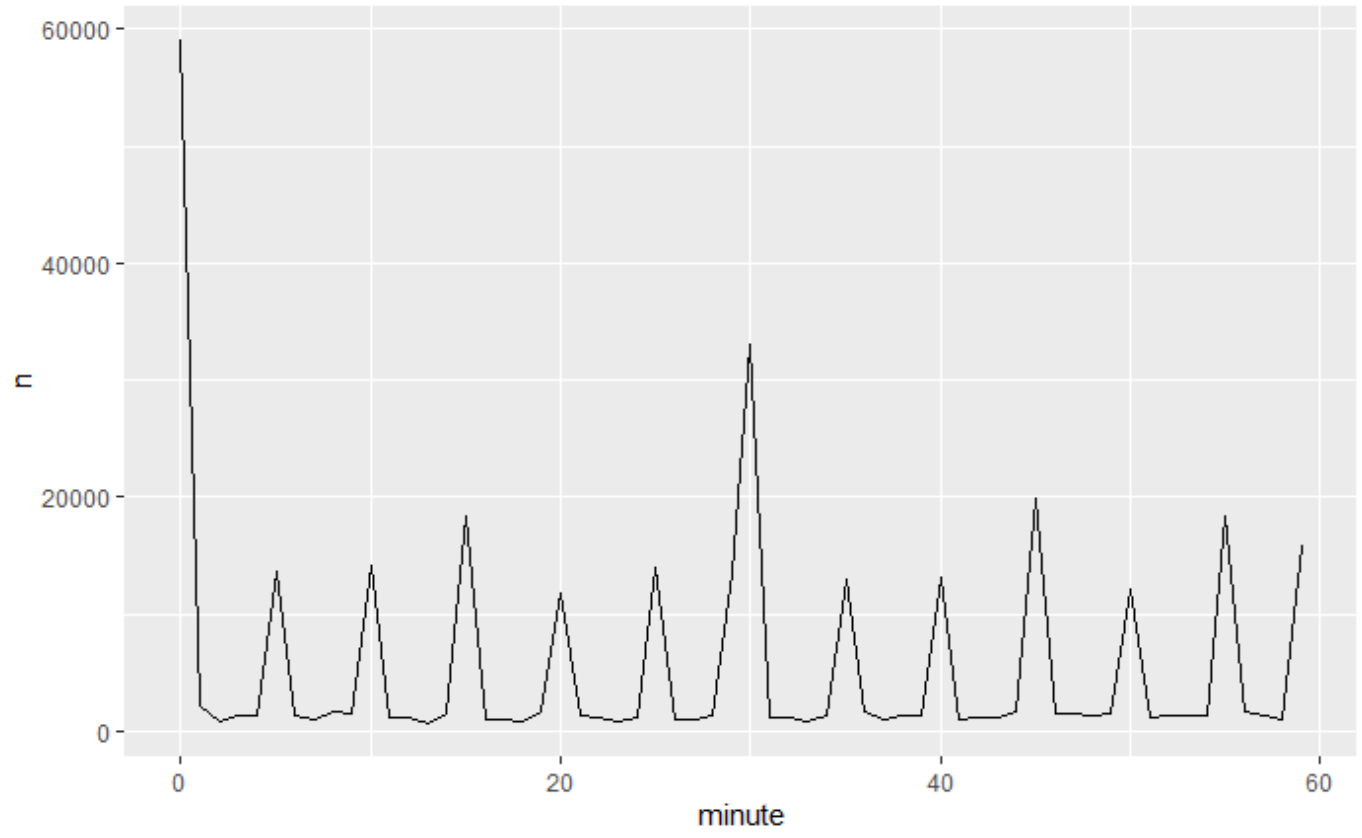
```
sched_dep <- flights_dt %>%
  mutate(minute = minute(sched_dep_time)) %>%
  group_by(minute) %>%
  summarise(
    avg_delay = mean(arr_delay, na.rm = TRUE),
    n = n())

ggplot(sched_dep, aes(minute, avg_delay)) +
  geom_line()
```



Task: Plotting the number of flights by minute of scheduled departure time

Hide

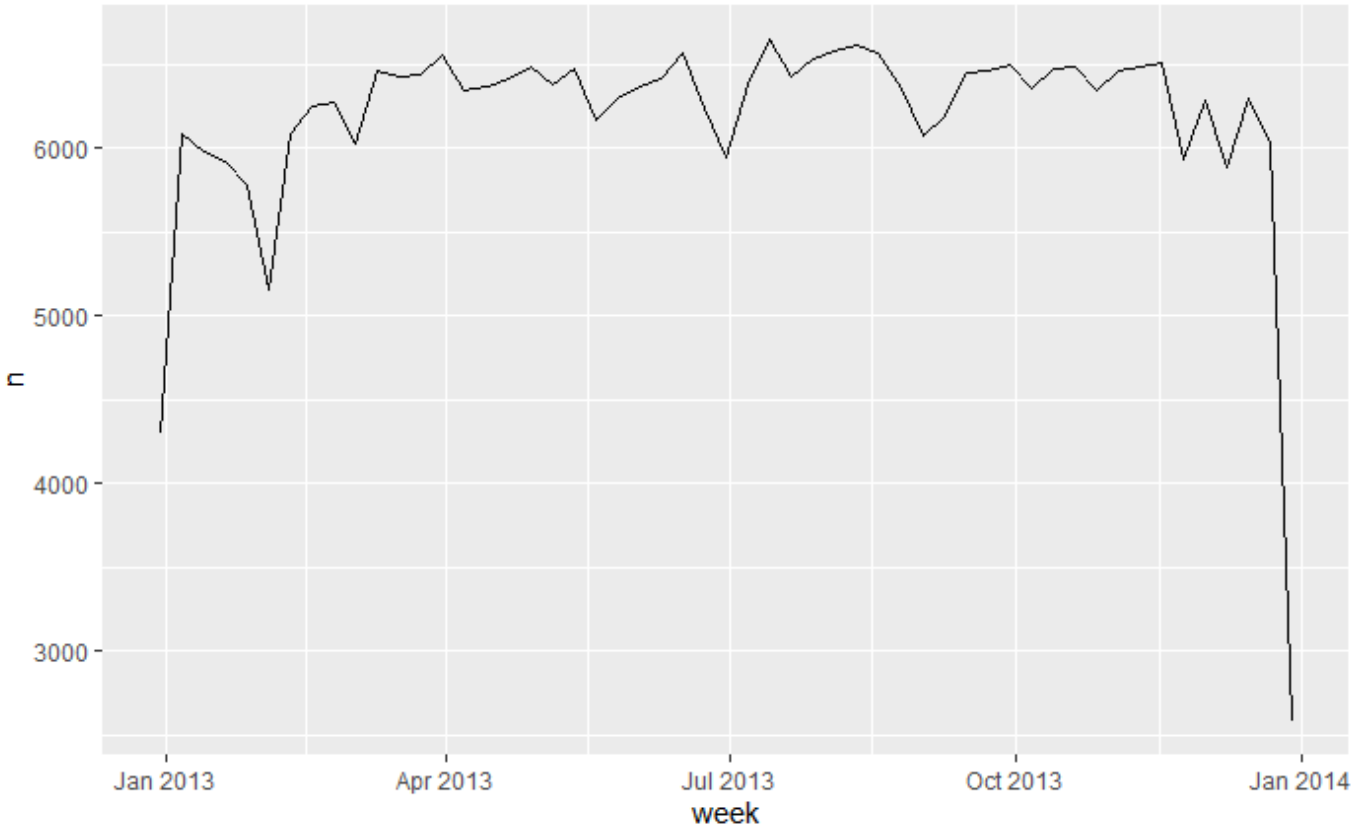
```
ggplot(sched_dep, aes(minute, n)) +
  geom_line()
```



Rounding Task:Plotting the number of flights by week, rounding to the nearest week

Hide

```
flights_dt %>%
  count(week = floor_date(dep_time, "week")) %>%
  ggplot(aes(week, n)) +
    geom_line()
```



setting compounds Task: Setting up a date-time object

Hide

```
(datetime <- ymd_hms("2016-07-08 12:34:56"))
```

```
[1] "2016-07-08 12:34:56 UTC"
```

Hide

```
year(datetime) <- 2020
datetime
```

```
[1] "2020-07-08 12:34:56 UTC"
```

Hide

```
month(datetime) <- 01
datetime
```

```
[1] "2020-01-08 12:34:56 UTC"
```

Hide

```
hour(datetime) <- hour(datetime) + 1
datetime
```

```
[1] "2020-01-08 13:34:56 UTC"
```

Hide

```
update(datetime, year = 2020, month = 2, mday = 2, hour = 2)
```

```
[1] "2020-02-02 02:34:56 UTC"
```

Hide

```
ymd("2015-02-01") %>%
  update(mday = 30)
```

```
[1] "2015-03-02"
```

Hide

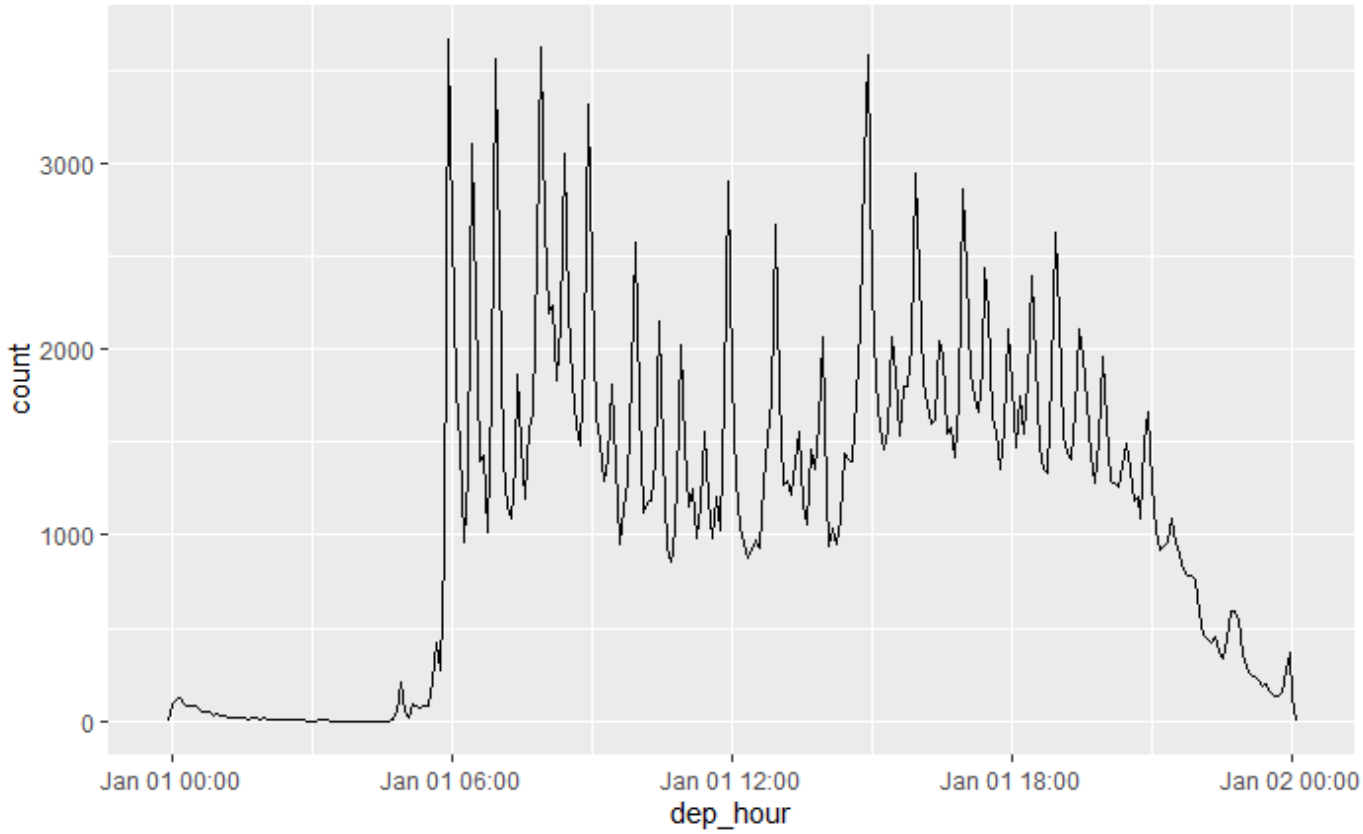
```
ymd("2015-02-01") %>%
  update(hour = 400)
```

```
[1] "2015-02-17 16:00:00 UTC"
```

Task: Creating a new variable ‘dep_hour’ by updating the ‘dep_time’ to the first day of the year

Hide

```
flights_dt %>%
  mutate(dep_hour = update(dep_time, yday = 1)) %>%
  ggplot(aes(dep_hour)) +
    geom_freqpoly(binwidth = 300)
```



Time Spans Compute the age of a person based on their birthdate and today's date

Hide

```
h_age <- today() - ymd(19791014)
h_age
```

```
Time difference of 16274 days
```

Hide

```
as.duration(h_age)
```

```
[1] "1406073600s (~44.56 years)"
```

Hide

```
dseconds(15)
```

```
[1] "15s"
```

Hide

```
dminutes(10)
```

```
[1] "600s (~10 minutes)"
```

Hide

```
dhours(c(12, 24))
```

```
[1] "43200s (~12 hours)" "86400s (~1 days)"
```

Hide

```
ddays(0:5)
```

```
[1] "0s"                "86400s (~1 days)"  "172800s (~2 days)" "259200s (~3 days)" "345600s
(~4 days)"
[6] "432000s (~5 days)"
```

Hide

```
dweeks(3)
```

```
[1] "1814400s (~3 weeks)"
```

Hide

```
dyears(1)
```

```
[1] "31557600s (~1 years)"
```

Hide

```
2 * dyears(1)
```

```
[1] "63115200s (~2 years)"
```

Hide

```
dyears(1) + dweeks(12) + dhours(15)
```

```
[1] "38869200s (~1.23 years)"
```

Hide

```
tomorrow <- today() + ddays(1)
last_year <- today() - dyears(1)
one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
one_pm
```

```
[1] "2016-03-12 13:00:00 EST"
```

Hide

```
one_pm + ddays(1)
```

```
[1] "2016-03-13 14:00:00 EDT"
```

Periods Create period objects representing different time spans and Perform arithmetic operations with period objects

Hide

```
one_pm
```

```
[1] "2016-03-12 13:00:00 EST"
```

Hide

```
one_om = days(1)
```

Hide

```
seconds(15)
```

```
[1] "15S"
```

Hide

```
minutes(10)
```

```
[1] "10M 0S"
```

Hide

```
hours(c(12, 24))
```

```
[1] "12H 0M 0S" "24H 0M 0S"
```

Hide

```
days(7)
```

```
[1] "7d 0H 0M 0S"
```

Hide

```
months(1:6)
```

```
[1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S" "4m 0d 0H 0M 0S" "5m 0d 0H 0M 0S" "6m
0d 0H 0M 0S"
```

Hide

```
weeks(3)
```

```
[1] "21d 0H 0M 0S"
```

Hide

```
years(1)
```

```
[1] "1y 0m 0d 0H 0M 0S"
```

Hide

```
10 * (months(6) + days(1))
```

```
[1] "60m 10d 0H 0M 0S"
```

Hide

```
days(50) + hours(25) + minutes(2)
```

```
[1] "50d 25H 2M 0S"
```

Hide

```
ymd("2016-01-01") + dyears(1)
```

```
[1] "2016-12-31 06:00:00 UTC"
```

Hide

```
ymd("2016-01-01") + years(1)
```

```
[1] "2017-01-01"
```

Hide

```
one_pm + ddays(1)
```

```
[1] "2016-03-13 14:00:00 EDT"
```

Hide

```
one_pm + days(1)
```

```
[1] "2016-03-13 13:00:00 EDT"
```

Filter flights where arrival time is before departure time

Hide

```
flights_dt %>%
  filter(arr_time < dep_time)
```

origin <chr>	d… <chr>	dep_delay <dbl>	arr_delay <dbl>	dep_time <S3: POSIXct>	sched_dep_time <S3: POSIXct>	<S
EWR	BQN	9	-4	2013-01-01 19:29:00	2013-01-01 19:20:00	2013-01
JFK	DFW	59	*NA*	2013-01-01 19:39:00	2013-01-01 18:40:00	2013-01
EWR	TPA	-2	9	2013-01-01 20:58:00	2013-01-01 21:00:00	2013-01
EWR	SJU	-6	-12	2013-01-01 21:02:00	2013-01-01 21:08:00	2013-01
EWR	SFO	11	-14	2013-01-01 21:08:00	2013-01-01 20:57:00	2013-01
LGA	FLL	-10	-2	2013-01-01 21:20:00	2013-01-01 21:30:00	2013-01
EWR	MCO	41	43	2013-01-01 21:21:00	2013-01-01 20:40:00	2013-01
JFK	LAX	-7	-24	2013-01-01 21:28:00	2013-01-01 21:35:00	2013-01
EWR	FLL	49	28	2013-01-01 21:34:00	2013-01-01 20:45:00	2013-01
EWR	FLL	-9	-14	2013-01-01 21:36:00	2013-01-01 21:45:00	2013-01

1-10 of 10,633 rows | 1-7 of 9 columns Previous **1** 2 3 4 5 6 … 100 Next

Update flights data to correct overnight flights

Hide

```
flights_dt <- flights_dt %>%
  mutate(
    overnight = arr_time < dep_time,
    arr_time = arr_time + days(overnight * 1),
    sched_arr_time = sched_arr_time + days(overnight * 1)
  )
```

Filter flights where overnight condition is true and arrival time is before departure time

Hide

```
flights_dt %>%
  filter(overnight, arr_time < dep_time)
```

0 rows | 1-9 of 10 columns

Intervals Calculate the ratio of one year in days

Hide

```
years(1) / days(1)
```

```
[1] 365.25
```

Hide

```
next_year <- today() + years(1)
(today() %--% next_year) / ddays(1)
```

```
[1] 365
```

Hide

```
(today() %--% next_year) %/% days(1)
```

```
[1] 365
```

Display time zone information

Hide

```
Sys.timezone()
```

```
[1] "Asia/Katmandu"
```

Hide

```
length(OlsonNames())
```

```
[1] 596
```

Hide

```
head(OlsonNames())
```

```
[1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa" "Africa/Algiers"     "Afri
ca/Asmara"
[6] "Africa/Asmera"
```

Hide

```
(x1 <- ymd_hms("2015-06-01 12:00:00", tz = "America/New_York"))
```

```
[1] "2015-06-01 12:00:00 EDT"
```

Hide

```
(x2 <- ymd_hms("2015-06-01 18:00:00", tz = "Europe/Copenhagen"))
```

```
[1] "2015-06-01 18:00:00 CEST"
```

Hide

```
(x3 <- ymd_hms("2015-06-02 04:00:00", tz = "Pacific/Auckland"))
```

```
[1] "2015-06-02 04:00:00 NZST"
```

Hide

```
x1 - x2
```

```
Time difference of 0 secs
```

Hide

```
x1 - x3
```

```
Time difference of 0 secs
```

Pipes

Task: To import the required library

Hide

```
packages_to_install <- c("tidyverse", "pryr")
for (package_name in packages_to_install) {
  if (!requireNamespace(package_name, quietly = TRUE)) {
    install.packages(package_name)
  }
  library(package_name, character.only = TRUE)
}

library(magrittr)
```

Create diamond data and calculate the object sizes

Hide

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

pryr::object_size(diamonds)
```

```
3.46 MB
```

Hide

```
pryr::object_size(diamonds2)
```

```
3.89 MB
```

Hide

```
pryr::object_size(diamonds, diamonds2)
```

```
3.89 MB
```

Functions Normalize the columns of a data frame

Hide

```
df <- tibble::tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

df$a <- (df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b <- (df$b - min(df$b, na.rm = TRUE)) /
  (max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$c <- (df$c - min(df$c, na.rm = TRUE)) /
  (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d <- (df$d - min(df$d, na.rm = TRUE)) /
  (max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

Normalize a single column of a data frame

Hide

```
(df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
```

```
 [1] 0.2660918 0.1288832 0.0769690 0.3163641 0.5612945 0.6241704 0.5271891 0.0000000 0.391336
9 1.0000000
```

Hide

```
x <- df$a
(x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
```

```
 [1] 0.2660918 0.1288832 0.0769690 0.3163641 0.5612945 0.6241704 0.5271891 0.0000000 0.391336
9 1.0000000
```

Hide

```
rng <- range(x, na.rm = TRUE)
(x - rng[1]) / (rng[2] - rng[1])
```

```
 [1] 0.2660918 0.1288832 0.0769690 0.3163641 0.5612945 0.6241704 0.5271891 0.0000000 0.391336
9 1.0000000
```

Hide

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
rescale01(c(0, 5, 10))
```

```
[1] 0.0 0.5 1.0
```

Rescale a vector to the range [0, 1]

Hide

```
rescale01(c(-10, 0, 10))
```

```
[1] 0.0 0.5 1.0
```

Hide

```
rescale01(c(1, 2, 3, NA, 5))
```

```
[1] 0.00 0.25 0.50   NA 1.00
```

Rescale each column of a DataFrame to the range [0, 1]

Hide

```
df$a <- rescale01(df$a)
df$b <- rescale01(df$b)
df$c <- rescale01(df$c)
df$d <- rescale01(df$d)
```

Hide

```
x <- c(1:10, Inf)
rescale01(x)
```

```
 [1]   0   0   0   0   0   0   0   0   0   0 NaN
```

Define the rescale01 function and apply it

Hide

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
rescale01(x)
```

```
 [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667 0.7777778 0.888888
9 1.0000000
[11]       Inf
```

Load required libraries and packages

Hide

```
library(tidyverse)
library(purrr)
library(magrittr)

# install.packages("pryr")
library(pryr)
```

18.2 Piping alternatives

This is a popular Children's poem that is accompanied by hand actions.We'll start by defining an object to represent little bunny Foo Foo:

Hide

```
# foo_foo <- little_bunny()
```

18.2.1 Intermediate steps

The simplest approach is to save each step as a new object:

Hide

```
# foo_foo_1 <- hop(foo_foo,through=forest)
# foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
# foo_foo_3 <- bop(foo_foo_2, on = head)
```

Create diamonds dataset and calculate price per carat

Hide

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

pryr::object_size(diamonds)
```

```
3.46 MB
```

Hide

```
pryr::object_size(diamonds2)
```

```
3.89 MB
```

Hide

```
pryr::object_size(diamonds,diamonds2)
```

```
3.89 MB
```

Introduce NA value into diamonds$carat and check object sizes

Hide

```
diamonds$carat[1] <- NA
pryr::object_size(diamonds)
```

```
3.46 MB
```

Hide

```
pryr::object_size(diamonds2)
```

```
3.89 MB
```

Hide

```
pryr::object_size(diamonds,diamonds2)
```

```
4.32 MB
```

18.2.2 Overwrite the original

Instead of creating intermediate objects at each step, we could overwrite the original object:

Hide

```
# foo_foo <- hop(foo_foo, through = forest)
# foo_foo <- scoop(foo_foo, up = field_mice)
# foo_foo <- bop(foo_foo, on = head)
```

18.2.3 Function composition

Another approach is to abandon assignment and just string the function calls together:

Hide

```
# bop(
#   scoop(
#     hop(foo_foo, through = forest),
#     up = field_mice
#   ),
#   on = head
# )
```

Here the disadvantage is that you have to read from inside-out, from right-to-left, and that the arguments end up spread far apart (evocatively called the dagwood sandwhich problem). In short, this code is hard for a human to consume.

18.2.4 Use the pipe

Finally, we can use the pipe:

Hide

```
# foo_foo %>%
#   hop(through = forest) %>%
#   scoop(up = field_mice) %>%
#   bop(on = head)
```

Hide

```
# my_pipe <- function(.) {
#   . <- hop(., through = forest)
#   . <- scoop(., up = field_mice)
#   bop(., on = head)
# }
# my_pipe(foo_foo)
```

TASK: Functions that use the current environment. For example, `assign()` will create a new variable with the given name in the current environment:

Hide

```
assign("x",10)
x
```

```
[1] 10
```

Hide

```
"x" %>% assign(100)
x
```

```
[1] 10
```

Assign value to “x” in the specified environment and check its value and Generate random numbers, create a matrix, plot it, and inspect its structure

Hide

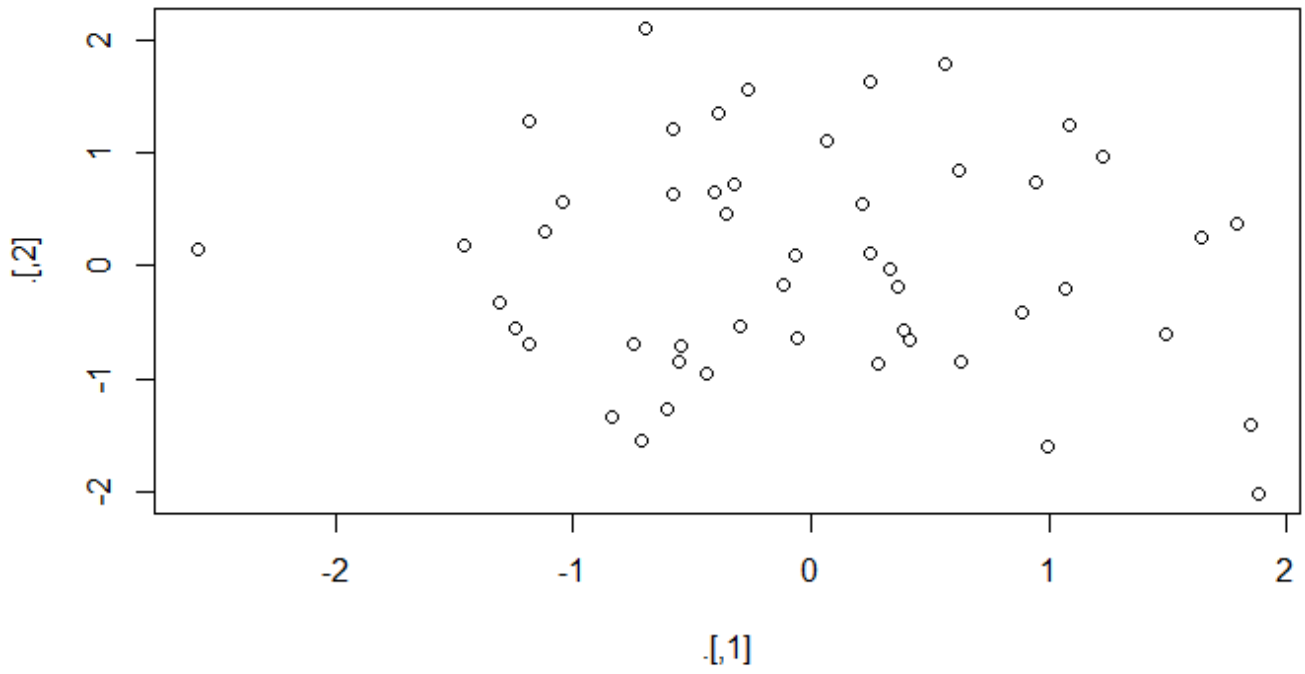
```
env <- environment()
"x" %>% assign(100,envir=env)
x
```

```
[1] 100
```

Hide

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

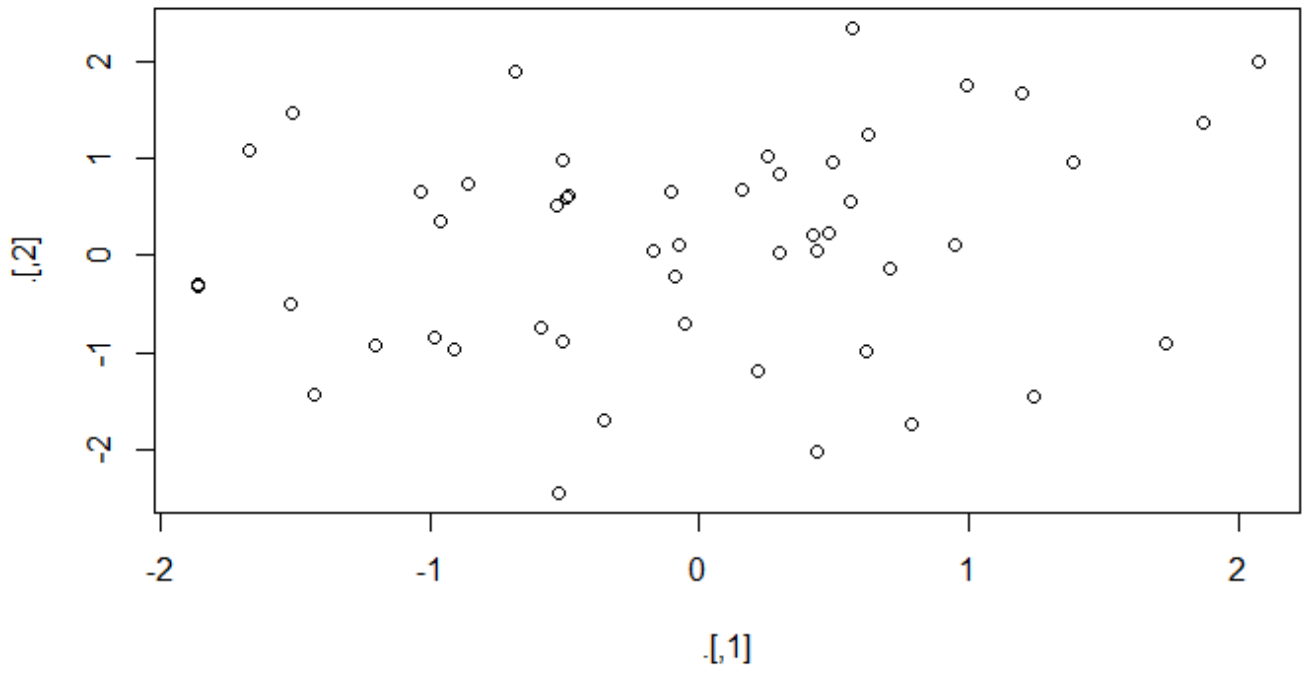
```
 NULL
```



Hide

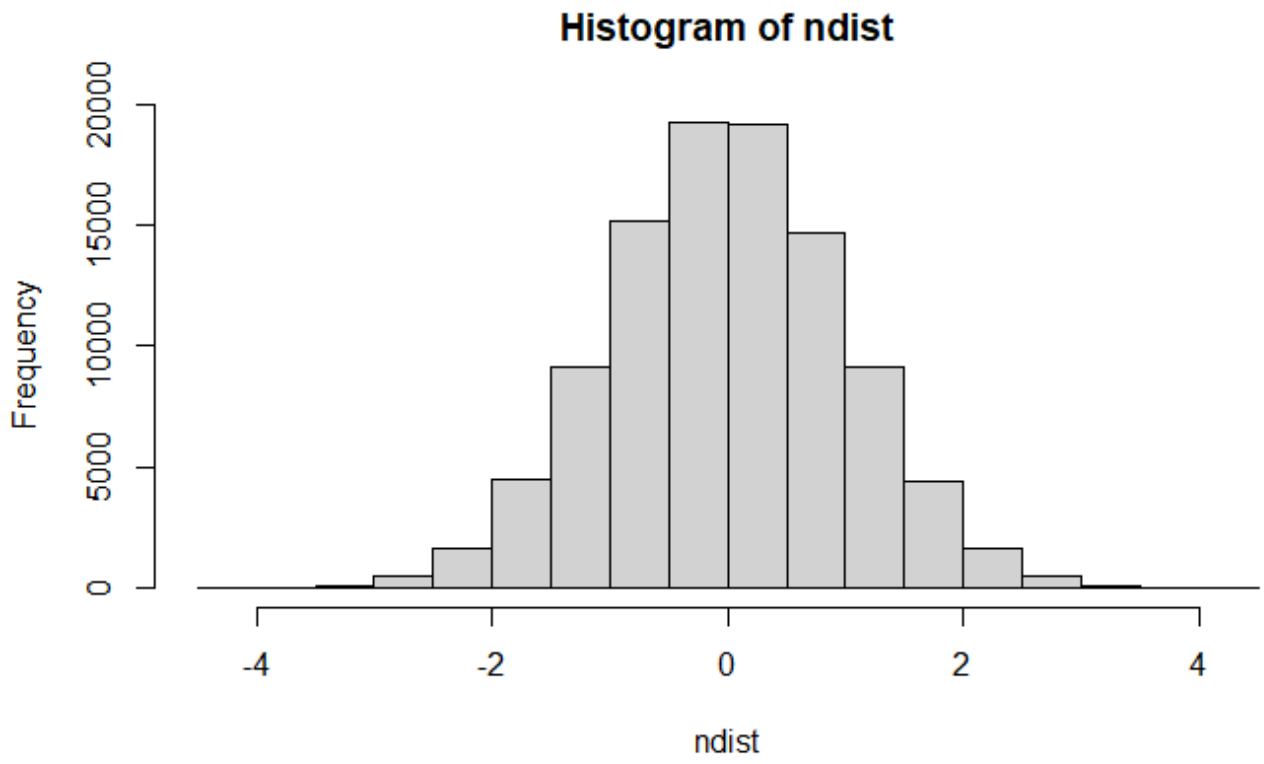
```
rnorm(100) %>%
  matrix(ncol=2) %>%
  plot() %>%
  str()
```

```
 NULL
```



Hide

```
ndist <- rnorm(100000)
hist(ndist)
```



Calculate the correlation between two variables in mtcars dataset

Hide

```
mtcars %$%
  cor(disp, mpg)
```

```
[1] -0.8475514
```

- For assignment magrittr provides the `%<>%` operator which allows you to replace code like:

Hide

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

Hide

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

Chapter 19 Functions

19.1 Introduction

19.2 When should you write a function?

Hide

```
df <- tibble::tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
df
```

a <dbl>	b <dbl>	c <dbl>	d <dbl>
0.04508185	0.87420314	-0.3446325	0.19733324
0.18159777	-1.43453574	-0.3400402	-2.18231604
-0.81072154	0.08701524	-1.2362148	-3.25489892
-1.77761283	-0.78170107	-0.0378040	0.07534082
1.70373373	-0.10076724	-0.4200254	0.78463456
-1.42274540	0.22664823	2.2792016	0.86359696
-0.44332673	-1.57582692	0.2321342	-1.88759627
1.36621029	-0.88583114	0.2791363	0.16364576
0.49444041	-1.08087133	1.9101411	0.56573779
0.59862660	-0.26212695	-0.6960504	0.85445704

1-10 of 10 rows

Hide

```
df$a <- (df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b <- (df$b - min(df$b, na.rm = TRUE)) /
  (max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$c <- (df$c - min(df$c, na.rm = TRUE)) /
  (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d <- (df$d - min(df$d, na.rm = TRUE)) /
  (max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

Rescale a single variable in a data frame

Hide

```
(df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
```

```
 [1] 0.5235603 0.5627738 0.2777349 0.0000000 1.0000000 0.1019340 0.3832672 0.9030480 0.652636
3 0.6825633
```

Rescale a single variable without creating a new object

Hide

```
x <- df$a
(x - min(x, na.rm = T)) / (max(x, na.rm = T)-min(x, na.rm = T))
```

```
 [1] 0.5235603 0.5627738 0.2777349 0.0000000 1.0000000 0.1019340 0.3832672 0.9030480 0.652636
3 0.6825633
```

Task: There is some duplication in this code. We're computing the range of the data three times, so it makes sense to do it in one step:

Hide

```
rng <- range(x, na.rm = T)
(x-rng[1])/(rng[2]-rng[1])
```

```
 [1] 0.5235603 0.5627738 0.2777349 0.0000000 1.0000000 0.1019340 0.3832672 0.9030480 0.652636
3 0.6825633
```

Pulling out intermediate calculations into named variables is a good practice because it makes it more clear what the code is doing. Now that I've simplified the code, and checked that it still works, I can turn it into a function:

Hide

```
rescale01 <- function(x){
  rng <- range(x, na.rm = T)
  (x-rng[1])/(rng[2]-rng[1])
}
rescale01(c(0,5,10))
```

```
[1] 0.0 0.5 1.0
```

Test the rescale01 function with various inputs

Hide

```
rescale01(c(-10,0,10))
```

```
[1] 0.0 0.5 1.0
```

Hide

```
rescale01(c(1,2,3,NA,5))
```

```
[1] 0.00 0.25 0.50   NA 1.00
```

We can simplify the original example now that we have a function:

Hide

```
df$a <- rescale01(df$a)
df$b <- rescale01(df$b)
df$c <- rescale01(df$c)
df$d <- rescale01(df$d)
```

Rescale a vector with infinite values

Hide

```
x <- c(1:10,Inf)
rescale01(x)
```

```
 [1]   0   0   0   0   0   0   0   0   0   0 NaN
```

Because we've extracted the code into a function, we only need to make the fix in one place:

Hide

```
rescale01 <- function(x){
  rng <- range(x,na.rm=T,finite=T)
  (x-rng[1])/(rng[2]-rng[1])
}
rescale01(x)
```

```
 [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667 0.7777778 0.888888
9 1.0000000
[11]       Inf
```

19.4 Conditional execution

An `if` statement allows you to conditionally execute code. It looks like this:

Hide

```
# if (condition) {
  # code executed when condition is TRUE
# } else {
  # code executed when condition is FALSE
# }
```

Define a function to check if an object has names

Hide

```
has_name <- function(x){
  nms <- names(x)
  if(is.null(nms)){
    rep(FALSE,length(x))
  }else {
    !is.na(nms) & nms !=""
  }
}
```

19.4.1 Conditions

how if condition works with warnings

Hide

```
# if (c(TRUE,FALSE)){}
#> Warning in if (c(TRUE, FALSE)) {: the condition has length > 1 and only the
#> first element will be used
#> NULL

# if (NA) {}
```

Check if two objects are identical

Hide

```
identical(0L,0)
```

```
[1] FALSE
```

Hide

```
x <- sqrt(2)^2
x==2
```

```
[1] FALSE
```

Hide

```
x-2
```

```
[1] 4.440892e-16
```

19.4.2 Multiple conditions

You can chain multiple if statement together:

Hide

```
# if (this) {
#   # do that
# } else if (that) {
#   # do something else
# } else {
#   #
# }
```

Hide

```
#> function(x, y, op) {
#>   switch(op,
#>     plus = x + y,
#>     minus = x - y,
#>     times = x * y,
#>     divide = x / y,
#>     stop("Unknown op!")
#>   )
#> }
```

19.4.3 Code style

Good practice for writing if statements

Hide

```
# Good
# if (y < 0 && debug) {
#   message("Y is negative")
# }
#
# if (y == 0) {
#   log(x)
# } else {
#   y ^ x
# }
#
# # Bad
# if (y < 0 && debug)
# message("Y is negative")
#
# if (y == 0) {
#   log(x)
# }
# else {
#   y ^ x
# }
```

It's ok to drop the curly braces if you have a very short if statement that can fit on one line:

Hide

```
y <- 10
x <- if (y < 20) "Too low" else "Too high"
```

I recommend this only for very brief `if` statements. Otherwise, the full form is easier to read:

Hide

```
if (y < 20) {
  x <- "Too low"
} else {
  x <- "Too high"
}
```

19.5 Function arguments

Hide

```
# Compute confidence interval around mean using normal approximation
mean_ci <- function(x, conf = 0.95) {
  se <- sd(x) / sqrt(length(x))
  alpha <- 1 - conf
  mean(x) + se * qnorm(c(alpha / 2, 1 - alpha / 2))
}

x <- runif(100)
mean_ci(x)
```

```
[1] 0.4370008 0.5485763
```

Hide

```
mean_ci(x, conf = 0.99)
```

```
[1] 0.4194710 0.5661061
```

19.5.1 Choosing names

19.5.2 Cheking values

Hide

```
wt_mean <- function(x, w) {
  sum(x * w) / sum(w)
}
wt_var <- function(x, w) {
  mu <- wt_mean(x, w)
  sum(w * (x - mu) ^ 2) / sum(w)
}
wt_sd <- function(x, w) {
  sqrt(wt_var(x, w))
}
```

What happens if x and w are not the same length?

Hide

```
wt_mean(1:6, 1:3)
```

```
[1] 7.666667
```

In this case, because of R's vector recycling rules, we don't get an error.

It's good practice to check important preconditions, and throw an error (with `stop()`), if they are not true:

Hide

```
wt_mean <- function(x, w) {
  if (length(x) != length(w)) {
    stop("`x` and `w` must be the same length", call. = FALSE)
  }
  sum(w * x) / sum(w)
}
```

Hide

```
wt_mean <- function(x, w, na.rm = FALSE) {
  if (!is.logical(na.rm)) {
    stop("`na.rm` must be logical")
  }
  if (length(na.rm) != 1) {
    stop("`na.rm` must be length 1")
  }
  if (length(x) != length(w)) {
    stop("`x` and `w` must be the same length", call. = FALSE)
  }

  if (na.rm) {
    miss <- is.na(x) | is.na(w)
    x <- x[!miss]
    w <- w[!miss]
  }
  sum(w * x) / sum(w)
}
```

This is a lot of extra work for little additional gain. A useful compromise is the built-in `stopifnot()`: it checks that each argument is `TRUE`, and produces a generic error message if not.

Hide

```
wt_mean <- function(x, w, na.rm = FALSE) {
  stopifnot(is.logical(na.rm), length(na.rm) == 1)
  stopifnot(length(x) == length(w))

  if (na.rm) {
    miss <- is.na(x) | is.na(w)
    x <- x[!miss]
    w <- w[!miss]
  }
  sum(w * x) / sum(w)
}
```

19.5.3 Dot-dot-dot(…)

Many functions in R take an arbitrary number of inputs:

Hide

```
sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
[1] 55
```

Hide

```
stringr::str_c("a", "b", "c", "d", "e", "f")
```

```
[1] "abcdef"
```

Define a function to concatenate strings with commas

Hide

```
commas <- function(...) stringr::str_c(..., collapse = ", ")
commas(letters[1:10])
```

```
[1] "a, b, c, d, e, f, g, h, i, j"
```

Hide

```
rule <- function(..., pad = "-") {
  title <- paste0(...)
  width <- getOption("width") - nchar(title) - 5
  cat(title, " ", stringr::str_dup(pad, width), "\n", sep = "")
}
rule("Important output")
```

```
Important output ----------------------------------------------------------------------
------------
```

Hide

```
x <- c(1,2)
sum(x,na.rm=T)
```

```
[1] 3
```

Define a function ‘complicated_function’ with conditions to return 0 if ‘x’ or ‘y’ is empty

Hide

```
complicated_function <- function(x,y,z){
  if (lenth(x)==0 || length(y)==0){
    return(0)
  }
}
```

Improve readability of if-else blocks by using early return for simple cases

Hide

```
f <- function() {
  if (x) {
    # Do
    # something
    # that
    # takes
    # many
    # lines
    # to
    # express
  } else {
    # return something short
  }
}
```

But if the first block is very long, by the time you get to the else, you've forgotten the condition. One way to rewrite it is to use an early return for the simple case:

Hide

```
f <- function() {
  if (!x) {
    return(something_short)
  }

  # Do
  # something
  # that
  # takes
  # many
  # lines
  # to
  # express
}
```

This tends to make the code easier to understand, because you don't need quite so much context to understand it.

19.6.2 Writing pipeable functions

Define a function to show the count of missing values in a data frame

Hide

```
show_missing <- function(df){
  n <- sum(is.na(df))
  cat("Missing values:",n,"\n",sep="")

  invisible(df)
}
```

If we call it interatively, the `invisible()` means that the input `df` does not get printed out:

Hide

```
show_missing(mtcars)
```

```
Missing values:0
```

But it's still there, it's not just printed by default:

Hide

```
x <- show_missing(mtcars)
```

```
Missing values:0
```

Hide

```
class(x)
```

```
[1] "data.frame"
```

Hide

```
dim(x)
```

```
[1] 32 11
```

And we can still use it in a pipe:

Hide

```
library(magrittr)
library(tidyverse)

mtcars %>%
  show_missing() %>%
  mutate(mpg=ifelse(mpg<20,NA,mpg)) %>%
  show_missing()
```

```
Missing values:0
Missing values:18
```

19.7 Environment

Define a function 'f' that takes an argument 'x' and returns the sum of 'x' and 'y'

Hide

```
f <- function(x){
  x+y
}
```

Demonstrate how changing the value of 'y' affects the result of calling function 'f'

Hide

```
y <- 100
f(10)
```

```
[1] 110
```

Hide

```
y <- 1000
f(10)
```

```
[1] 1010
```

Overload the '+' operator to behave differently based on a random condition

Hide

```
`+` <- function(x, y) {
  if (runif(1) < 0.1) {
    sum(x, y)
  } else {
    sum(x, y) * 1.1
  }
}
table(replicate(1000, 1 + 2))
```

```

  3 3.3
 94 906
```

Hide

```
#>
#>   3 3.3
#> 100 900
rm(`+`)
```

Chapter 20: Vectors

20.1.1 PRerequisites

Hide

```
library(tidyverse)
```

20.2 Vector basics

Determine the data type of different vectors

Hide

```
typeof(letters)
```

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

Hide

```
typeof(1:10)
```

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

Determine the length of a list and display its contents

Hide

```
x <- list("a","b",1:10)
length(x)
```

```
[1] 3
```

Hide

```
x
```

```
[[1]]
[1] "a"

[[2]]
[1] "b"

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

Demonstrate modulo operation and creation of logical vectors

Hide

```
1:10 %% 3 ==0
```

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

Hide

```
c(T,T,F,NA)
```

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

20.3.2 Numeric

Integer and double vectors are known collectively as numeric vectors. In R, numbers are doubles by default. To make an integer, place an L after the number:

Hide

```
typeof(1)
```

```
[1] "double"
```

Hide

```
typeof(1L)
```

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

Hide

```
1.5
```

```
[1] 1.5
```

Demonstrate the behavior of floating point arithmetic

Hide

```
x <- sqrt(2)^2
x
```

```
[1] 2
```

Hide

```
x-2
```

```
[1] 4.440892e-16
```

Demonstrate the behavior of division by zero

Hide

```
c(-1,0,1)%/% 0
```

```
[1] -Inf  NaN  Inf
```

Hide

```
# [1] -Inf  NaN  Inf
```

20.3.3 Character

Determine the memory size of a string and a replicated string vector

Hide

```
x <- "This is a reasonably long string."
pryr::object_size(x)
```

```
152 B
```

Hide

```
y <- rep(x,1000)
pryr::object_size(y)
```

```
8.14 kB
```

20.3.4 Missing values

Note that each type of atomic vector has its own missing value:

Hide

```
NA            # logical
```

```
[1] NA
```

Hide

```
NA_integer_   # integer
```

```
[1] NA
```

Hide

```
NA_real_      # double
```

```
[1] NA
```

Hide

```
NA_character_ # character
```

```
[1] NA
```

Calculate the number and proportion of elements in a vector greater than 10

Hide

```
x <- sample(20,100,replace=T)
y <- x > 10
sum(y) # how many are greater than 10?
```

```
[1] 49
```

Hide

```
mean(y) # what proportion are greater than 10?
```

```
[1] 0.49
```

Hide

```
if (length(x)){
}
```

```
NULL
```

Determine the data type of different vectors

Hide

```
typeof(c(TRUE,1L))
```

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

Hide

```
typeof(c(1L,1.5))
```

```
[1] "double"
```

Hide

```
typeof(c(1.5,"a"))
```

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

Generate random numeric or logical vectors

Hide

```
sample(10)+100
```

```
 [1] 110 103 104 101 102 107 109 106 105 108
```

Hide

```
runif(10)>0.5
```

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

Demonstrate vector arithmetic with vectors of different lengths

Hide

```
1:10 +1:2
```

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

Hide

```
1:10+1:3
```

```
Warning: longer object length is not a multiple of shorter object length
```

```
 [1]  2  4  6  5  7  9  8 10 12 11
```

Create a tibble with two columns, ‘x’ and ‘y’, with different lengths

Hide

```
library(tidyverse)



tibble(
  x=1:4,
  y=rep(1:2,each=2)
)
```

x <int>	y <int>
1	1
2	1
3	2
4	2

4 rows

20.4.4 Naming vectors

All types of vectors can be named. You can name them during creatin with `c()`:

Hide

```
c(x=1,y=2,z=4)
```

```
x y z
1 2 4
```

Or after the fact with `purr::set_names()`

Hide

```
set_names(1:3,c("a","b","c"))
```

```
a b c
1 2 3
```

Named vectors are most useful for subsetting, described next.

20.4.5 Subsetting

Demonstrate subsetting vectors with integer indices

Hide

```
x <- c("one","two","three","four","five")
x[c(3,2,5)]
```

```
[1] "three" "two"   "five"
```

By repeating a position, you can actually make a longer output than input:

Hide

```
x[c(1,1,5,5,5,2)]
```

```
[1] "one"  "one"  "five" "five" "five" "two"
```

Negative values drop the elements at the specified positions:

Hide

```
x[c(-1,-3,-5)]
```

```
[1] "two"  "four"
```

The error message mentions subsetting with zero, which returns no values:

Hide

```
x[0]
```

```
character(0)
```

Hide

```
library(tidyverse)
x <- c(10,3,NA,5,8,1)

# tibble test
x <- as.tibble(x,ncol=1)
```

```
Warning: `as.tibble()` was deprecated in tibble 2.0.0.
Please use `as_tibble()` instead.
The signature and semantics have changed, see `?as_tibble`.
```

Hide

```
names(x)="v1"
is.na(x)
```

```
       v1
[1,] FALSE
[2,] FALSE
[3,]  TRUE
[4,] FALSE
[5,] FALSE
[6,] FALSE
```

Hide

```
x %>% filter(v1 == NA)
```

```
0 rows
```

Hide

```
# all non-missing values of x
x <- c(10,3,NA,5,8,1)
x[!is.na(x)]
```

```
[1] 10  3  5  8  1
```

Hide

```
# all even (or missing) values of x
x[x %% 2==0]
```

```
[1] 10 NA  8
```

3. If you have a named vector, you can subset it with a character vector:

Hide

```
x <- c(abc=1, def=2,xyz=5)
x[c("xyz","def")]
```

```
xyz def
  5   2
```

20.5 Recursive vectors (lists)

Create a list with numeric elements

Hide

```
x <- list(1,2,3)
x
```

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

[[2]]
[1] 2

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

Display the structure of lists with and without names

Hide

```
str(x)
```

```
List of 3
 $ : num 1
 $ : num 2
 $ : num 3
```

Hide

```
x_named <- list(a=1,b=2,c=3)
str(x_named)
```

```
List of 3
 $ a: num 1
 $ b: num 2
 $ c: num 3
```

Unlike atomic vectors, `list()` can contain a mix of objects:

Hide

```
y <- list("a",1L,1.5,T)
str(y)
```

```
List of 4
 $ : chr "a"
 $ : int 1
 $ : num 1.5
 $ : logi TRUE
```

List can even contain other lists!

Hide

```
z <- list(list(1,2),list(3,4))
str(z)
```

```
List of 2
 $ :List of 2
  ..$ : num 1
  ..$ : num 2
 $ :List of 2
  ..$ : num 3
  ..$ : num 4
```

20.5.1 Visualizing lists

Hide

```
x1 <- list(c(1,2),c(3,4))
x2 <- list(list(1,2),list(3,4))
x3 <- list(1,list(2,list(3)))
x1
```

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

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

Hide

```
x2
```

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

[[1]][[2]]
[1] 2


[[2]]
[[2]][[1]]
[1] 3

[[2]][[2]]
[1] 4
```

Hide

```
x3
```

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

[[2]]
[[2]][[1]]
[1] 2

[[2]][[2]]
[[2]][[2]][[1]]
[1] 3
```

20.5.2 Subsetting

Create a list ‘a’ with named elements and demonstrate subsetting

Hide

```
a <- list(a = 1:3, b = "a string", c = pi, d = list(-1, -5))
```

Hide

```
str(a)
```

```
List of 4
 $ a: int [1:3] 1 2 3
 $ b: chr "a string"
 $ c: num 3.14
 $ d:List of 2
  ..$ : num -1
  ..$ : num -5
```

Hide

```
str(a[1:2])
```

```
List of 2
 $ a: int [1:3] 1 2 3
 $ b: chr "a string"
```

Hide

```
str(a[4])
```

```
List of 1
 $ d:List of 2
  ..$ : num -1
  ..$ : num -5
```

Demonstrate subsetting lists using double square brackets

Hide

```
str(a[[1]])
```

```
 int [1:3] 1 2 3
```

Hide

```
str(a[[4]])
```

```
List of 2
 $ : num -1
 $ : num -5
```

Access list elements by name using $ or [[ ]]

Hide

```
a$a
```

```
[1] 1 2 3
```

Hide

```
a[["a"]]
```

```
[1] 1 2 3
```

20.6 Attributes

Demonstrate setting and retrieving attributes of vectors

Hide

```
x <- 1:10
attr(x,"greeting")
```

```
NULL
```

Hide

```
attr(x,"greeting") <- "Hi!"
attr(x,"farewell") <- "Bye!"
attributes(x)
```

```
$greeting
[1] "Hi!"

$farewell
[1] "Bye!"
```

Demonstrate methods for class ‘Date’

Hide

```
as.Date
```

```
function (x, ...)
UseMethod("as.Date")
<bytecode: 0x0000022ea4225378>
<environment: namespace:base>
```

Hide

```
methods("as.Date")
```

```
[1] as.Date.character   as.Date.default     as.Date.factor      as.Date.numeric     as.Date.P
OSIXct
[6] as.Date.POSIXlt     as.Date.vctrs_sclr* as.Date.vctrs_vctr*
see '?methods' for accessing help and source code
```

Retrieve specific methods for ‘as.Date’

Hide

```
getS3method("as.Date","default")
```

```
function (x, ...)
{
    if (inherits(x, "Date"))
        x
    else if (is.null(x))
        .Date(numeric())
    else if (is.logical(x) && all(is.na(x)))
        .Date(as.numeric(x))
    else stop(gettextf("do not know how to convert '%s' to class %s",
        deparse1(substitute(x)), dQuote("Date")), domain = NA)
}
<bytecode: 0x0000022eb85b7c20>
<environment: namespace:base>
```

Hide

```
getS3method("as.Date","numeric")
```

```
function (x, origin, ...)
if (missing(origin)) .Date(x) else as.Date(origin, ...) + x
<bytecode: 0x0000022ec2ac86e0>
<environment: namespace:base>
```

20.7.1 Factors

Demonstrate creating a factor and inspecting its attributes

Hide

```
x <- factor(c("ab","cd","ab"),levels=c("ab","cd","ed"))
typeof(x)
```

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

Hide

```
attributes(x)
```

```
$levels
[1] "ab" "cd" "ed"

$class
[1] "factor"
```

20.7.2 Dates and date-times

Dates in R are numeric vectors that represent the number of days since 1 January 1970.

Hide

```
x <- as.Date("1971-01-01")
unclass(x)
```

```
[1] 365
```

Hide

```
typeof(x)
```

```
[1] "double"
```

Hide

```
attributes(x)
```

```
$class
[1] "Date"
```

Demonstrate creating and inspecting a date-time object

Hide

```
x <- lubridate::ymd_hm("1970-01-01 01:00")
unclass(x)
```

```
[1] 3600
attr(,"tzone")
[1] "UTC"
```

Hide

```
typeof(x)
```

```
[1] "double"
```

Hide

```
attributes(x)
```

```
$class
[1] "POSIXct" "POSIXt"

$tzone
[1] "UTC"
```

Demonstrate setting and retrieving time zone for date-time object

Hide

```
attr(x,"tzone") <- "US/Pacific"
x
```

```
[1] "1969-12-31 17:00:00 PST"
```

Hide

```
attr(x,"tzone") <- "US/Eastern"
x
```

```
[1] "1969-12-31 20:00:00 EST"
```

There is another type of date-times called POSIXlt. There are built on top of named lists:

Hide

```
y <- as.POSIXlt(x)
typeof(y)
```

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

Hide

```
#> [1] "list"
attributes(y)
```

```
$names
 [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"   "yday"   "isdst"  "zone"
"gmtoff"

$class
[1] "POSIXlt" "POSIXt"

$tzone
[1] "US/Eastern" "EST"        "EDT"

$balanced
[1] TRUE
```

20.7.3 Tibbles

Tibbles are augmented lists: they have class “tbl_df” + “tbl” + “data.frame”, and `names` (column) and `row.names` attributes:

Hide

```
tb <- tibble::tibble(x = 1:5, y = 5:1)
typeof(tb)
```

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

Hide

```
attributes(tb)
```

```
$class
[1] "tbl_df"     "tbl"        "data.frame"

$row.names
[1] 1 2 3 4 5

$names
[1] "x" "y"
```

Hide

```
df <- data.frame(x = 1:5, y = 5:1)
typeof(df)
```

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

Hide

```
attributes(df)
```

```
$names
[1] "x" "y"

$class
[1] "data.frame"

$row.names
[1] 1 2 3 4 5
```

Chapter 21: Iteration

21.1.1 Prerequisites

Hide

```
library(tidyverse)
```

21.2 For loops

Imagine we have this simple tibble:

Hide

```
df <- tibble(
  a=rnorm(10),
  b=rnorm(10),
  c=rnorm(10),
  d=rnorm(10)
)
```

Calculate the median for each column in a tibble

Hide

```
median(df$a)
```

```
[1] -0.3157254
```

Hide

```
median(df$b)
```

```
[1] -0.8006407
```

Hide

```
median(df$c)
```

```
[1] -0.2668019
```

Hide

```
median(df$d)
```

```
[1] -0.02814063
```

Calculate the median for each column in the data frame ‘df’ using a for loop

Hide

```
df
```

a <dbl>	b <dbl>	c <dbl>	d <dbl>
-1.093586814	-0.4781807	1.051162466	1.04642857
0.360744062	-1.0089671	1.014811812	0.06561313
-3.267196736	-0.3453872	-0.681894700	2.08345420
0.397686234	-0.8250089	1.023585432	-0.12189439
1.364972013	-1.7465822	-2.073352998	1.06465499
-0.717558975	-0.9241738	0.009441455	-1.34762298
-0.628546567	-1.6676786	-1.027602745	0.24414093
-0.002904135	-0.6403235	-0.543045303	-1.16098142
-0.884824164	-0.7762724	-0.660444264	-1.41167540
1.056161469	0.1922766	0.962490952	-0.71263224

1-10 of 10 rows

Hide

```
output <- vector("double",ncol(df))
for (i in seq_along(df)){
  output[[i]] <- median(df[[i]])
}
output <- tibble(output)
```

Demonstrate the behavior of seq_along and length functions with an empty vector ‘y’

Hide

```
y <- vector("double", 0)
seq_along(y)
```

```
integer(0)
```

Hide

```
#> integer(0)
1:length(y)
```

```
[1] 1 0
```

Hide

```
#> [1] 1 0
```

21.3.1v Modifying an existing object

Sometimes, you want to use a for loop to modify an existing object. For example, remember our challenges from functions. We wanted to rescale every column in a data frame:

Hide

```
library(tidyverse)

df <- tibble(
  a=rnorm(10),
  b=rnorm(10),
  c=rnorm(10),
  d=rnorm(10)
)

rescale01 <- function(x){
  rng <- range(x,na.rm=T)
  (x-rng[1])/(rng[2]-rng[1])
}

df$a <- rescale01(df$a)
df$b <- rescale01(df$b)
df$c <- rescale01(df$c)
df$d <- rescale01(df$d)

df
```

a <dbl>	b <dbl>	c <dbl>	d <dbl>
0.1557623	0.7527127	1.0000000	0.59317076
1.0000000	1.0000000	0.1041454	0.75164916
0.2393430	0.5793436	0.3573255	0.55847160
0.3648265	0.2708436	0.0000000	0.08247656
0.0000000	0.4962822	0.4390808	1.00000000
0.7892904	0.5622966	0.4678596	0.86571564
0.1161658	0.3776416	0.8060481	0.00000000
0.7137158	0.1492057	0.4712845	0.62864031
0.7523917	0.0000000	0.4775396	0.50295649
0.3879652	0.6207524	0.5996382	0.66323436

1-10 of 10 rows

Hide

```
for ( i in seq_along(df)){
  df[[i]] <- rescale01(df[[i]])
}
```

21.3.2 Looping patterns

Hide

```
x
```

```
[1] "1969-12-31 20:00:00 EST"
```

Hide

```
results <- vector("list",length(x))
names(results) <- names(x)
```

Demonstrate looping patterns using a for loop to iterate over a list ‘x’ and store results in a list ‘results’

Hide

```
for(i in seq_along(x)){
  name <- names(x)[[i]]
  value <- x[[i]]
}
```

21.3.3 Unknown output length

Create a vector ‘output’ with unknown length and store results from a for loop in it

Hide

```
means <- c(0,1,2)

output <- double()
for (i in seq_along(means)){
  n <- sample(100,1)
  output <- c(output,rnorm(n,means[[i]]))
}
str(output)
```

```
 num [1:223] 2.4083 1.5499 0.6081 0.0844 0.7443 ...
```

Hide

```
output
```

```
  [1]  2.40834490  1.54985893  0.60813582  0.08444820  0.74433326  0.23589873  0.13677913 -1.
51138770
  [9] -1.27301392 -1.76413099 -0.63497070  1.54956856  1.50375944  0.71571312  0.57801330  0.
33952611
 [17]  0.55112157  0.17114550 -0.45463725  1.16554397  0.69994812 -1.38517572 -0.21089332  0.
59729886
 [25]  0.96649672  0.27565281  2.07450311  0.94767106 -1.19450592 -1.17615918 -0.06135937  0.
31565475
 [33]  0.46863199 -2.44533524  1.06774440 -0.53263928  0.79354070 -1.03657232 -1.41232073 -1.
32268012
 [41] -0.80619868 -1.48689463 -1.54482571  1.03872808 -1.69903338 -1.03393281  1.63922764  1.
21681751
 [49] -1.50423215  0.08619177 -0.64176595 -0.43528690  2.35908464 -0.07057289  0.79716367  1.
39285456
 [57]  1.99268652  1.96561675  4.02439840  2.43533373  1.71102597 -1.18110259  0.59191107  1.
10348971
 [65]  0.73300412  0.12159976  0.27290089  1.61147640  0.05298932  0.28489074  0.51530326  2.
48330465
 [73]  1.37412695  1.25174310  1.51181660  0.51833391  1.22731542  0.94751275  2.14534938  3.
73603533
 [81]  2.50448575 -0.07692596  0.38360741  1.13672743  1.59343745  1.59795077  1.43992598  1.
32878827
 [89]  4.19537063  1.71172058  1.10573970  0.60868714  1.07205788  2.30309804  0.89510610  1.
53102920
 [97]  1.34831205  1.42881339  1.98257191  1.81541903  1.01326728  0.19392951  0.33627612  0.
47158036
[105]  2.83906013  0.38940341  1.87692593  0.86561042  1.79769711  1.32415638  1.66865453  2.
99697358
[113]  0.46228139  0.33591193  0.24631821 -0.43897896 -0.11946077  1.11951899  2.53562810  1.
15236359
[121] -0.09132156 -1.16767499  0.69617045  1.53977593 -0.25021887  2.18171198  1.97165693 -0.
23020504
[129] -0.44086089  1.15684137  2.24634334  1.55135055  2.31883196  0.67556263  0.08018167  1.
13714558
[137]  1.91669266  1.11139134  2.44770118  0.19471109  2.17871196  4.23369053 -0.15692858  2.
21704642
[145]  2.90250569 -0.47232133  1.53732998  0.24186549  2.47811777  3.33046579  2.23734182  0.
96433477
[153]  2.53867102  2.31206058  1.97274207  2.13828221  2.57547609  1.13761332  1.18604257  1.
61607018
[161]  2.37121047  2.84795864  3.27669950  1.60560488  2.10355804  3.13176141  3.26092255  0.
75913241
[169]  0.30603003  2.52789744  0.60599780  2.73159671  0.64479899 -0.11114226  0.61712535  3.
05991705
[177]  1.35211904  1.02472332  1.29546010  2.25444890  1.57803759  0.76376634 -0.73903589  2.
93147906
[185]  2.07135777  1.17651876  1.52535341  1.73003413  0.57964605  1.54963744  2.16950772  1.
45261534
[193]  2.78626480  2.75094794  2.99609585  2.54613765  1.71969902  4.22074008  2.97537992  1.
47012235
[201]  1.27664953  2.99738151  2.29757845  1.24353502  2.34876874  0.40451196  0.98056608  2.
15815198
[209]  1.21343698  2.37080467  1.56698860  3.19657275  0.47615986  0.53829438  3.06730806  1.
80701539
[217]  2.43974635  1.12710760  2.72351352  2.57993773  3.26991511  1.74089217  0.42477812
```

Create a list ‘out’ with unknown length and store results from a for loop in it

Hide

```
out <- vector("list",length(means))
for (i in seq_along(means)){
  n <- sample(100,1)
  out[[i]] <- rnorm(n,means[[i]])
}
str(out)
```

```
List of 3
 $ : num [1:23] 0.109 0.669 -0.159 -0.325 -0.81 ...
 $ : num [1:97] 1.57 2.15 2.41 1.75 1.1 ...
 $ : num [1:9] 1.86 2.62 2.25 1.62 2.24 ...
```

Hide

```
str(unlist(out))
```

```
 num [1:129] 0.109 0.669 -0.159 -0.325 -0.81 ...
```

21.3.4 Unknown sequence length

A while loop is also more general than a for loop, because you can rewrite any for loop as a while loop, but you can’t rewrite every while loop as for loop:

Hide

```
for (i in seq_along(x)) {
  # body
}

# Equivalent to
i <- 1
while (i <= length(x)) {
  # body
  i <- i + 1
}
```

Herhow we could use a while loop to find how many tries it takes to get three heads in a row:

Hide

```
flip <- function() sample(c("T", "H"), 1)

flips <- 0
nheads <- 0

while (nheads < 3) {
  if (flip() == "H") {
    nheads <- nheads + 1
  } else {
    nheads <- 0
  }
  flips <- flips + 1
}
flips
```

```
[1] 26
```

21.4 For loops vs. functionals

Compare for loop and functional approaches for calculating column means in a data frame

Hide

```
df <- tibble(
  a=rnorm(10),
  b=rnorm(10),
  c=rnorm(10),
  d=rnorm(10)
)
```

Using for loop

Hide

```
output <- vector("double",length(df))
for (i in seq_along(df)){
  output[[i]] <- mean(df[[i]])
}
output
```

```
[1] 0.476473102 0.001854536 0.558698854 0.220409290
```

Using functional approach with a custom function ‘col_mean’

Hide

```
col_mean <- function(df){
  output <- vector("double",length(df))
  for (i in seq_along(df)){
    output[i] <- mean(df[[i]])
  }
  output
}
```

Define a function ‘col_median’ to calculate the median for each column in the data frame ‘df’

Hide

```
col_median <- function(df){
  output <- vector("double",hh(df))
  for (i in seq_along(df)){
    output[i] <- median(df[[i]])
  }
  output
}

col_sd <- function(df){
  output <- vector("double",length(df))
  for (i in seq_along(df)){
    output[i] <- sd(df[[i]])
  }
  output
}

df
```

a <dbl>	b <dbl>	c <dbl>	d <dbl>
-0.59424108	-1.68448348	1.51726158	0.2021551
0.32005523	0.36049180	0.40034078	1.2241879
0.64975730	1.13706648	0.66252561	-0.1877430
0.62648826	0.34973367	0.01501266	0.5889792
2.28760378	-0.15291861	0.30353919	-0.7364203
-0.17251628	-0.16285294	-0.08672882	0.2646598
-1.58866804	1.43494295	-0.26808423	-0.7787254
2.16931528	-1.12885705	0.79183230	1.0440623
-0.05421073	-0.02241468	0.23917343	0.1718362
1.12114730	-0.11216277	2.01211603	0.4111009

1-10 of 10 rows

Define functions f1, f2, and f3 for calculating different powers of absolute deviation from the mean

Hide

```
f1 <- function(x) abs(x-mean(x))^1
f2 <- function(x) abs(x-mean(x))^2
f3 <- function(x) abs(x-mean(x))^3
```

Define a function ‘f’ to calculate the absolute deviation from the mean raised to a given power ‘i’

Hide

```
f <- function(x,i) abs(x-mean(x))^i
```

Define a function ‘col_summary’ to apply a summary function ‘fun’ to each column of the data frame ‘df’

Hide

```
col_summary <- function(df, fun) {
  out <- vector("double", length(df))
  for (i in seq_along(df)) {
    out[i] <- fun(df[[i]])
  }
  out
}
col_summary(df, median)
```

```
[1]  0.47327175 -0.06728873  0.35193999  0.23340748
```

Hide

```
col_summary(df, mean)
```

```
[1] 0.476473102 0.001854536 0.558698854 0.220409290
```

Demonstrate the use of ‘map_dbl’ from the ‘purrr’ package to apply a function to each column of the data frame ‘df’

Hide

```
library(purrr)
head(df)
```

a <dbl>	b <dbl>	c <dbl>	d <dbl>
-0.5942411	-1.6844835	1.51726158	0.2021551
0.3200552	0.3604918	0.40034078	1.2241879
0.6497573	1.1370665	0.66252561	-0.1877430
0.6264883	0.3497337	0.01501266	0.5889792
2.2876038	-0.1529186	0.30353919	-0.7364203
-0.1725163	-0.1628529	-0.08672882	0.2646598

6 rows

Hide

```
# Reference - for loop()
output <- vector("double",length(df))
for (i in seq_along(df)){
  output[[i]] <- mean(df[[i]])
}
output
```

```
[1] 0.476473102 0.001854536 0.558698854 0.220409290
```

Hide

```
map_dbl(df,mean)
```

```
          a           b           c           d
0.476473102 0.001854536 0.558698854 0.220409290
```

Hide

```
map_dbl(df,median)
```

```
          a           b           c           d
 0.47327175 -0.06728873  0.35193999  0.23340748
```

Hide

```
map_dbl(df,sd)
```

```
        a         b         c         d
1.1903576 0.9296956 0.7214505 0.6619177
```

Hide

```
df %>% map_dbl(mean)
```

```
          a           b           c           d
0.476473102 0.001854536 0.558698854 0.220409290
```

Hide

```
df %>% map_dbl(median)
```

```
          a           b           c           d
 0.47327175 -0.06728873  0.35193999  0.23340748
```

Hide

```
df %>% map_dbl(sd)
```

```
        a         b         c         d
1.1903576 0.9296956 0.7214505 0.6619177
```

Demonstrate the use of ‘map_dbl’ from the ‘purrr’ package with additional arguments

Hide

```
map_dbl(df,mean,trim=0.5)
```

```
          a           b           c           d
 0.47327175 -0.06728873  0.35193999  0.23340748
```

Demonstrate the use of ‘map_int’ from the ‘purrr’ package to apply a function that returns integers to each element of a list

Hide

```
z <- list(x=1:3,y=4:5)
z
```

```
$x
[1] 1 2 3

$y
[1] 4 5
```

Hide

```
map_int(z,length)
```

```
x y
3 2
```

21.5.1 Shortcuts

Demonstrate the use of ‘safely’ from the ‘purrr’ package to create a safe version of a function

Hide

```
safe_log <- safely(log)
str(safe_log(10))
```

```
List of 2
 $ result: num 2.3
 $ error : NULL
```

Hide

```
str(safe_log("a"))
```

```
List of 2
 $ result: NULL
 $ error :List of 2
  ..$ message: chr "non-numeric argument to mathematical function"
  ..$ call   : language .Primitive("log")(x, base)
  ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

Demonstrate the use of ‘map’ from the ‘purrr’ package with ‘safely’ to apply a safe version of a function to each element of a list

Hide

```
x <- list(1,10,"a")
y <- x %>% map(safely(log))
str(y)
```

```
List of 3
 $ :List of 2
  ..$ result: num 0
  ..$ error : NULL
 $ :List of 2
  ..$ result: num 2.3
  ..$ error : NULL
 $ :List of 2
  ..$ result: NULL
  ..$ error :List of 2
  .. ..$ message: chr "non-numeric argument to mathematical function"
  .. ..$ call   : language .Primitive("log")(x, base)
  .. ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

Demonstrate the use of 'transpose' from the 'purrr' package to transpose a list of lists

Hide

```
y <- x %>% transpose()
str(y)
```

```
List of 1
 $ :List of 3
  ..$ : num 1
  ..$ : num 10
  ..$ : chr "a"
```

Demonstrate the use of error handling with 'map_lgl' and 'is_null' from the 'purrr' package

Hide

```
is_ok <- y$error %>% map_lgl(is_null)
x[!is_ok]
```

```
list()
```

Hide

```
# y$result[is_ok] %>% flatten_dbl()
```

Purrr provides two other useful adverbs:

Hide

```
x <- list(1,10,"a")
x %>% map_dbl(possibly(log,NA_real_))
```

```
[1] 0.000000 2.302585       NA
```

Demonstrate the use of ‘quietly’ from the ‘purrr’ package to suppress errors and return results with warnings

Hide

```
x <- list(1,-1)
x %>% map(quietly(log)) %>% str()
```

```
List of 2
 $ :List of 4
  ..$ result  : num 0
  ..$ output  : chr ""
  ..$ warnings: chr(0)
  ..$ messages: chr(0)
 $ :List of 4
  ..$ result  : num NaN
  ..$ output  : chr ""
  ..$ warnings: chr "NaNs produced"
  ..$ messages: chr(0)
```

21.7 Mapping over multiple arguments

Generate random numbers from normal distributions with different means using ‘map’ from the ‘purrr’ package

Hide

```
mu <- list(5,10,-3)
mu %>%
  map(rnorm,n=5) %>%
  str()
```

```
List of 3
 $ : num [1:5] 5.11 4.32 5.23 5.09 4.36
 $ : num [1:5] 9.66 9.56 12.07 9.6 7.39
 $ : num [1:5] -2.57 -3.32 -4.2 -2.16 -2.32
```

Generate random numbers from normal distributions with different means and standard deviations using ‘map2’ from the ‘purrr’ package

Hide

```
sigma <- list(1,5,10)
seq_along(mu) %>%
  map(~rnorm(5,mu[[.]],sigma[[.]])) %>%
  str()
```

```
List of 3
 $ : num [1:5] 5.36 7.16 4.32 4.53 5.97
 $ : num [1:5] 21.09 8.26 15.76 8.76 8.33
 $ : num [1:5] -11.56 -11 -4.91 -10.91 -13.74
```

Define a custom ‘map2’ function to apply a binary function to corresponding elements of two lists

Hide

```
map2(mu,sigma,rnorm,n=5) %>% str()
```

```
List of 3
 $ : num [1:5] 4.85 4.27 4.85 3.68 3.97
 $ : num [1:5] 15.4 16.3 13.5 11.2 11.4
 $ : num [1:5] 4.561 -0.279 -11.559 0.298 -5.152
```

Hide

```
map2 <- function(x,y,f,...){
  out <- vector("list",length(x))
  for (i in seq_along(x)){
    out[[i]] <- f(x[[i]],y[[i]],...)
  }
  out
}
```

Apply a function to corresponding elements of multiple lists using ‘pmap’ from the ‘purrr’ package

Hide

```
library(magrittr)
library(purrr)

n <- list(1,3,5)
args1 <- list(n,mu,sigma)
args1 %>%
  pmap(rnorm) %>%
  str()
```

```
List of 3
 $ : num 3.71
 $ : num [1:3] 6.11 12.73 9.31
 $ : num [1:5] -12.562 -1.359 -0.823 -20.731 -12.676
```

Apply a function to corresponding elements of multiple lists with named parameters using ‘pmap’ from the ‘purrr’ package

Hide

```
args2 <- list(mean=mu, sd=sigma,n=n)
args2 %>%
  pmap(rnorm) %>%
  str()
```

```
List of 3
 $ : num 4.34
 $ : num [1:3] 9.78 8.19 11.21
 $ : num [1:5] 0.742 6.791 14.991 -1.91 0.511
```

Apply a function to corresponding rows of a data frame using ‘pmap’ from the ‘purrr’ package with a tibble

Hide

```
library(tidyverse)
parms <- tribble(
  ~mean,~sd,~n,
  5,1,1,
  10,5,3,
  -3,10,5
)

parms %>%
  pmap(rnorm)
```

```
[[1]]
[1] 3.243648

[[2]]
[1]  9.605242 11.025883  6.007252

[[3]]
[1]  16.669768   5.855557 -12.841618  -3.360844   5.139707
```

21.7.1 Involing different functions

Invoke different functions with different parameters using ‘invoke_map’ from the ‘purrr’ package

Hide

```
f <- c("runif","rnorm","rpois")
param <- list(
  list(min=-1,max=1),
  list(sd=5),
  list(lambda=10)
)

f
```

```
[1] "runif" "rnorm" "rpois"
```

Hide

```
param
```

```
[[1]]
[[1]]$min
[1] -1

[[1]]$max
[1] 1


[[2]]
[[2]]$sd
[1] 5


[[3]]
[[3]]$lambda
[1] 10
```

To handle this case, you can use `invoke_map()` :

Hide

```
invoke_map(f,param,n=5) %>%
  str()
```

```
Warning: `invoke_map()` was deprecated in purrr 1.0.0.
Please use map() + exec() instead.
```

```
List of 3
 $ : num [1:5] 0.7386 -0.9649 -0.0834 -0.4986 -0.9628
 $ : num [1:5] 6.447 -6.531 -0.428 6.091 8.155
 $ : int [1:5] 9 11 9 9 11
```

Invoke different functions with different parameters using ‘pmap’ from the ‘purrr’ package and a tibble

Hide

```
sim <- tribble(
  ~f,      ~params,
  "runif", list(min = -1, max = 1),
  "rnorm", list(sd = 5),
  "rpois", list(lambda = 10)
)
sim %>%
  mutate(sim = invoke_map(f, params, n = 10))
```

f	params	sim
<chr>	<list>	<list>
runif	<list [2]>	<dbl [10]>
rnorm	<list [1]>	<dbl [10]>
rpois	<list [1]>	<int [10]>

3 rows

21.8 Walk

Perform side effects without returning a value for each element of a list using ‘walk’ from the ‘purrr’ package

Hide

```
x <- list(1,"a",3)
x %>%
  walk(print)
```

```
[1] 1
[1] "a"
[1] 3
```

Perform side effects on each element of a list using ‘walk’ from the ‘purrr’ package, then save the results

Hide

```
library(ggplot2)
plots <- mtcars %>%
  split(.$cyl) %>%
  map(~ggplot(., aes(mpg, wt)) + geom_point())
paths <- stringr::str_c(names(plots), ".pdf")

pwalk(list(paths, plots), ggsave, path = tempdir())
```

```
Saving 7 x 7 in image
```

Retain or remove elements of a list based on a predicate function using ‘keep’ and ‘discard’ from the ‘purrr’ package

Hide

```
iris %>%
  keep(is.factor) %>%
  str()
```

```
'data.frame':   150 obs. of  1 variable:
 $ Species: Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 ...
```

Hide

```
iris %>%
  discard(is.factor) %>%
  str()
```

```
'data.frame':   150 obs. of  4 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

Hide

```
library(tidyverse)
library(magrittr)
```

21.9.2 Reduce and accumulate

Iteratively combine elements of a list using a binary function with ‘reduce’ from the ‘purrr’ package

Hide

```
dfs <- list(
  age=tibble(name="John",age=30),
  sex=tibble(name=c("John","Mary"),sex=c("M","F")),
  trt=tibble(name="Mary",treatment="A")
)

dfs %>% reduce(full_join)
```

```
Joining with `by = join_by(name)`Joining with `by = join_by(name)`
```

name <chr>	age <dbl>	sex <chr>	treatment <chr>
John	30	M	*NA*
Mary	*NA*	F	A

2 rows

Find the intersection of multiple vectors using ‘reduce’ from the ‘purrr’ package

Hide

```
vs <- list(
  c(1,3,5,6,10),
  c(1,2,3,7,8,10),
  c(1,2,3,4,8,9,10)
)
vs %>% reduce(intersect)
```

```
[1]  1  3 10
```

Iteratively apply a function to elements of a list using ‘accumulate’ from the ‘purrr’ package

Hide

```
x <- sample(10)
x
```

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

Hide

```
x %>% accumulate(`+`)
```

```
[1]  7 10 14 16 24 33 34 39 49 55
```