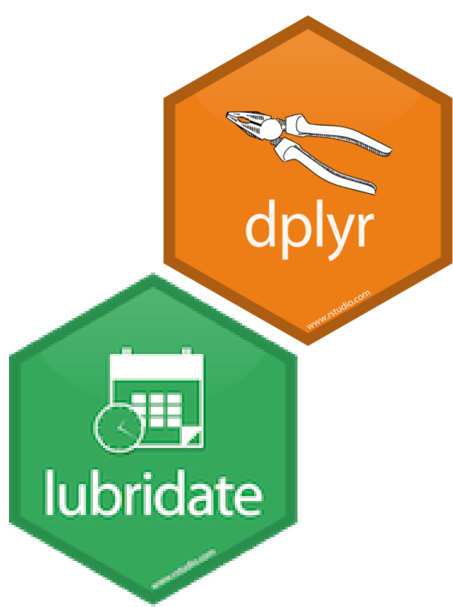


PARIS
LODRON
UNIVERSITÄT
SALZBURG

dplyr and lubridate

Olga Mironova, Birgit Mitter and Pia Neuwirth

SE Statistics, Visualization and More Using "R“
April 9th, 2024

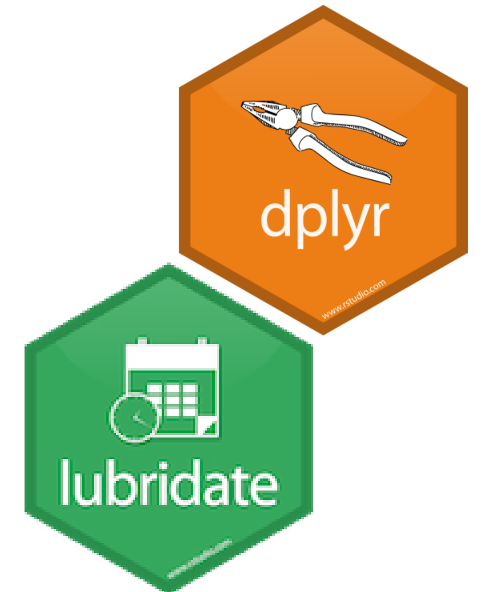
PARIS LODRON UNIVERSITÄT SALZBURG

Contents & Schedule

- **Quick introduction** to packages and datasets of our session

- Introduction, examples and exercises of **dplyr**
- Introduction, examples and exercises of **lubridate**

- Final exercise **merging our knowledge** on dplyr and lubridate
- Final **questions & feedback** on our session

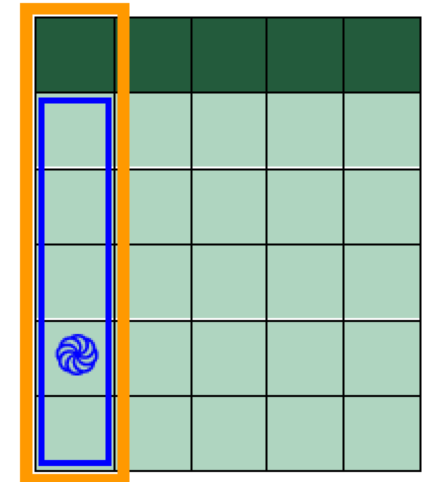


PARIS LODRON UNIVERSITÄT SALZBURG

What is this session all about?

The **package dplyr** can be used for data transformation and to do basic (mostly univariate) statistics of variables.

- Types of variables: boolean, categorial (unsorted and sorted), metric
- Important terms to know: value, vector, variable, dataset
- Functions of dplyr require "tidy data"
    - Each variable is in its own collumn
    - Each observation/case is in its own row
- Functions of dplyr work with pipes: x %>% f(y)



The **package lubridate** helps us to work with dates and times and to do basic maths like calculating periods, durations or intervals.

Which data are we working with?

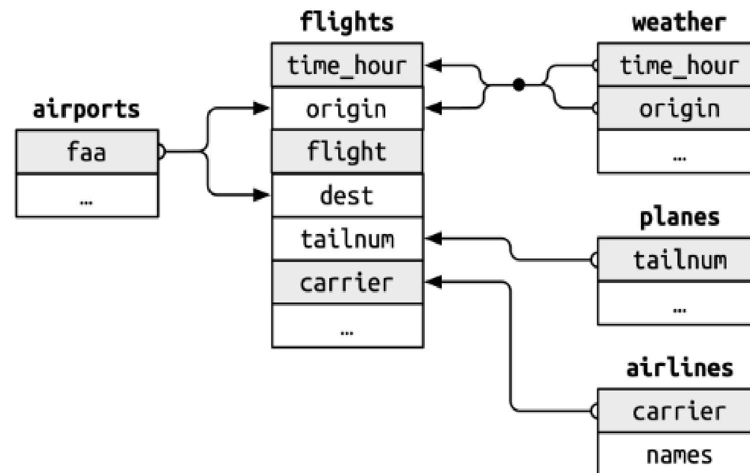
PARIS LODRON UNIVERSITÄT SALZBURG

Let's go to New York City! And also leave from there...

✈ We use the public dataset "nycflights13" - please load in RStudio!

✈ 336 776 observations of 19 variables

✈ Includes different types of variables for dplyr as well as times and dates for lubridate



dplyr

Why do we want to use dplyr?

Operations of dplyr can also be achieved using basic R functions.

However, advantages of dplyr are:

- More efficient processing
- Intuitive syntax
- Use of tidyverse pipe operator allows for easy chaining of operations

Grouping & Summarising Cases

- Calculate the number of cases for a category by using **count()**.
- Use **group_by()** to get a new table based on a categorial variable. Any dplyr functions are applied seperately for these groups and the results are displayed in a newly created table.
- Create a table on indicators you need for a specific variable by using **summarise()**. The function applies multiple sub-functions.
- Very convenient to combine summarise() and group_by() to show differences between case groups, e. g. experimental group and control group in laboratory experiments.

```
flights_carrierdistances = flights %>% group_by(carrier)  %>%
  summarise(
    sum_distance = sum(distance)
  )
```

	carrier	sum_distance
1	9E	9788152
2	AA	43864584
3	AS	1715028
4	B6	58384137
5	DL	59507317
6	EV	30498951
7	F9	1109700
8	FL	2167344
9	HA	1704186
10	MQ	15033955
11	OO	16026
12	UA	89705524
13	US	11365778
14	VX	12902327
15	WN	12229203
16	YV	225395

summarise()-Function

- **summarise_all()**: Applies a summary function to all columns/variables.
- **summarise_at()**: Applies a summary function to selected columns based on conditions specified by vars() or a list of column names.
- **summarise_if()**: Applies a summary function to columns that meet specific conditions specified by a predicate function.

Within **summarise()**-function:

- Basic information: **sum(), n(), first(), last(),**
- Central tendency: **mean(), median(), weighted.mean()**
- Distribution: **sd(), var()**
- Variability: **min(), max(), quantile(), IQR(), range()**

Manipulate Cases

- Extract cases:
  - **filter()** to select on logical criteria
  - **distinct()** to remove rows with duplicates
  - **sample_frac()** and **sample_n()** for a data sample (fraction or specific sample size)
  - **slice()** and **top_n()** to select rows by their position or the top n entries
- Arrange cases with **arrange()** (low to high), include **desc()** if high to low
- Add one more more rows to data table: **add_row()**

```
flights %>%
  filter(air_time>60)
```

```
flights %>%
  slice(1:100)
```

```
flights %>%
  arrange(desc(air_time))
```

Manipulate Variables: Extracting

PARIS LODRON UNIVERSITÄT SALZBURG

- **pull()** extracts the values of a column/variable as a vector.
- **select()** and **select_if()** extracts them as a whole column/variable.
  You can also select more variables at once (then: table!)
  You can refine your selection more precisely by including:
  contains(), starts_with(), ends_with(), matches(), one_of(),...

```
flights %>%
  pull(var = dest)
```

```
  [1] "IAH" "IAH" "MIA" "BQN" "ATL" "ORD" "FLL" "IAD" "MCO" "ORD" "PBI" "TPA" "LAX" "SFO" "DFW"
 [16] "BOS" "LAS" "FLL" "ATL" "PBI" "MSP" "DTW" "MIA" "ATL" "MIA" "ORD" "SFO" "RSW" "SJU" "ATL"
 [31] "PHX" "MIA" "IAH" "MSP" "MSP" "PHX" "SJU" "LAX" "ORD" "BWI" "CLT" "IAD" "DFW" "MCO" "BOS"
 [46] "PBI" "CLT" "FLL" "BUF" "DEN" "SNA" "LAS" "MSY" "PBI" "SLC" "SFO" "MIA" "ORD" "MCO" "XNA"
 [61] "TPA" "FLL" "ATL" "LAX" "MIA" "FLL" "DTW" "RSW" "SJU" "LAX" "ORD" "SJU" "FLL" "ORD" "MKE"
 [76] "MCO" "PBI" "DFW" "SEA" "DFW" "DEN" "IAH" "SFO" "ROC" "RSW" "MCO" "SYR" "SFO" "ORD" "IAH"
 [91] "TPA" "LAX" "SRQ" "SEA" "SFO" "SFO" "ORD" "MCO" "DEN" "CLT" "MIA" "ATL" "DEN" "MCO" "MIA"
[106] "MSP" "RDU" "BOS" "BOS" "SFO" "MCO" "CLT" "FLL" "CMH" "ATL" "JAX" "MSP" "PBI" "CHS" "CLT"
```

```
flights %>%
  select(starts_with("dep"))
```

```
   dep_time dep_delay
      <int>     <dbl>
 1      517         2
 2      533         4
 3      542         2
 4      544        -1
 5      554        -6
 6      554        -4
 7      555        -5
 8      557        -3
 9      557        -3
10      558        -2
# ℹ 336,766 more rows
```

Manipulate Variables: Adding

PARIS LODRON UNIVERSITÄT SALZBURG

- **mutate()** to compute new columns based on existing ones.
- **transmute()** to compute new columns and drop (all) old ones.
- **mutate_all()** to apply a function to all columns (e.g. log).
- **mutate_at()** to apply a function when specific conditions are met. You can refine your selection just as with **select()**.
- **rename()** columns to give new names to your variables.

```
flights <- flights %>%
  mutate(
    distance = distance*1,61
    )
```

```
flights <- flights %>%
  rename(
    distance_km = distance
  )
```

Exercise 1: Working with cases

PARIS LODRON UNIVERSITÄT SALZBURG

- Count flights departing from each of the three airports.

- Group the dataset by the airports of New York City ("*origin*"). Summarise the data for each airport in a nice table by storing the average delay at arrival and standard deviation for each of them.
  Hint: There are missing values. Include "na.rm = TRUE" as an argument when calculating the mean and standard deviation.

- Add a new boolean variable to the dataset if a flight was delayed.
  Hint: A flight is delayed when "*arr_delay*" is positive (>0 !).

Combining Data

- **`bind_rows()`** is a function in R used to combine multiple data frames by row-wise concatenation.
- Matches columns by name,ensuring proper alignment
- *places one table "under" another*

Data Frame One

Var1	Var2	Var3
1		
2		
3		
4		

Data Frame Two

Var1	Var2	Var3
5		
6		
7		
8		

Var1	Var2	Var3
1		
2		
3		
4		
5		
6		
7		
8		

Combining Data `bind_rows()`

PARIS LODRON UNIVERSITÄT SALZBURG

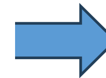
```
# Load departure delays dataset
departure_delays <- flights %>%
  select(year, month,day, dep_delay)

# Load arrival delays dataset
arrival_delays <- flights %>%
  select(year,month,day, arr_delay)

combined_delays <- bind_rows(departure_delays, arrival_delays)
```

	year	month	day	dep_delay
1	2013	1	1	2
2	2013	1	1	4
3	2013	1	1	2
4	2013	1	1	-1
5	2013	1	1	-6
6	2013	1	1	-4
7	2013	1	1	-5
8	2013	1	1	-3
9	2013	1	1	-3
10	2013	1	1	-2
11	2013	1	1	-2

wing 1 to 12 of 336,776 entries, 5 total columns

	year	month	day	arr_delay
1	2013	1	1	11
2	2013	1	1	20
3	2013	1	1	33
4	2013	1	1	-18
5	2013	1	1	-25
6	2013	1	1	12
7	2013	1	1	19
8	2013	1	1	-14
9	2013	1	1	-8
10	2013	1	1	8
11	2013	1	1	-2

howing 1 to 12 of 336,776 entries, 4 total columns

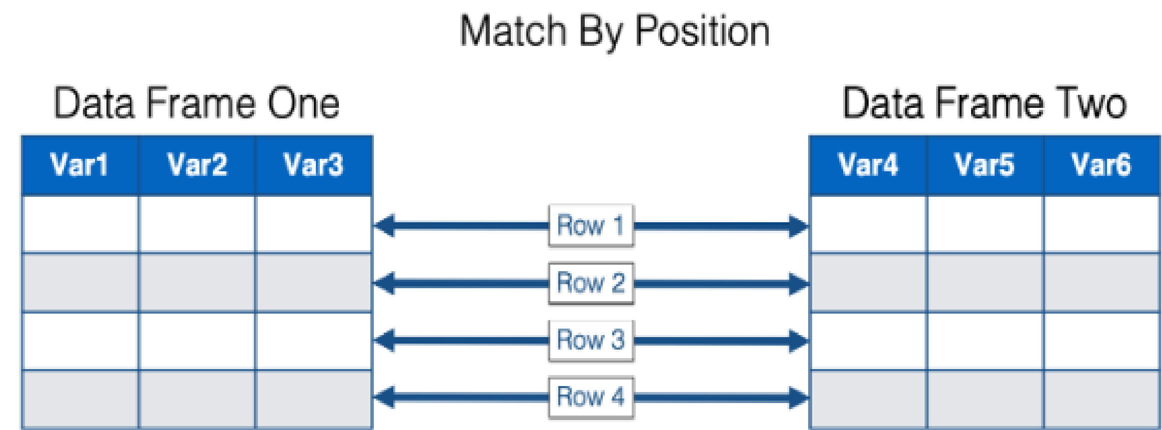


	year	month	day	dep_delay	arr_delay
1	2013	1	1	2	NA
2	2013	1	1	4	NA
3	2013	1	1	2	NA
4	2013	1	1	-1	NA
5	2013	1	1	-6	NA
6	2013	1	1	-4	NA
7	2013	1	1	-5	NA
8	2013	1	1	-3	NA
9	2013	1	1	-3	NA
10	2013	1	1	-2	NA
11	2013	1	1	-2	NA

Showing 1 to 12 of 673,552 entries, 5 total columns

Combining Data

- **bind_cols()** is a function in the dplyr package used to bind multiple data frames column-wise.
- combine datasets without altering their row order.
- when we horizontally combine data frames by position both data frames must have the same number of rows
- *puts one table "to the right" of the other*



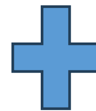
Combining Data bind_cols()

```
library(nycflights13)

# Load departure delays dataset
departure_delays <- flights %>%
  select(year, month, dep_delay)

# Load arrival delays dataset
arrival_delays <- flights %>%
  select(year, arr_delay)

combined_delays <- bind_cols(departure_delays, arrival_delays)
```

departure_delays × R presentation.R* ×

Filter

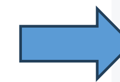
	year	month	dep_delay
1	2013	1	2
2	2013	1	4
3	2013	1	2
4	2013	1	-1
5	2013	1	-6



departure_delays × R presentation.R* ×

Filter

	year	arr_delay
1	2013	11
2	2013	20
3	2013	33
4	2013	-18
5	2013	-25



	year...1	month	dep_delay	year...4	arr_delay
1	2013	1	2	2013	11
2	2013	1	4	2013	20
3	2013	1	2	2013	33
4	2013	1	-1	2013	-18
5	2013	1	-6	2013	-25

Combining Data

PARIS LODRON UNIVERSITÄT SALZBURG

- **union()** is a function in R used to combine the rows of two or more datasets, removing duplicate rows.
- Retains the column names of the datasets
- *returns rows that exist in any of the tables (duplicates are excluded)*

Table A

COL_A	COL_B
1	A
2	B
3	C
5	E

Table B

COL_A	COL_B
1	A
2	B
3	C
4	D

```
Table_A %>%
      union (., Table_B )
```

RESULT

COL_A	COL_B
1	A
2	B
3	C
5	E
4	D

Combining Data

```r
# Filter flights with arrival delay of 100 or 15 minutes
flights_100 <- filter(flights, arr_delay == 100 | arr_delay == 15)
# Filter flights arriving with arrival delay of 15 minutes
flights_15 <- filter(flights, arr_delay == 15)
# Combine datasets using union()
combined_flights_100_15 <- union(flights_100, flights_15)
```

Showing 1 to 11 of 2,742 entries, 19 total columns

Console | Terminal | Background Jobs

R 4.3.1 · ~/R presentation/

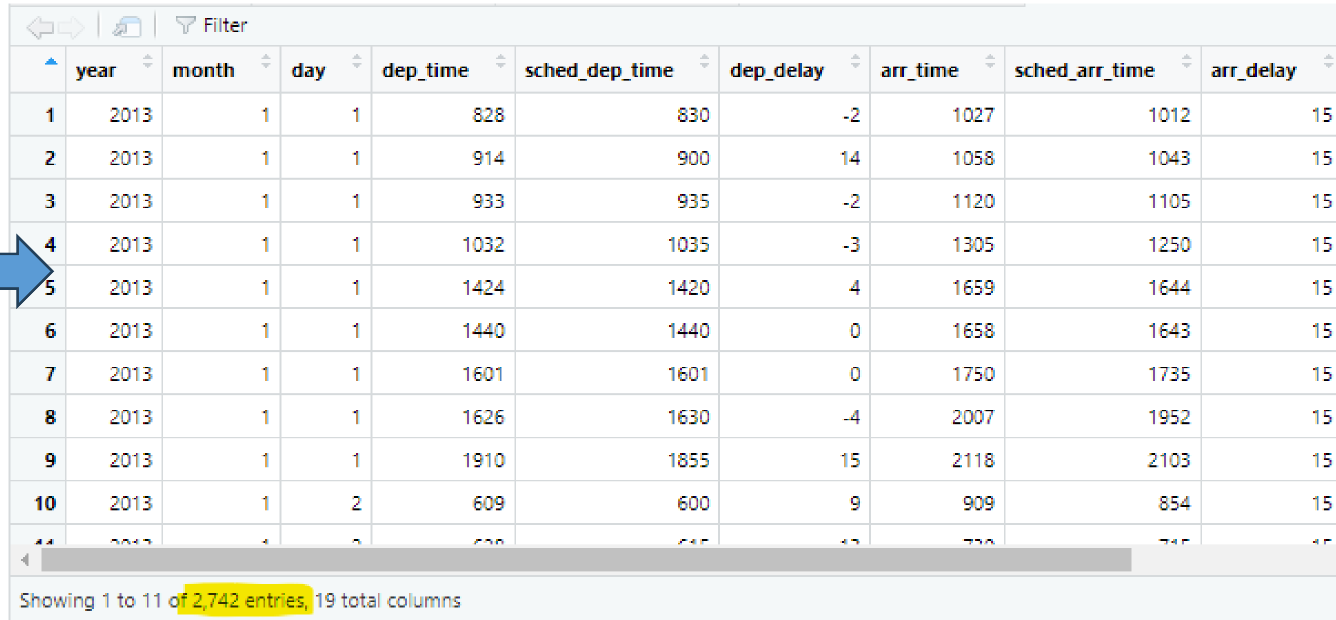
```
>
> # Combine datasets using union()
> combined_flights_100_15 <- union(flights_100, flights_15)
> # Filter flights with arrival delay of 100 or 15 minutes
> flights_100 <- filter(flights, arr_delay == 100 | arr_delay == 15)
>
```

Showing 2,460 to 2,470 of 2,470 entries, 19 total columns

Console | Terminal | Background Jobs

R 4.3.1 · ~/R presentation/

```
> # Filter flights with arrival delay of 100 or 15 minutes
> flights_100 <- filter(flights, arr_delay == 100 | arr_delay == 15)
>
> # Filter flights arriving at JFK with arrival delay of 15 minutes
> flights_15 <- filter(flights, arr_delay == 15)
>
```



Filter

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
1	2013	1	1	828	830	-2	1027	1012	15
2	2013	1	1	914	900	14	1058	1043	15
3	2013	1	1	933	935	-2	1120	1105	15
4	2013	1	1	1032	1035	-3	1305	1250	15
5	2013	1	1	1424	1420	4	1659	1644	15
6	2013	1	1	1440	1440	0	1658	1643	15
7	2013	1	1	1601	1601	0	1750	1735	15
8	2013	1	1	1626	1630	-4	2007	1952	15
9	2013	1	1	1910	1855	15	2118	2103	15
10	2013	1	2	609	600	9	909	854	15

Showing 1 to 11 of 2,742 entries, 19 total columns

Combining Data

PARIS LODRON UNIVERSITÄT SALZBURG

- **Union_all()** is a function in R used to combine rows from two or more datasets without removing duplicate rows.
- *returns rows that exist in any of the tables (duplicates are included)*

Table A

COL_A	COL_B
1	A
2	B
3	C
5	E

Table B

COL_A	COL_B
1	A
2	B
3	C
4	D

```
Table_A %>%
  union_all (., Table_B )
```

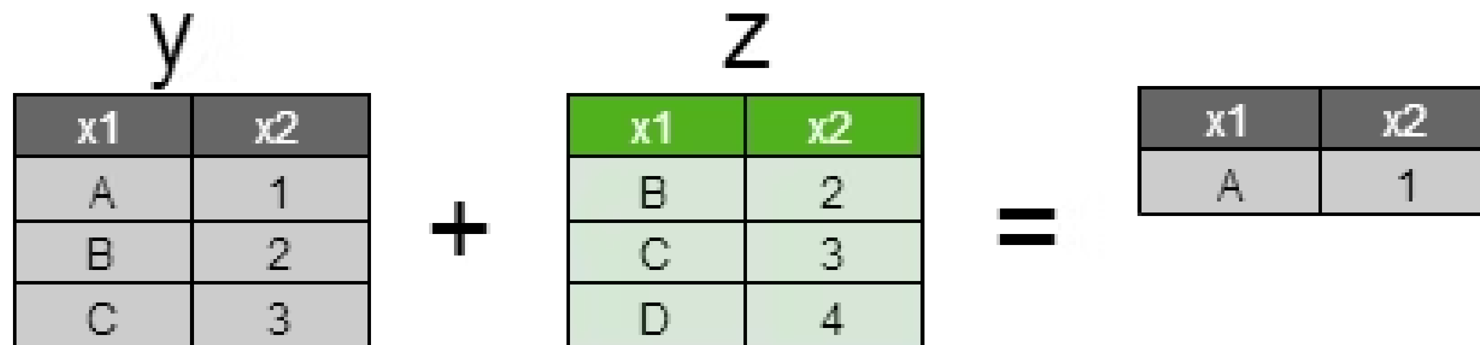
RESULT

COL_A	COL_B
1	A
2	B
3	C
5	E
1	A
2	B
3	C
4	D

Combining Data

PARIS LODRON UNIVERSITÄT SALZBURG

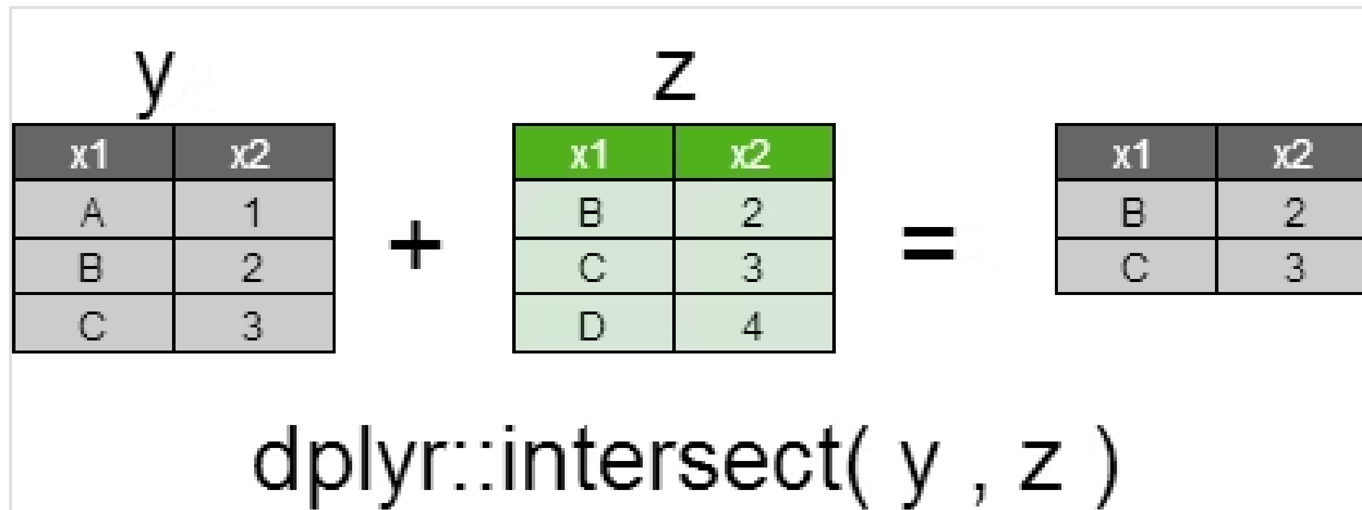
- **Setdiff()** is a function in R used to find the set difference between two vectors or data frames.
- *rows from the first table that are not in the second table*



Combining Data

PARIS LODRON UNIVERSITÄT SALZBURG

- **intersect()** is a function in R used to find the intersection of two vectors or data frames.

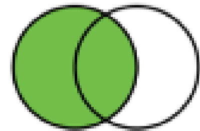


Combining Data

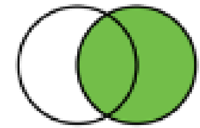
PARIS LODRON UNIVERSITÄT SALZBURG

With joins:

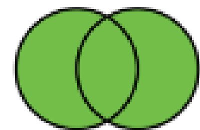
left_join



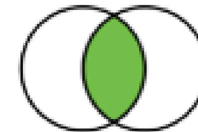
right_join



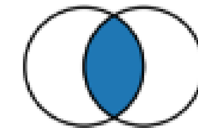
full_join



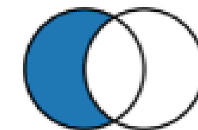
inner_join



semi_join

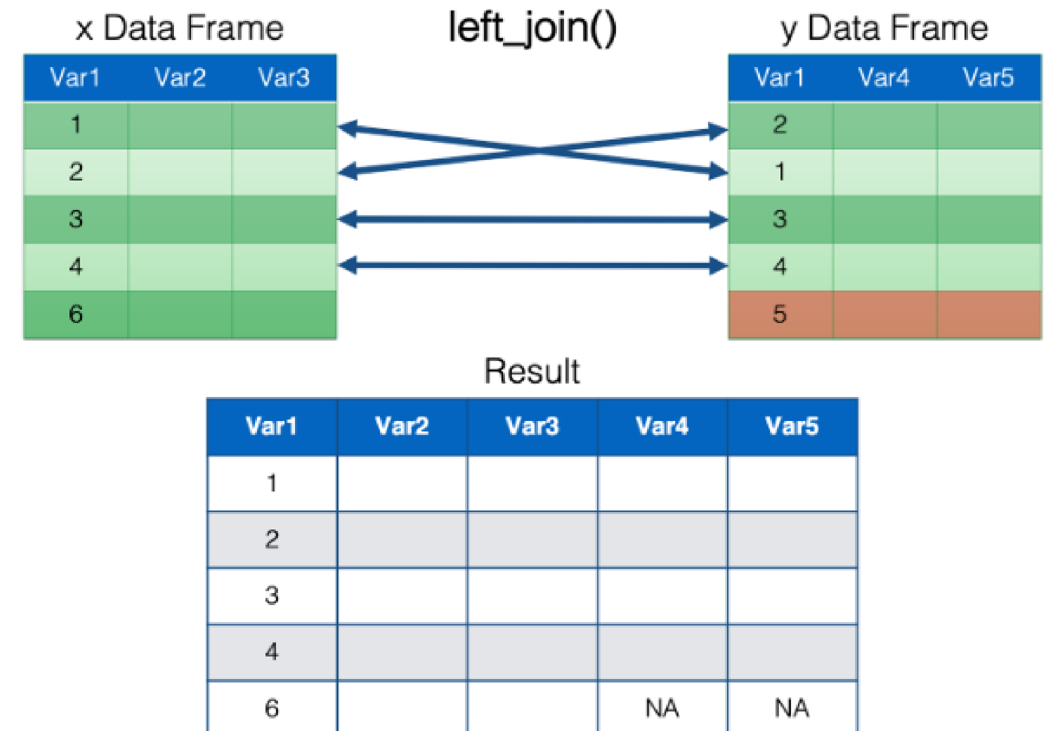


anti_join



Combining Data

left_join() keeps all the rows from the x data fr ame in the resulting combined data frame. However, it only keeps the rows from the y data frame that have a key value match in the x data frame. The values for columns with no key value match in the opposite data frame are set to NA.



Combining Data

- left_join()

```
27 flights2 <- flights |>
28   select(year, time_hour, origin, dest, tailnum, carrier)
29 flights2
```

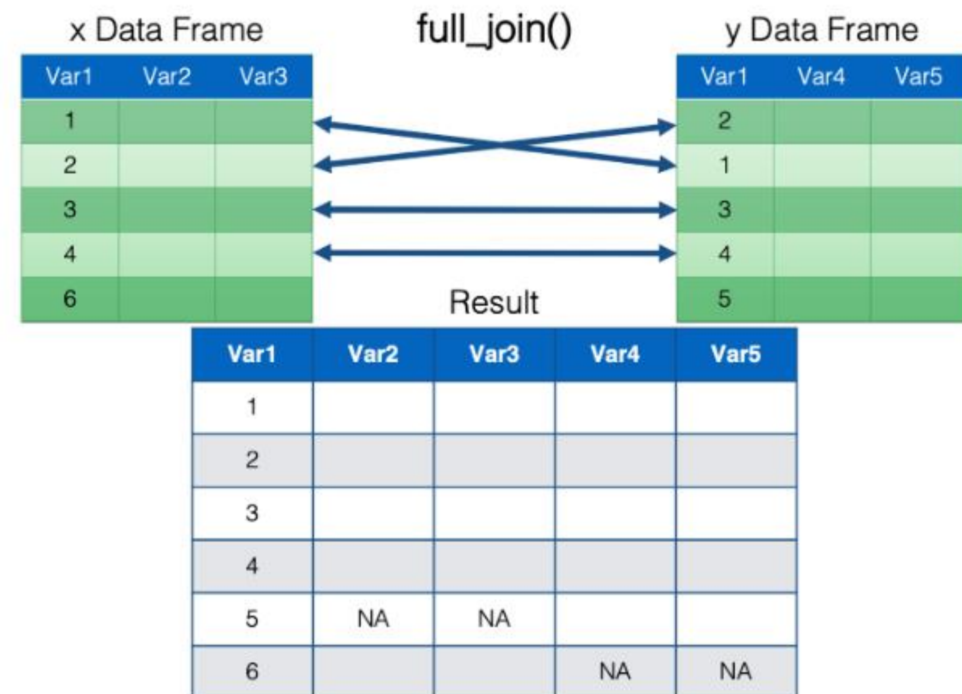
```
27:1   (Top Level)
Console   Terminal   Background Jobs
R 4.3.1 · ~/R presentation/
flights2
A tibble: 336,776 x 6
   year time_hour           origin dest  tailnum carrier
  <int> <dttm>              <chr>  <chr> <chr>   <chr>
  2013 2013-01-01 05:00:00 EWR    IAH   N14228  UA
  2013 2013-01-01 05:00:00 LGA    IAH   N24211  UA
  2013 2013-01-01 05:00:00 JFK    MIA   N619AA  AA
  2013 2013-01-01 05:00:00 JFK    BQN   N804JB  B6
  2013 2013-01-01 06:00:00 LGA    ATL   N668DN  DL
  2013 2013-01-01 05:00:00 EWR    ORD   N39463  UA
  2013 2013-01-01 06:00:00 EWR    FLL   N516JB  B6
  2013 2013-01-01 06:00:00 LGA    IAD   N829AS  EV
  2013 2013-01-01 06:00:00 JFK    MCO   N593JB  B6
  2013 2013-01-01 06:00:00 LGA    ORD   N3ALAA  AA
i 336,766 more rows
```

```
30 #> Joining with `by = join_by(carrier)`
31 flights2 |>
32   left_join(airlines)
33
```

```
30:1   (Top Level)
Console   Terminal   Background Jobs
R 4.3.1 · ~/R presentation/
A tibble: 336,776 x 7
   year time_hour           origin dest  tailnum carrier name
  <int> <dttm>              <chr>  <chr> <chr>   <chr>   <chr>
  2013 2013-01-01 05:00:00 EWR    IAH   N14228  UA      United Air Lines Inc.
  2013 2013-01-01 05:00:00 LGA    IAH   N24211  UA      United Air Lines Inc.
  2013 2013-01-01 05:00:00 JFK    MIA   N619AA  AA      American Airlines Inc.
  2013 2013-01-01 05:00:00 JFK    BQN   N804JB  B6      JetBlue Airways
  2013 2013-01-01 06:00:00 LGA    ATL   N668DN  DL      Delta Air Lines Inc.
  2013 2013-01-01 05:00:00 EWR    ORD   N39463  UA      United Air Lines Inc.
  2013 2013-01-01 06:00:00 EWR    FLL   N516JB  B6      JetBlue Airways
  2013 2013-01-01 06:00:00 LGA    IAD   N829AS  EV      ExpressJet Airlines Inc.
  2013 2013-01-01 06:00:00 JFK    MCO   N593JB  B6      JetBlue Airways
  2013 2013-01-01 06:00:00 LGA    ORD   N3ALAA  AA      American Airlines Inc.
i 336,766 more rows
```

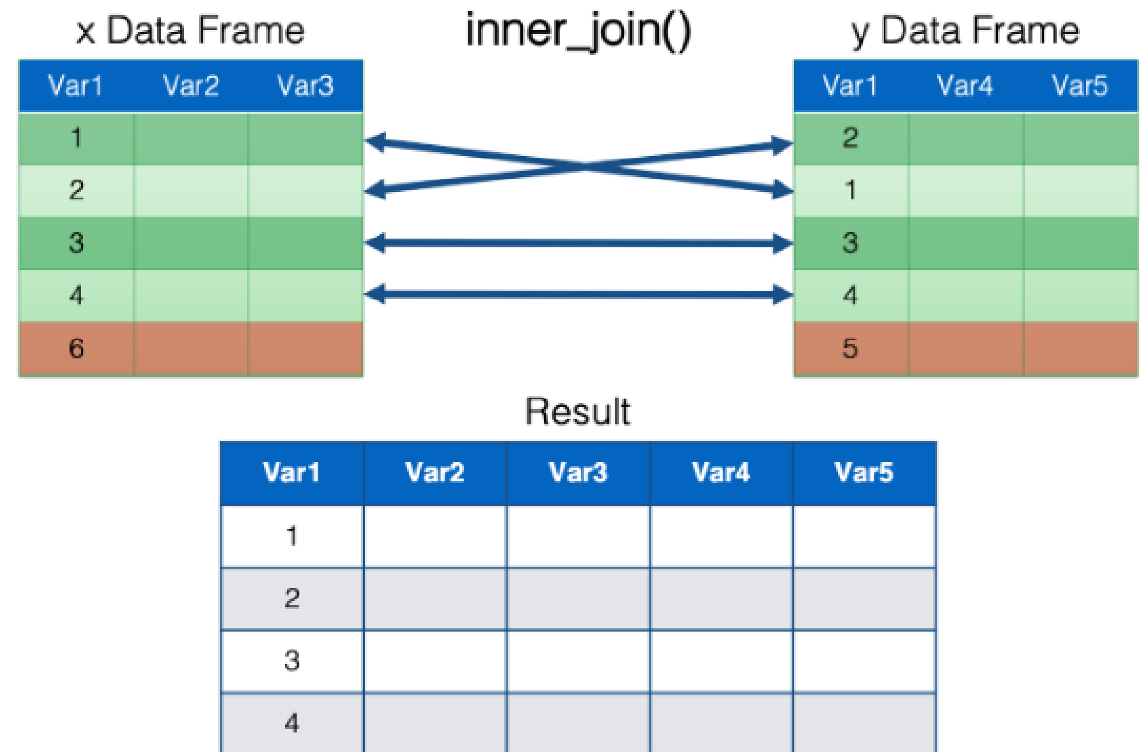
Combining Data

full_join() keeps all the rows from both data frames in the resulting combined data frame. The values for columns with no key value match in the opposite data frame are set to NA



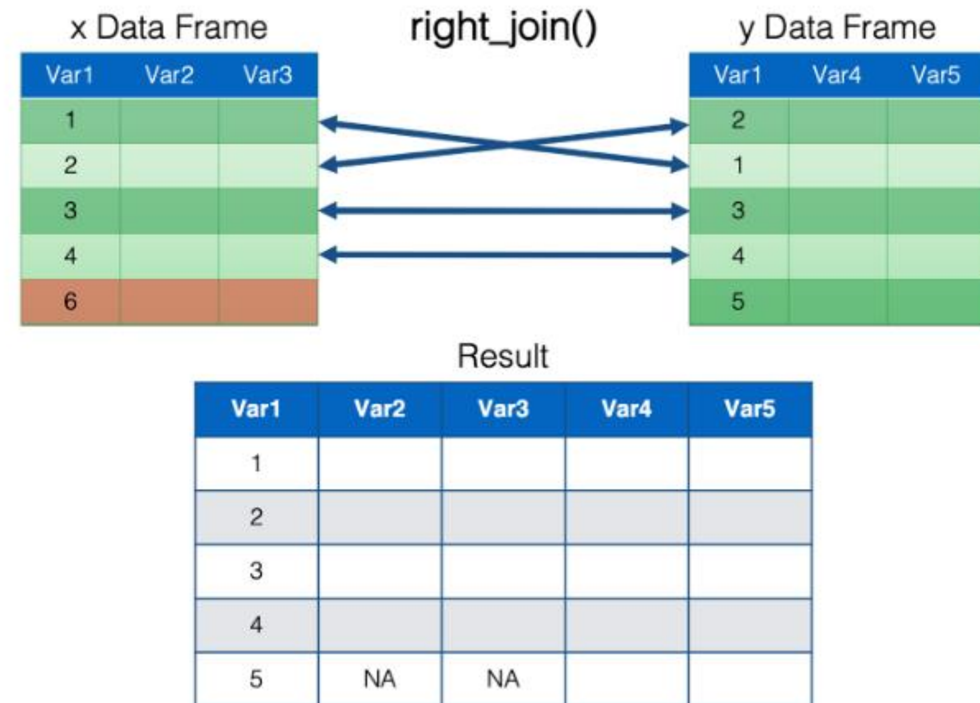
Combining Data

inner_join() keeps only the rows from both data frames that have a key value match in the opposite data frame in the resulting combined data frame.



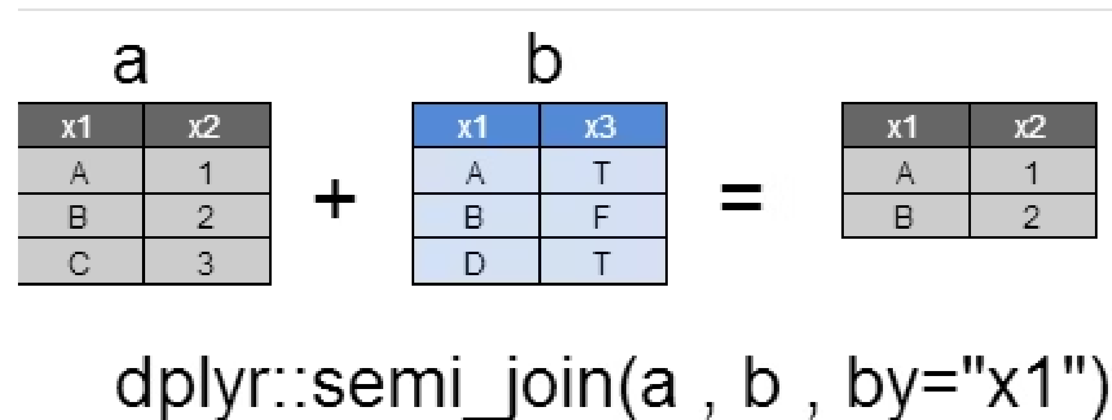
Combining Data

right_join() keeps all the rows from the y data frame in the resulting combined data frame, and only keep the rows from the x data frame that have a key value match in the y data frame. The values for columns with no key value match in the opposite data frame are set to NA.



Combining Data

- **semi_join()** is a 'Filtering Join" to filter one table against the rows of another.
- Provides a list of unique rows from the left data frame that have matching rows in the right data frame.
- The result contains all columns from the left data frame



Combining Data - Example

PARIS LODRON UNIVERSITÄT SALZBURG

- **semi_join():** to use a semi-join to filter the airports dataset to show just the origin airports:

	faa	name	lat	lon	alt	tz	dst	tzone
1	04G	Lansdowne Airport	41.13047	-80.61958	1044	-5	A	America/New_York
2	06A	Moton Field Municipal Airport	32.46057	-85.68003	264	-6	A	America/Chicago
3	06C	Schaumburg Regional	41.98934	-88.10124	801	-6	A	America/Chicago
4	06N	Randall Airport	41.43191	-74.39156	523	-5	A	America/New_York
5	09J	Jekyll Island Airport	31.07447	-81.42778	11	-5	A	America/New_York

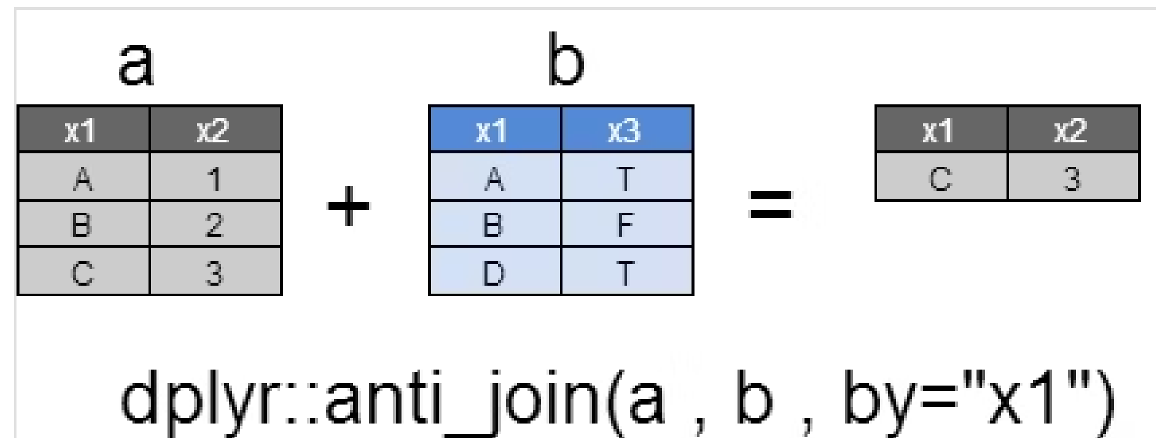
	year	time_hour	origin	dest	tailnum	carrier
1	2013	2013-01-01 05:00:00	EWR	IAH	N14228	UA
2	2013	2013-01-01 05:00:00	LGA	IAH	N24211	UA
3	2013	2013-01-01 05:00:00	JFK	MIA	N619AA	AA
4	2013	2013-01-01 05:00:00	JFK	BQN	N804JB	B6
5	2013	2013-01-01 06:00:00	LGA	ATL	N668DN	DL

```
airports %>%
  semi_join(flights2, join_by(faa == origin))
```

```
  faa   name                  lat   lon   alt    tz dst   tzone
  <chr> <chr>               <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 EWR   Newark Liberty Intl  40.7 -74.2    18    -5 A     America/New_York
2 JFK   John F Kennedy Intl  40.6 -73.8    13    -5 A     America/New_York
3 LGA   La Guardia           40.8 -73.9    22    -5 A     America/New_York
```

Combining Data

- **anti_join()** is a 'Filtering Join" to filter one table against the rows of another.
- Provides a list of unique rows from the left data frame that do not have matching rows in the right data frame
- The result contains all columns from the left data frame



Exercise 2.1

For each plane, determine the temperature and wind speed when it departed.

Please use columns from the table **flight:**

year, time_hour, origin, dest, tailnum, carrier

And from the table **weather:**

origin, time_hour, temp, wind_speed

Exercise 2.2

Find rows that are missing from airports by looking for flights that don’t have a matching destination airport .
Hint: use anti_join; dest == faa, distinct

Please use columns from the table **flight:**

year, time_hour, origin, dest, tailnum, carrier

PARIS
LODRON
UNIVERSITÄT
SALZBURG

Custom function

Define custom mutate(), filter(), arrange(), summarize() functions and reuse them.

```
# Define custom method
filter.flights <- function(.data, min_distance, max_distance) {
    filtered_data <- filter(.data, between(distance, min_distance, max_distance))
    return(filtered_data)
}
# Call custom methods
filtered_flights <- flights %>% filter.flights(1000, 2000)
```

lubridate

PARIS LODRON UNIVERSITÄT SALZBURG

When do we work with date-time data?

PARIS LODRON UNIVERSITÄT SALZBURG

Every time we track events or measure/calculate duration of activities.

Examples:

- Track transactions
- Analyze intervals of vulcanic eruptions
- Collect timestamps from various IoT sensors

Time can be tricky.

PARIS
LODRON
UNIVERSITÄT
SALZBURG

Does every year have 365 days? → leap years

Does every day have 24 hours? → daylight saving

Does every minute have 60 seconds? → leap seconds

Does everyone use the same format for date and time? → time zones, local differences

What is lubridate?

- Package in the tidyverse ecosystem
- Provides functions and methods for easily creating, manipulating, and extracting information from date-time data
- Robust to leap years, daylight savings times, leap seconds and time zones

Timestamps

Three types of data describing date and/or time:

1. A date → YYYY-MM-DD
2. A time → HH:MM:SS
3. A date-time (date + time) → YYYY-MM-DDTHH:MM:SS

Each stored as the number of days behind 1970-01-01 UTC and seconds behind 00:00:00.

Parse Date-times

PARIS LODRON UNIVERSITÄT SALZBURG

- Timestamps are often stored as strings
- Lubridate provides methods to parse different strings or numbers into date-time objects
- If not declared otherwise, UTC time zone is assumed

```
"2024-04-09T18:15:00"
"2024-09-04 18:15:00", tz="CET"
"09/04/2024 6:15pm"
"9th of April ‘24"
"04-2024"
```

Parse Date-times

Method as order of

year (y), month (m), day (d), hour (h), minute (m), second (s)

```
ymd_hms("2024-04-09T18:15:00")                    #[1]  "2024-04-09 18:15:00 UTC"
ydm_hms("2024-09-04 18:15:00", tz="CET") #[1] "2024-04-09 18:15:00 CEST""
dmy_hm("09/04/2024 6:15pm")                       #[1] "2024-04-09 18:15:00 UTC"
dmy("9th of April '24")                           #[1] "2024-04-09"
my("04-2024")                                     #[1] "2024-04-01"
```

Parse Date-times

```
parse_date_time(x, orders)                        #custom order
make_datetime(year, month, day, hour, ...)        #create date-time

today( )                #get current date
now( )                  #get current date-time

as.numeric(ymd("19700102"))                       #[1] 1
as.numeric(ymd_hms("19700101 00:00:05"))          #[1] 5
```

Get and Set Date-times

PARIS LODRON UNIVERSITÄT SALZBURG

As soon as we have our data in date-time format, we can get and set components:

```
> dt        #[1] "2024-04-09 18:15:00 UTC"
```

```
# Getter
> date(dt)          #[1] "2024-04-09"
> year(dt)          #[1] 2024
> month(dt)         #[1] 4
> hour(dt)          #[1] 18
> am(dt)            #[1] FALSE
> leap_year(dt)     #[1] TRUE
```

```
# Setter
> year(dt) <- 2025
> dt        #[1] "2025-04-09 18:15:00 UTC"
> month(dt) <- 1
> dt        #[1] "2025-01-09 18:15:00 UTC"
> hour(dt) <- 15
> dt        #[1] "2025-01-09 15:15:00 UTC"
```

Exercise 3.1 – date-time

PARIS LODRON UNIVERSITÄT SALZBURG

Look at the provided dataset 'flights'.

1. Parse the date and departure time to a date-time object and store it in the new column 'departure'.
2. Look at the flight in the first row.
3. Get the year of the flight. Was the year a leap year?
4. On what weekday did the flight depart?

Time zones

PARIS LODRON UNIVERSITÄT SALZBURG

- Default time zone in R is UTC = Coordinated Universal Time

  → Has no Daylight Saving Time
- CE(S)T = Central European (Summer) Time
- R incorporates time zones as <continent>/<city> and some abbreviations

Time zones

```
OlsonNames()                     #returns all available time zones

Sys.timezone()                   #[1] "Europe/Berlin“
dt <- ymd_hms("2024-04-09 18:00:00", tz="Europe/Berlin")


# display time in different time zone
with_tz(dt, tzone = "US/Eastern")          #[1] "2024-04-09 12:00:00 EDT“
# change underlying time
force_tz(dt, tzone = "US/Eastern")         #[1] "2024-04-09 18:00:00 EDT"
```

Exercise 3.2 – time zones

Look at the provided dataset 'flights'. As in 3.1, look at the flight in the first row.

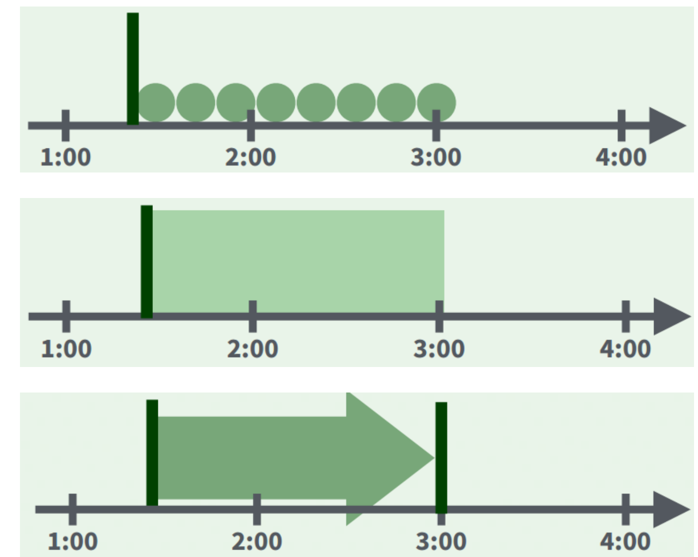
1. What is the departure time of the flight in your current time zone?
2. Does this expression convert the time zones properly? When would you use *force_tz()*?

```
departure_my_tz <- force_tz(departure, tzone = "Europe/Berlin")
with_tz(departure_my_tz, tzone = "America/New_York")
```

Time Spans

Lubridate introduces three new time span classes:

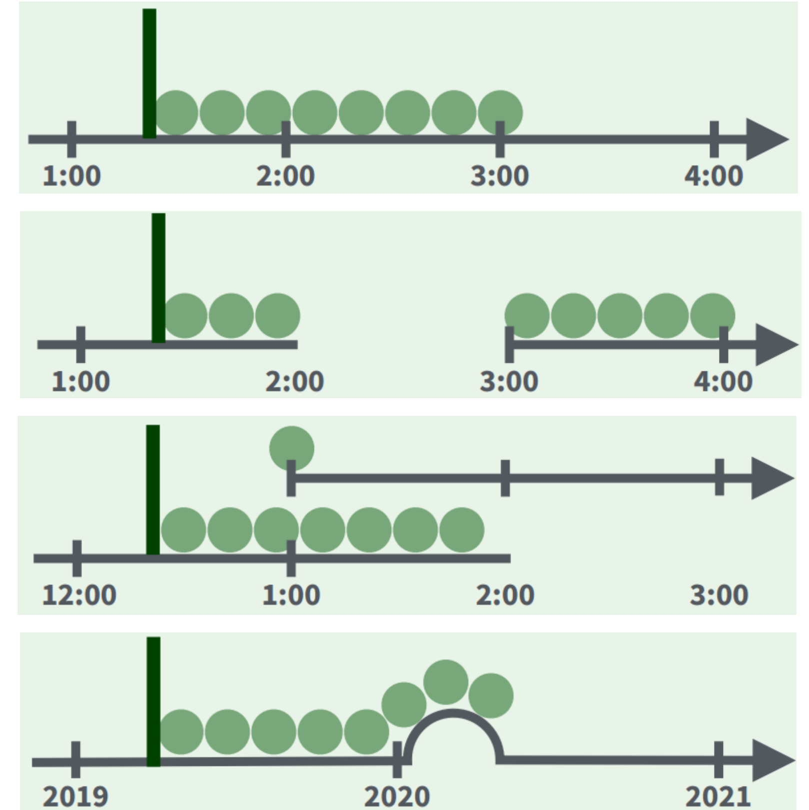
1. **Durations** measure the seconds starting from a starting point
2. **Periods** track changes in clock times from a starting point
3. **Intervals** are timespans between two distinct points in time



Graphic from https://lubridate.tidyverse.org/index.html

Durations

- Represent a fixed length of time measured in seconds
- Don't adjust for leap years, leap seconds, DLS and varying month lengths



Graphic from https://lubridate.tidyverse.org/index.html

Durations

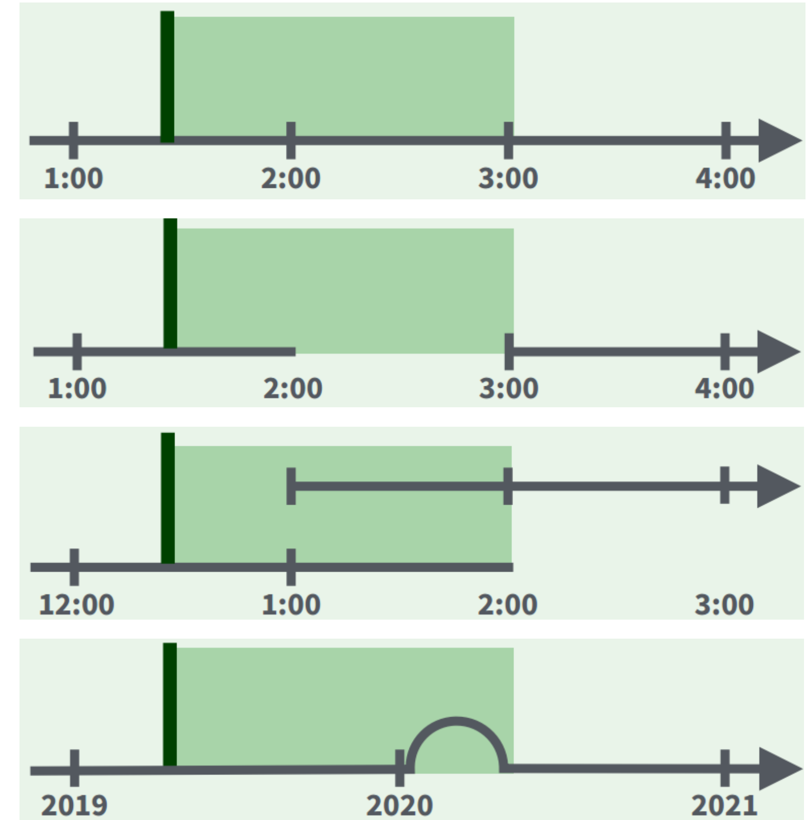
Helper functions are called as “d” + the pluralized time unit (dyears, dhours, …) :

```
dyears(1)                                    #[1] "31557600s (~1 years)"
dmonths(1)                                   #[1] "2629800s (~4.35 weeks)"
duration(num = 1, units = "months")          #[1] "2629800s (~4.35 weeks)"
as.duration(ymd("2024-04-09")- ymd("2024-04-04"))
                                             #[1] "432000s (~5 days)“
ymd_hms("2024-04-09 18:00:00", tz="UTC") - dmonths(1)
                                             #[1] "2024-03-10 07:30:00 UTC"
```

PARIS LODRON UNIVERSITÄT SALZBURG

Periods

- Represent a relative amount of time measured in “human” units
- Adjust for leap years, leap seconds, DLS and varying month lengths



Graphic from https://lubridate.tidyverse.org/index.html

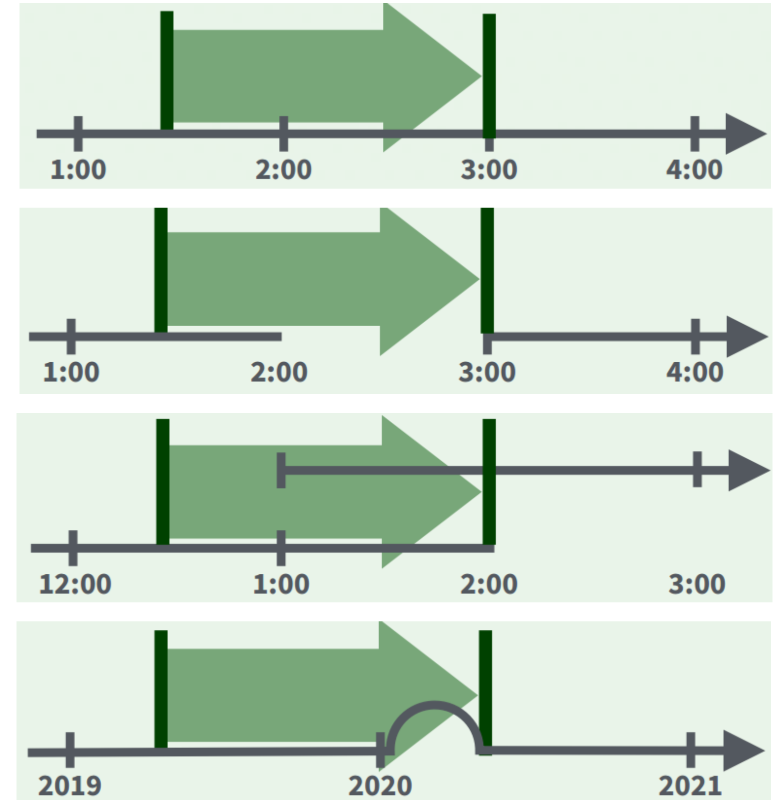
Periods

Helper functions are called as the pluralized time unit (years, hours, ...) :

```
years(1)                                          #[1] "1y 0m 0d 0H 0M 0S“
months(1)                                         #[1] "1m 0d 0H 0M 0S"
period(num = 1, units = "months")                 #[1] "1m 0d 0H 0M 0S"
as.period(ymd("2024-04-09")- ymd("2024-04-04"))
                                                  #[1] "5d 0H 0M 0S"
ymd_hms("2024-04-09 18:00:00", tz="UTC") - months(1)
                                                  #[1] "2024-03-09 18:00:00 UTC"
```

Intervals

- Represent a specific time span between two distinct points in time
- Adjust for leap years, leap seconds, DLS and varying month lengths
- Allows for precise divisions with periods and durations



Graphic from https://lubridate.tidyverse.org/index.html

Intervals

```
years(1) / days(1)                          #[1] 365.25
dyears(1) / ddays(1)                        #[1] 365.25

start_date <- ymd("2023-04-09")
end_date <- ymd("2024-04-09")
i <- interval(start_date, end_date)         #i is 2023-04-09 UTC--2024-04-09 UTC
i / ddays(1)                                #[1] 366
i <- int_shift(i, years(1))
i / ddays(1)                                #[1] 365
```

Exercise 3.3 - time spans

PARIS LODRON UNIVERSITÄT SALZBURG

Look at the provided dataset 'flights'.

1. The duration of each flight is given in minutes by 'air_time'. Calculate the arrival time of each flight and store it in the new column 'departure'.
2. Create an Interval for each flight from departure to arrival and store it in the new column 'flight_duration'.
3. The flight in the first row got delayed by three hours. Adapt the interval accordingly.
4. Bonus: Do the flights in row 1 and row 2 overlap? Find a suiting method.

Sources (lubridate)

- https://lubridate.tidyverse.org/
- https://r4ds.had.co.nz/dates-and-times.html
- https://www.datascienceverse.com/data-engineering/lubridate-in-r-practical-guide-to-handling-and-analyzing-date-time-data/
- https://cran.r-project.org/web/packages/lubridate/lubridate.pdf
- https://nycflights13.tidyverse.org/reference/flights.html
- http://www.trutschnig.net/Slides_WR_03.pdf

Last accessed on 07.04.2024

Sources (dplyr)

PARIS LODRON UNIVERSITÄT SALZBURG

- https://www.r4epi.com/working-with-multiple-data-frames.html
- https://r4ds.hadley.nz/joins
- https://dplyr.tidyverse.org/articles/two-table.html
- https://nyu-cdsc.github.io/learningr/assets/data-transformation.pdf
- https://md.psych.bio.uni-goettingen.de/mv/unit/dplyr/dplyr.html

PARIS
LODRON
UNIVERSITÄT
SALZBURG

Backup slides

Exercise 4 – dplyr + lubridate

PARIS LODRON UNIVERSITÄT SALZBURG

Look at the provided dataset 'flights'.

What departure times are (un)popular? Plot the distribution of flight times change over the course of the day.

Hints:

1. Use pipes to concatenate methods.
2. For each flight, extract the hour of the departure time.
3. Group by hour.

Time Spans

Allowed arithmetic operations:

	date	date time	duration	period	interval	number
date	-		- +	- +		- +
date time		-	- +	- +		- +
duration	- +	- +	- + /			- + × /
period	- +	- +		- +		- + × /
interval			/	/		
number	- +	- +	- + ×	- + ×	- + ×	- + × /

Graphic from https://r4ds.had.co.nz/dates-and-times.html

Stamp Date-times

PARIS LODRON UNIVERSITÄT SALZBURG

```
> my_stamp <- stamp("Presentation held on Sunday, Jan 17th, 1999 10:43")
> my_stamp(now())
#[1] "Presentation held on Sunday, Apr 05th, 2024 21:02"
```