Hw-data_science lubridate and purrr

Ignacio Cortés Niilus

2025-02-25

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(purrr)
```

```
#Create sequence of dates from2015 to 2025
date_seq <- seq(ymd("2015-01-01"), ymd("2025-12-31"), by = "2 months")

#
extracted_data <- data.frame(
  Date = date_seq,
  Year = year(date_seq),
  Quarter = quarter(date_seq),
  ISO_Week = isoweek(date_seq)
)

print(extracted_data)
```

Exercise 1: Advanced Date Manipulation with lubridate

```
##          Date Year Quarter ISO_Week
## 1  2015-01-01 2015       1        1
## 2  2015-03-01 2015       1        9
## 3  2015-05-01 2015       2       18
## 4  2015-07-01 2015       3       27
## 5  2015-09-01 2015       3       36
## 6  2015-11-01 2015       4       44
## 7  2016-01-01 2016       1       53
## 8  2016-03-01 2016       1        9
## 9  2016-05-01 2016       2       17
## 10 2016-07-01 2016       3       26
```

```
## 11 2016-09-01 2016       3       35
## 12 2016-11-01 2016       4       44
## 13 2017-01-01 2017       1       52
## 14 2017-03-01 2017       1        9
## 15 2017-05-01 2017       2       18
## 16 2017-07-01 2017       3       26
## 17 2017-09-01 2017       3       35
## 18 2017-11-01 2017       4       44
## 19 2018-01-01 2018       1        1
## 20 2018-03-01 2018       1        9
## 21 2018-05-01 2018       2       18
## 22 2018-07-01 2018       3       26
## 23 2018-09-01 2018       3       35
## 24 2018-11-01 2018       4       44
## 25 2019-01-01 2019       1        1
## 26 2019-03-01 2019       1        9
## 27 2019-05-01 2019       2       18
## 28 2019-07-01 2019       3       27
## 29 2019-09-01 2019       3       35
## 30 2019-11-01 2019       4       44
## 31 2020-01-01 2020       1        1
## 32 2020-03-01 2020       1        9
## 33 2020-05-01 2020       2       18
## 34 2020-07-01 2020       3       27
## 35 2020-09-01 2020       3       36
## 36 2020-11-01 2020       4       44
## 37 2021-01-01 2021       1       53
## 38 2021-03-01 2021       1        9
## 39 2021-05-01 2021       2       17
## 40 2021-07-01 2021       3       26
## 41 2021-09-01 2021       3       35
## 42 2021-11-01 2021       4       44
## 43 2022-01-01 2022       1       52
## 44 2022-03-01 2022       1        9
## 45 2022-05-01 2022       2       17
## 46 2022-07-01 2022       3       26
## 47 2022-09-01 2022       3       35
## 48 2022-11-01 2022       4       44
## 49 2023-01-01 2023       1       52
## 50 2023-03-01 2023       1        9
## 51 2023-05-01 2023       2       18
## 52 2023-07-01 2023       3       26
## 53 2023-09-01 2023       3       35
## 54 2023-11-01 2023       4       44
## 55 2024-01-01 2024       1        1
## 56 2024-03-01 2024       1        9
## 57 2024-05-01 2024       2       18
## 58 2024-07-01 2024       3       27
## 59 2024-09-01 2024       3       35
## 60 2024-11-01 2024       4       44
## 61 2025-01-01 2025       1        1
## 62 2025-03-01 2025       1        9
## 63 2025-05-01 2025       2       18
## 64 2025-07-01 2025       3       27
```

```
## 65 2025-09-01 2025       3       36
## 66 2025-11-01 2025       4       44
```

In this exercise, I generated a sequence of dates from January 1, 2015, to December 31, 2025, spaced every two months. I extracted the year, quarter, and ISO week number from each date using the ***lubridate*** *package. This allows for structured date analysis while keeping track of time-based classifications.*

```r
sample_dates <- ymd(c("2018-03-15", "2020-07-20", "2023-01-10", "2025-09-05"))

# compute diff in months
diff_months <- as.numeric(interval(sample_dates
          [-length(sample_dates)],
          sample_dates[-1]) / months(1))

#compute diff in weeks
diff_weeks <- as.numeric(difftime(sample_dates
  [-1], sample_dates[-length(sample_dates)],
  units = "weeks"))

date_diff <- data.frame(
  Start_Date = sample_dates[-length(sample_dates)],
  End_Date = sample_dates[-1],
  Difference_Months = diff_months,
  Difference_Weeks = diff_weeks
)

print(date_diff)
```

Exercise 2: Complex Date Arithmetic

```
##   Start_Date   End_Date Difference_Months Difference_Weeks
## 1 2018-03-15 2020-07-20          28.16129         122.5714
## 2 2020-07-20 2023-01-10          29.67742         129.1429
## 3 2023-01-10 2025-09-05          31.83871         138.4286
```

I calculated the difference in months and weeks between consecutive dates in a given list. Using interval() from lubridate, I correctly computed month differences, while difftime() helped determine week differences. This approach ensures accurate time interval calculations for analyzing time-based changes.

```r
num_lists <- list(c(4, 16, 25, 36, 49), c(2.3, 5.7, 8.1, 11.4), c(10, 20, 30, 40, 50))

#mean median and sd
stats_results <- tibble::tibble(
  Mean = map_dbl(num_lists, mean),
  Median = map_dbl(num_lists, median),
  Std_Dev = map_dbl(num_lists, sd)
)

print(stats_results)
```

Exercise 3: Higher-Order Functions with purrr

```
## # A tibble: 3 x 3
##    Mean Median Std_Dev
##   <dbl>  <dbl>   <dbl>
## 1 26      25     17.4
## 2  6.88    6.9    3.84
## 3 30      30     15.8
```

In this exercise, I used the purrr package to calculate the mean, median, and standard deviation for each numeric vector in a given list. The map_dbl() function efficiently applied these statistical computations to each vector, demonstrating functional programming techniques for handling lists.

```
# Defining mixed date format list
date_strings <- list("2023-06-10", "2022/12/25", "15-Aug-2021", "InvalidDate")

safe_date <- possibly(~ parse_date_time(.x, orders = c("ymd", "dmy", "mdy")), NA_real_)

converted_dates <- map(date_strings, safe_date)
```

Exercise 4: Combining lubridate and purrr

```
## Warning: All formats failed to parse. No formats found.
```

```
month_names <- map_chr(converted_dates, ~ ifelse(is.na(.x), "Invalid", month(.x, label = TRUE)))
```

```
## Warning: Automatic coercion from integer to character was deprecated in purrr 1.0.0.
## i Please use an explicit call to `as.character()` within `map_chr()` instead.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
result_df <- tibble::tibble(
  Original_String = date_strings,
  Converted_Date = converted_dates,
  Month_Name = month_names
)
print(result_df)
```

```
## # A tibble: 4 x 3
##   Original_String Converted_Date Month_Name
##   <list>          <list>         <chr>
## 1 <chr [1]>       <dttm [1]>     6
## 2 <chr [1]>       <dttm [1]>     12
## 3 <chr [1]>       <dttm [1]>     8
## 4 <chr [1]>       <dttm [1]>     Invalid
```

I processed a list of mixed date formats and safely converted them to Date format using possibly() from purrr. This ensures that invalid dates do not cause errors. I then extracted the month name for each valid date. This approach is useful for handling inconsistent date inputs in real-world data.