Julia Janczyk

2023-12-01

Zadanie 1

```
library(gapminder)
library(tidyr)
library(dplyr)
library(nycflights13)
```

Korzystamy z biblioteki dplyr, która zapewnia funkcję dla każdego podstawowego czasownika manipulującego danymi.

Funkcje działające na wierszach to: filter(), slice() oraz arrange().

Działające na wierszach to: select(), rename(), mutate() a także relocate().

Funkcją działającą na grupie wierszy jest summarise().

Wszystkie te funkcje jako argument przyjmuja data frame.

Biblioteka udostępnia również opreator %>% - pipe - potok. Pozwala on przekazywać dane z funkcji do funkcji w łańcuchu, ułatwia ona dodawanie kroków w dowolnym miejscu operacji, uporządkuje to kod a operacje bedą działać od lewj do prawej.

Działa tak, że wyrażenie :

x %>% f(y)

oznacza :

f(x,y)

a) Wyświetl wyłącznie loty do Atlanty (ATL) z lotniska LaGuardia Nowy Jork (LGA)

```
flights %>%
  filter(dest == "ATL", origin == "LGA")
```

```
## # A tibble: 10,263 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      554            600        -6      812            837
##  2  2013     1     1      600            600         0      837            825
##  3  2013     1     1      658            700        -2      944            939
##  4  2013     1     1      754            759        -5     1039           1041
##  5  2013     1     1      814            810         4     1047           1030
##  6  2013     1     1      830            835        -5     1052           1105
```

```
##  7  2013     1     1      855            859        -4     1143           1145
##  8  2013     1     1      940            955       -15     1226           1220
##  9  2013     1     1      956           1000        -4     1241           1241
## 10  2013     1     1     1021           1023        -2     1254           1252
## # i 10,253 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Wyświetlamy wszytkie informacje na temat lotów zgodnych z wybranymi założeniami, funkcja filter() pozwala na określenie grupy wierszy jakie chcemy wybrać i zobaczyć. Argumenty odnoszą się do zmiennych w data frame, funkcja wybiera wyrażenia zgodne z podanym wzorcem.

dest - destination, szkua lotów do 'Atlanty' - ATL

origin - wyświetla te loty które startują z 'LaGuardia Nowy Jork' - LGA

b) Wyświetl zmienne: tailnum oraz wszystkie zawierające na końcu "time"

```
flights %>%
  select(tailnum, ends_with("time"))
```

```
## # A tibble: 336,776 x 6
##    tailnum dep_time sched_dep_time arr_time sched_arr_time air_time
##    <chr>      <int>          <int>    <int>          <int>    <dbl>
##  1 N14228       517            515      830            819      227
##  2 N24211       533            529      850            830      227
##  3 N619AA       542            540      923            850      160
##  4 N804JB       544            545     1004           1022      183
##  5 N668DN       554            600      812            837      116
##  6 N39463       554            558      740            728      150
##  7 N516JB       555            600      913            854      158
##  8 N829AS       557            600      709            723       53
##  9 N593JB       557            600      838            846      140
## 10 N3ALAA       558            600      753            745      138
## # i 336,766 more rows
```

Wybieramy kolumny zawierające zmienną "tailnum" oraz kolumny kończące się na "time". Funkcja select() wyświetla konkretne kolumny pliku - interesujące nas zmienne określone danymi warunkami. Możemy wyświetlić konkretne kolumny np. tailnum, czy też kolumny zgodne z danym wzorcem - np. starts_with(), ends_with(),

contains()-(wybiera te które zawierają podane słowo czy znaki),

matches()-(wybiera te które pasują do wyrażenia).

Dodatkowo dostępne są operatory:

: wybiera zakres od do

! - neguje wcześniejszą sekcję

& - operator "i", łączy podane założenia, mają się oba wykonać

| - operator "lub", zachodzi przynajmniej jedno z podanych założeń

c) Oblicz czas lotu i zapisz jako time_in_air, wyświetl tylko informacje o numerze lotu,czasie wylotu i czasie przylotu oraz nowo utworzoną zmienną

```
flights %>%
  mutate(time_in_air = arr_time - dep_time) %>% # nowa zmienna wyliczona na podstawie wybranych kolumn
  select(flight,arr_time, dep_time, time_in_air ) # wyświetlamy wybrane informacje
```

```
## # A tibble: 336,776 x 4
##    flight arr_time dep_time time_in_air
##     <int>    <int>    <int>       <int>
##  1   1545      830      517         313
##  2   1714      850      533         317
##  3   1141      923      542         381
##  4    725     1004      544         460
##  5    461      812      554         258
##  6   1696      740      554         186
##  7    507      913      555         358
##  8   5708      709      557         152
##  9     79      838      557         281
## 10    301      753      558         195
## # i 336,766 more rows
```

Funkcją select() wyświetlamy tylko pożądane informacje, numer lotu - flight, czas wylotu - dep_time, oraz przylotu - arr_time, za pomocą funkcji mutate() obliczamy czas lotu - time_in_air, który jest różnicą czasu przylotu i czasu odlotu.

Tworzy ona nowe kolumny na podstawie innych wybranych kolumn, oblicza z nich nową zmienną

d) Wyświetl wyłącznie wiersze 1300 i 1400 z danych

```
flights %>%
  slice(1300, 1400)
```

```
## # A tibble: 2 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     2     1352           1355        -3     1635           1709
## 2  2013     1     2     1540           1545        -5     2047           2039
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Funkcja slice() wybiera konkretne wiersze, określajac je po wartości ich pozycji.

e) Wyświetl numer samolotu (tailnum), lotnisko wylotu oraz lotnisko docelowe dla lotów, których występuje opóźnienie po przylocie i jest ono minimum dziesięciokrotnie dłuższe niż opóźnienie wylotu

```
flights %>%
  filter(arr_delay > 0) %>% # pozbywamy się z danych wartości mniejszych od 0,
  filter(dep_delay > 0) %>% # które oznaczają wcześniejszy przylot lub odlot
  filter(arr_delay > 10*dep_delay) %>%
  select(tailnum, origin, dest)
```

```
## # A tibble: 2,250 x 3
##    tailnum origin dest
##    <chr>   <chr>  <chr>
##  1 N619AA  JFK    MIA
##  2 N779JB  JFK    LAX
##  3 N841UA  EWR    ORD
##  4 N521JB  JFK    SAN
##  5 N479UA  LGA    ORD
##  6 N722TW  JFK    SFO
##  7 N24212  EWR    AUS
##  8 N3DUAA  LGA    DFW
##  9 N14250  EWR    LAS
## 10 N418UA  LGA    ORD
## # i 2,240 more rows
```

Funkcja filter() wybiera wiersze na podstawie wartości w podanych kolumnach, spełniających wyrażenie logiczne.

Szukamy opóźnień po przylocie 10 razy większych niż opóźnienie wylotu, korzystamy z kolumn opóźnienie przylotu - arr_delay oraz oóźnienie wylotu -dep_delay

Wyświetlamy z kolumny: numer samolotu - tailnum, lotnisko wylotu - origin, kierunek lotu - dest i otrzymujemyu loty z dziesięciokrotnie większym opóźnienem przylotu niż odlotu

f) Sprawdź czy w danych występują wartości NA dla tailnum. Jeśli występują zapisz wszystkie rekordy z NA do flight_na_tailnum

```
flight_na_tailnum <- flights %>%
  filter(is.na(tailnum) == 1)
```

Funkcja is.na() sprawdza czy dane zawierają wartość niedostępną NA, gdy posiada takie wartości to wartość logiczna tego wyrażenia bedzie równa 1.

Funkcją filter() wybieramy te wiersze, które spełniaja kolejny warunek logiczny (is.na() == 1), czyli te zawierające NA.

g) Policz liczbę lotów oraz statystyki dla zmiennej distance: wartość średnią, minimum i maksimum dla lotów wykonanych w poszczególnych miesiącach

```
flights %>%
  group_by(month) %>%
  summarise(liczba_lotów=n(), średni_dystans=mean(distance),
            minimalny_dystans=min(distance), maksymalny_dystans=max(distance))
```

```
## # A tibble: 12 x 5
##    month liczba_lotów średni_dystans minimalny_dystans maksymalny_dystans
##    <int>        <int>          <dbl>              <dbl>               <dbl>
##  1     1        27004          1007.                 80                4983
##  2     2        24951          1001.                 80                4983
##  3     3        28834          1012.                 80                4983
##  4     4        28330          1039.                 80                4983
##  5     5        28796          1041.                 94                4983
##  6     6        28243          1057.                 94                4983
##  7     7        29425          1059.                 17                4983
##  8     8        29327          1062.                 94                4983
##  9     9        27574          1041.                 94                4983
## 10    10        28889          1039.                 94                4983
## 11    11        27268          1050.                 94                4983
## 12    12        28135          1065.                 94                4983
```

Funkcją group_by() segregujemy dane miesiącami, dzięki czemu operacje wykonywane są dla każdej grupy osobno.

Summarise oblicza statystki dla zmiennej w grupach poszczególnych miesięcy. Tworzy uporządkowaną tabelę podsumowań przekazanych wartości.

n() - funkcja liczy ilość wierszy - zlicza liczbę lotów w każdym miesiącu

h) Wyświetl 3 loty o największym opóźnieniu dla każdego typu samolotu

```
flights %>%
  group_by(carrier) %>% # za typ samolotu przyjumjemy carrier, grupowanie
  arrange(desc(arr_delay), .by_group = TRUE) %>%
  slice(1:3) %>%
  select(arr_delay, carrier) # wyświetlamy wybrane informacje
```

```
## # A tibble: 48 x 2
## # Groups:   carrier [16]
##    arr_delay carrier
##        <dbl> <chr>
##  1       744 9E
##  2       458 9E
##  3       421 9E
##  4      1007 AA
##  5       878 AA
##  6       852 AA
##  7       198 AS
##  8       196 AS
##  9       188 AS
## 10       497 B6
## # i 38 more rows
```

Funkcja arrange() porządkuje wiersze według kolejności wartości wybranych kolumn, w tym przypadku porządkujemy wiersze według opóźnieniu samolotu - arr_delay, desc() oznacza kolejność malejącą.

Wiersze uporządkowane są więc rosnąco przez wartości arr_delay.

Aby funkcja zwracała uwagę na grupowanie używamy ".by_group = TRUE". Dzięki temu mamy pewność, że wiersze są pogrupowane po typie samolotu - carrier.

Funkcją slice() obcinamy 3 pierwsze wiersze, czyli wiersze o największej wartości opóźnienia dla każdego typu samolotu

i) Dla poszczególnych typów samolotów (carrier) oblicz liczbę lotów i średni czas w powietrzu. Następnie wybierz wyłącznie te wyniki do wyświetlenia dla których liczba lotów jest wyższa niż 700 oraz średni czas w powietrzu jest wyższy od 100. Wyniki wyświetl w kolejności malejącej dla liczby lotów.

```
flights %>%
  group_by(carrier) %>%
  filter(n() > 700, mean(air_time, na.rm = TRUE) > 100) %>%
  arrange(desc(n())) %>%
  summarise(liczba_lotów=n(), średni_czas=mean(air_time, na.rm = TRUE))
```

```
## # A tibble: 8 x 3
##   carrier liczba_lotów średni_czas
##   <chr>          <int>       <dbl>
## 1 AA             32729        189.
## 2 AS               714        326.
## 3 B6             54635        151.
## 4 DL             48110        174.
## 5 FL              3260        101.
## 6 UA             58665        212.
## 7 VX              5162        337.
## 8 WN             12275        148.
```

Grupujemy pownieważ wyniki mają odnosić się do poszczególnych typów samolotów - carrier, w funkcji group_by() nazwywamy kolumnę grup jako typ_samolotu.

Funkcją filter() wybieramy wyniki, gdzie liczba lotów jest większa niż 700 a średni czas w powietrzu wiekszy niż 100.

Uwzględniamy to, że mogą pojawić się wartości NA, stosujemy ich pominięcie (na.rm = TRUE).

Sortujemy wyniki malejąco i wyświetlamy ilość lotów oraz średni czas w powietrzu.

j) Znajdź informacje o funkcji slice_sample a następnie wylosuj 50 obserwacji ze zbioru i zapisz do flights_sample

```
flights_sample <- flights %>%
  slice_sample(n=50)
flights_sample
```

```
## # A tibble: 50 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     5    18      757            756         1      953            954
##  2  2013     4    21     2026           2029        -3     2356              4
```

```
##  3  2013     7    12      554            600        -6      650            715
##  4  2013    12     9      940            925        15     1313           1305
##  5  2013    11    21     1629           1630        -1     1917           1915
##  6  2013    12    26     1528           1535        -7     1903           1857
##  7  2013     7    31      824            829        -5     1024           1028
##  8  2013     7     2     2218           2126        52       51             15
##  9  2013     8    23     2133           2130         3       28           2359
## 10  2013     8    16      908            915        -7     1148           1215
## # i 40 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Funkcja slice_sample() tak jak slice() również działa na wierszach. Slice_sample() wybiera jednak losowo określoną ilość wierszy.

slice_sample(n=5, replace=FALSE)

n - oznacza ilość wierszy jaką chcemy wylosować

replace - jeśli wartość FALSE (domyślna) to każdy wiersz może być wylosowany jeden raz, nie bedzie mogł wylosować się dwa razy. Wartość TRUE oznacza więc, że jest możliwość wylosowania dwa razy tego samego wiersza z ramki danych.

k) Sprawdź ile unikatowych rekordów jest w danych

```
flights %>%
  n_distinct()
```

```
## [1] 336776
```

Funkcja n_distinct() zwraca ilość unikalnych wierszy w data frame, czyli takich z niepowtarzającymi się wartościami w kolumnach.

Może służyć też do zliczania ilości unnikalnych wartości w np. wektorze.

l) Wykonaj losowy podzbiór danych składający się z 3 rekordów i zmiennych (tailnum, dest, origin, flight) następnie zmień na postać długą danych (gather)

```
flights %>%
  slice_sample(n=3) %>%
  select(tailnum, dest, origin, flight) %>%
  gather
```

```
## # A tibble: 12 x 2
##    key     value
##    <chr>   <chr>
##  1 tailnum N328AA
##  2 tailnum N546UW
##  3 tailnum N11121
##  4 dest    LAX
```

```
##  5 dest    PHX
##  6 dest    GRR
##  7 origin  JFK
##  8 origin  JFK
##  9 origin  EWR
## 10 flight  185
## 11 flight  625
## 12 flight  4155
```

Funkcją slice_sample() losujemy 3 rekordy.

Select() wybiera zmienne które nas interesują, czyli tailnum, dest, origin, flight.

W paczce tidyr dostępna jest funkcja gather(), która pozwala zmienić postać danych na długą.

Z przekazanych danych tworzy ona dwie kolumny, gdzie pierwsza - key - jest nazwą kolumny z której dane są wyświelane, drugą - value - są ich wartości.