
Exercises with dplyr and tidyr

March 13, 2017

Enrico Tonini
enrico.tonini@quantide.com¹

¹<mailto:enrico.tonini@quantide.com>

Contents

1	Introduction	7
1.1	Introduction to <code>nycflights13</code> data	7
1.1.1	<code>flights</code>	7
1.1.2	<code>airlines</code>	9
1.1.3	<code>airports</code>	9
1.1.4	<code>planes</code>	10
1.1.5	<code>weather</code>	12
2	Verb functions	17
2.1	<code>select()</code> and its friends	17
2.1.1	Exercise 1	17
2.1.2	Exercise 2	18
2.1.3	Exercise 3	19
2.1.4	Exercise 4	19
2.1.5	Exercise 5	19
2.1.6	Exercise 6	20
2.2	<code>filter()</code> and its friends	21
2.2.1	Exercise 1	21
2.2.2	Exercise 2	21
2.2.3	Exercise 3	22
2.2.4	Exercise 4	22
2.2.5	Exercise 5	23
2.2.6	Exercise 6	23
2.3	<code>arrange()</code>	24
2.3.1	Exercise 1	24

2.3.2	Exercise 2	25
2.3.3	Exercise 3	25
2.4	<code>mutate()</code> and its friends	26
2.4.1	Exercise 1	26
2.4.2	Exercise 2	27
2.4.3	Exercise 3	27
2.4.4	Exercise 4	28
2.4.5	Exercise 5	28
2.4.6	Exercise 6	29
2.5	<code>summarise()</code>	29
2.5.1	Exercise 1	30
2.5.2	Exercise 2	30
2.5.3	Exercise 3	30
3	Grouping data	31
3.1	<code>group_by()</code>	31
3.1.1	Exercise 1	31
3.1.2	Exercise 2	32
3.1.3	Exercise 3	32
3.1.4	Exercise 4	33
3.1.5	Exercise 5	33
3.1.6	Exercise 6	33
4	Do	35
4.1	<code>do</code>	35
4.1.1	Exercise 1	35
4.1.2	Exercise 2	36
4.1.3	Exercise 3	36
4.1.4	Exercise 4	36
5	Combining data	39
5.1	Joins: <code>inner_join()</code> , <code>left_join()</code> , <code>right_join()</code> , etc.	39
5.1.1	Exercise 1	39
5.1.2	Exercise 2	40

<i>CONTENTS</i>	5
5.1.3 Exercise 3	41
5.1.4 Exercise 4	41
5.1.5 Exercise 5	42
5.1.6 Exercise 6	42
6 Tidy data with tidyr	45
6.1 tidyr	45
6.1.1 Exercise 1	45
6.1.2 Exercise 2	46
6.1.3 Exercise 3	46
6.1.4 Exercise 4	46
7 Handling Missing values	47
7.1	47
7.1.1 Exercise 1	47
7.1.2 Exercise 2	48
7.1.3 Exercise 3	48
8 Dates with lubridate	51
8.1 lubridate	51
8.1.1 Exercise 1	51
8.1.2 Exercise 2	52
8.1.3 Exercise 3	52
9 Manipulating strings with stringr	53
9.1	53
9.1.1 Exercise 1	53
9.1.2 Exercise 2	54
9.1.3 Exercise 3	54
10 Case study	57
10.1 Recap exercise	57
10.1.1 Exercise 1	57

Chapter 1

Introduction

In this document you will find some exercises with the `tidyverse` R packages. They are mainly based on the `nycflights13` data, taken from the `nycflights13` package.

1.1 Introduction to `nycflights13` data

The `nycflights13` package contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013: 336,776 flights in total.

```
require(nycflights13)
ls(pos = "package:nycflights13")

## [1] "airlines" "airports" "flights"  "planes"   "weather"
```

To help understand what causes delays, it includes a number of useful datasets:

- `flights`: information about all flights that departed from NYC
- `weather`: hourly meteorological data for each airport;
- `planes`: construction information about each plane;
- `airports`: airport names and locations;
- `airlines`: translation between two letter carrier codes and names.

1.1.1 `flights`

This dataset contains on-time data for all flights that departed from NYC (i.e. JFK, LGA or EWR) in 2013. The data frame has 16 variables and 336776 observations. The variables are organised as follow:

- Date of departure: `year`, `month`, `day`;
- Departure and arrival times (local tz): `dep_time`, `arr_time`;
- Departure and arrival delays, in minutes: `dep_delay`, `arr_delay` (negative times represent early departures/arrivals);
- Time of departure broken in to hour and minutes: `hour`, `minute`;
- Two letter carrier abbreviation: `carrier`;
- Plane tail number: `tailnum`;
- Flight number: `flight`;
- Origin and destination: `origin`, `dest`;
- Amount of time spent in the air: `air_time`;
- Distance flown: `distance`.

```
dim(flights)
```

```
## [1] 336776      16
```

```
head(flights)
```

```
##   year month day dep_time dep_delay arr_time arr_delay carrier tailnum flight
## 1 2013     1   1      517         2      830         11      UA  N14228  1545
## 2 2013     1   1      533         4      850         20      UA  N24211  1714
## 3 2013     1   1      542         2      923         33      AA  N619AA  1141
## 4 2013     1   1      544        -1     1004        -18      B6  N804JB   725
## 5 2013     1   1      554        -6      812        -25      DL  N668DN   461
## 6 2013     1   1      554        -4      740         12      UA  N39463  1696
##   origin dest air_time distance hour minute
## 1   EWR  IAH      227     1400     5      17
## 2   LGA  IAH      227     1416     5      33
## 3   JFK  MIA      160     1089     5      42
## 4   JFK  BQN      183     1576     5      44
## 5   LGA  ATL      116       762     5      54
## 6   EWR  ORD      150       719     5      54
```

```
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':   336776 obs. of  16 variables:
## $ year      : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
## $ month     : int   1 1 1 1 1 1 1 1 1 1 1 ...
## $ day       : int   1 1 1 1 1 1 1 1 1 1 1 ...
## $ dep_time  : int   517 533 542 544 554 554 555 557 557 558 ...
```



```
## $ dep_delay: num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
## $ arr_time : int  830 850 923 1004 812 740 913 709 838 753 ...
## $ arr_delay: num  11 20 33 -18 -25 12 19 -14 -8 8 ...
## $ carrier  : chr  "UA" "UA" "AA" "B6" ...
## $ tailnum  : chr  "N14228" "N24211" "N619AA" "N804JB" ...
## $ flight   : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
## $ origin   : chr  "EWR" "LGA" "JFK" "JFK" ...
## $ dest     : chr  "IAH" "IAH" "MIA" "BQN" ...
## $ air_time : num  227 227 160 183 116 150 158 53 140 138 ...
## $ distance : num  1400 1416 1089 1576 762 ...
## $ hour     : num  5 5 5 5 5 5 5 5 5 ...
## $ minute   : num  17 33 42 44 54 54 55 57 57 58 ...
```

1.1.2 airlines

This dataset contains airlines names and their respective carrier codes, it has 2 variables and 16 observations. Data structure shows that both variables involved are categorical.

```
dim(airlines)
```

```
## [1] 16  2
```

```
head(airlines)
```

```
##   carrier      name
## 1     9E Endeavor Air Inc.
## 2     AA American Airlines Inc.
## 3     AS  Alaska Airlines Inc.
## 4     B6   JetBlue Airways
## 5     DL   Delta Air Lines Inc.
## 6     EV ExpressJet Airlines Inc.
```

```
str(airlines)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':   16 obs. of  2 variables:
## $ carrier: Factor w/ 1570 levels "02Q","04Q","05Q",...: 127 143 265 305 485 551 564 584 668 903 ...
## $ name   : Factor w/ 1571 levels "40-Mile Air",...: 604 268 236 837 554 635 678 229 751 606 ...
```

1.1.3 airports

This dataset contains useful metadata about airports, that is:

- FAA airport code: `faa`;
- Usual name of the airport: `name`;
- Location of airport: `lat`, `lon`;
- Altitude (in feet): `alt`;
- Timezone offset from GMT: `tz`;
- Daylight savings time zone: `dst` A = Standard US DST: starts on the second Sunday of March, ends on the first Sunday of November U = unknown N = no dst

The data frame has 7 variables and 1397 observations.

```
dim(airports)
```

```
## [1] 1397    7
```

```
head(airports)
```

```
##   faa                name      lat      lon  alt tz dst
## 1 04G      Lansdowne Airport 41.13047 -80.61958 1044 -5  A
## 2 06A Moton Field Municipal Airport 32.46057 -85.68003 264 -5  A
## 3 06C      Schaumburg Regional 41.98934 -88.10124 801 -6  A
## 4 06N      Randall Airport 41.43191 -74.39156 523 -5  A
## 5 09J      Jekyll Island Airport 31.07447 -81.42778 11 -4  A
## 6 0A9 Elizabethton Municipal Airport 36.37122 -82.17342 1593 -4  A
```

```
str(airports)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1397 obs. of  7 variables:
## $ faa : chr  "04G" "06A" "06C" "06N" ...
## $ name: chr  "Lansdowne Airport" "Moton Field Municipal Airport" "Schaumburg Regional" "Randall Airport" ...
## $ lat : num  41.1 32.5 42 41.4 31.1 ...
## $ lon : num  -80.6 -85.7 -88.1 -74.4 -81.4 ...
## $ alt : int  1044 264 801 523 11 1593 730 492 1000 108 ...
## $ tz : num  -5 -5 -6 -5 -4 -4 -5 -5 -5 -8 ...
## $ dst : chr  "A" "A" "A" "A" ...
```

1.1.4 planes

This dataset contains plane metadata for all plane tailnumbers found in the FAA aircraft registry (American Airways (AA) and Envoy Air (MQ) report fleet numbers rather than tail numbers). The data frame has 9 variables and 3322 observations. The variables are organised as follow:

- Tail number: `tailnum`;
- Year manufactured: `year`;
- Type of plane: `type`;
- Manufacturer and model: `manufacturer`, `model`;
- Number of engines and seats: `engines`, `seats`;
- Average cruising speed in mph: `speed`;
- Type of engine: `engine`.

```
dim(planes)
```

```
## [1] 3322    9
```

```
head(planes)
```

```
##   tailnum year          type      manufacturer      model engines seats
## 1  N10156 2004 Fixed wing multi engine      EMBRAER EMB-145XR      2    55
## 2  N102UW 1998 Fixed wing multi engine AIRBUS INDUSTRIE A320-214      2   182
## 3  N103US 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214      2   182
## 4  N104UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214      2   182
## 5  N10575 2002 Fixed wing multi engine      EMBRAER EMB-145LR      2    55
## 6  N105UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214      2   182
##   speed      engine
## 1    NA Turbo-fan
## 2    NA Turbo-fan
## 3    NA Turbo-fan
## 4    NA Turbo-fan
## 5    NA Turbo-fan
## 6    NA Turbo-fan
```

```
str(planes)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':   3322 obs. of  9 variables:
## $ tailnum      : chr  "N10156" "N102UW" "N103US" "N104UW" ...
## $ year         : int   2004 1998 1999 1999 2002 1999 1999 1999 1999 1999 ...
## $ type         : chr  "Fixed wing multi engine" "Fixed wing multi engine" "Fixed wing multi engine" "Fixed wing multi engine" ...
## $ manufacturer: chr  "EMBRAER" "AIRBUS INDUSTRIE" "AIRBUS INDUSTRIE" "AIRBUS INDUSTRIE" ...
## $ model        : chr  "EMB-145XR" "A320-214" "A320-214" "A320-214" ...
## $ engines      : int   2 2 2 2 2 2 2 2 2 2 ...
## $ seats        : int   55 182 182 182 55 182 182 182 182 182 ...
## $ speed        : int   NA NA NA NA NA NA NA NA NA NA ...
## $ engine       : chr  "Turbo-fan" "Turbo-fan" "Turbo-fan" "Turbo-fan" ...
```

1.1.5 weather

This dataset is about hourly meteorological data for LGA, JFK and EWR. The data frame has 14 variables and 8719 observations. The variables are organised as follow:

- Weather station: `origin` (named `origin` to facilitate merging with `flights` data);
- Time of recording: `year`, `month`, `day`, `hour`;
- Temperature and dewpoint in F: `temp`, `dewp`;
- Relative humidity: `humid`;
- Wind direction (in degrees), speed and gust speed (in mph): `wind_dir`, `wind_speed`, `wind_gust`;
- Precipitation, in inches: `precip`;
- Sea level pressure in millibars: `pressure`;
- Visibility in miles: `visib`.

```
dim(weather)
```

```
## [1] 8719 14
```

```
head(weather)
```

```
##  origin year month day hour  temp  dewp humid wind_dir wind_speed wind_gust
## 1    EWR 2013     1   1    0 37.04 21.92 53.97     230   10.35702  11.91865
## 2    EWR 2013     1   1    1 37.04 21.92 53.97     230   13.80936  15.89154
## 3    EWR 2013     1   1    2 37.94 21.92 52.09     230   12.65858  14.56724
## 4    EWR 2013     1   1    3 37.94 23.00 54.51     230   13.80936  15.89154
## 5    EWR 2013     1   1    4 37.94 24.08 57.04     240   14.96014  17.21583
## 6    EWR 2013     1   1    6 39.02 26.06 59.37     270   10.35702  11.91865
##   precip pressure visib
## 1      0    1013.9    10
## 2      0    1013.0    10
## 3      0    1012.6    10
## 4      0    1012.7    10
## 5      0    1012.8    10
## 6      0    1012.0    10
```

```
str(weather)
```

```
## Classes 'grouped_df', 'tbl_df', 'tbl' and 'data.frame': 8719 obs. of 14 variables:
## $ origin      : chr  "EWR" "EWR" "EWR" "EWR" ...
## $ year        : num  2013 2013 2013 2013 2013 ...
## $ month       : num  1 1 1 1 1 1 1 1 1 1 ...
## $ day         : int  1 1 1 1 1 1 1 1 1 1 ...
## $ hour        : int  0 1 2 3 4 6 7 8 9 10 ...
## $ temp        : num  37 37 37.9 37.9 37.9 ...
## $ dewp        : num  21.9 21.9 21.9 23 24.1 ...
## $ humid       : num  54 54 52.1 54.5 57 ...
## $ wind_dir    : num  230 230 230 230 240 270 250 240 250 260 ...
## $ wind_speed  : num  10.4 13.8 12.7 13.8 15 ...
## $ wind_gust   : num  11.9 15.9 14.6 15.9 17.2 ...
## $ precip      : num  0 0 0 0 0 0 0 0 0 0 ...
## $ pressure    : num  1014 1013 1013 1013 1013 ...
## $ visib       : num  10 10 10 10 10 10 10 10 10 10 ...
## - attr(*, "vars")=List of 3
## ..$ : symbol month
## ..$ : symbol day
## ..$ : symbol hour
## - attr(*, "indices")=List of 8719
## ..$ : int 0
## ..$ : int 1
## ..$ : int 2
## ..$ : int 3
## ..$ : int 4
## ..$ : int 5
## ..$ : int 6
## ..$ : int 7
## ..$ : int 8
## ..$ : int 9
## ..$ : int 10
## ..$ : int 11
## ..$ : int 12
## ..$ : int 13
## ..$ : int 14
## ..$ : int 15
## ..$ : int 16
## ..$ : int 17
## ..$ : int 18
## ..$ : int 19
## ..$ : int 20
## ..$ : int 21
## ..$ : int 22
## ..$ : int 23
## ..$ : int 24
## ..$ : int 25
## ..$ : int 26
## ..$ : int 27
## ..$ : int 28
```

```
## ..$ : int 29
## ..$ : int 30
## ..$ : int 31
## ..$ : int 32
## ..$ : int 33
## ..$ : int 34
## ..$ : int 35
## ..$ : int 36
## ..$ : int 37
## ..$ : int 38
## ..$ : int 39
## ..$ : int 40
## ..$ : int 41
## ..$ : int 42
## ..$ : int 43
## ..$ : int 44
## ..$ : int 45
## ..$ : int 46
## ..$ : int 47
## ..$ : int 48
## ..$ : int 49
## ..$ : int 50
## ..$ : int 51
## ..$ : int 52
## ..$ : int 53
## ..$ : int 54
## ..$ : int 55
## ..$ : int 56
## ..$ : int 57
## ..$ : int 58
## ..$ : int 59
## ..$ : int 60
## ..$ : int 61
## ..$ : int 62
## ..$ : int 63
## ..$ : int 64
## ..$ : int 65
## ..$ : int 66
## ..$ : int 67
## ..$ : int 68
## ..$ : int 69
## ..$ : int 70
## ..$ : int 71
## ..$ : int 72
## ..$ : int 73
## ..$ : int 74
## ..$ : int 75
## ..$ : int 76
## ..$ : int 77
```

```
## ..$ : int 78
## ..$ : int 79
## ..$ : int 80
## ..$ : int 81
## ..$ : int 82
## ..$ : int 83
## ..$ : int 84
## ..$ : int 85
## ..$ : int 86
## ..$ : int 87
## ..$ : int 88
## ..$ : int 89
## ..$ : int 90
## ..$ : int 91
## ..$ : int 92
## ..$ : int 93
## ..$ : int 94
## ..$ : int 95
## ..$ : int 96
## ..$ : int 97
## ..$ : int 98
## .. [list output truncated]
## - attr(*, "group_sizes")= int 1 1 1 1 1 1 1 1 1 1 ...
## - attr(*, "biggest_group_size")= int 1
## - attr(*, "labels")='data.frame': 8719 obs. of 3 variables:
## ..$ month: num 1 1 1 1 1 1 1 1 1 1 ...
## ..$ day : int 1 1 1 1 1 1 1 1 1 1 ...
## ..$ hour : int 0 1 2 3 4 6 7 8 9 10 ...
## ..- attr(*, "vars")=List of 3
## .. ..$ : symbol month
## .. ..$ : symbol day
## .. ..$ : symbol hour
```


Chapter 2

Verb functions

In this section you will find exercises on the basic verbs of data manipulating provided by `dplyr`:

1. `select()`;
2. `filter()`;
3. `arrange()`;
4. `mutate()`;
5. `summarise()`.

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without  
## 'methods' package
```

2.1 `select()` and its friends

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)  
require(nycflights13)
```

```
## Loading required package: nycflights13
```

2.1.1 Exercise 1

Extract the following information about flights:

- month;

- day;
- air_time;
- distance.

```
# require(nycflights13)
data(flights)

flights %>% select(month, day, air_time, distance)

## # A tibble: 336,776 × 4
##   month   day air_time distance
##   <int> <int>   <dbl>   <dbl>
## 1     1     1     227     1400
## 2     1     1     227     1416
## 3     1     1     160     1089
## 4     1     1     183     1576
## 5     1     1     116      762
## 6     1     1     150      719
## 7     1     1     158     1065
## 8     1     1      53      229
## 9     1     1     140      944
## 10    1     1     138      733
## # ... with 336,766 more rows
```

2.1.2 Exercise 2

Extract all information about flights except hour and minute.

```
flights %>% select(-hour, -minute)

## # A tibble: 336,776 × 17
##   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     517             515           2     830             819
## 2  2013     1     1     533             529           4     850             830
## 3  2013     1     1     542             540           2     923             850
## 4  2013     1     1     544             545          -1    1004            1022
## 5  2013     1     1     554             600          -6     812             837
## 6  2013     1     1     554             558          -4     740             728
## 7  2013     1     1     555             600          -5     913             854
## 8  2013     1     1     557             600          -3     709             723
## 9  2013     1     1     557             600          -3     838             846
## 10 2013     1     1     558             600          -2     753             745
## # ... with 336,766 more rows, and 9 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, time_hour <dtm>
```

2.1.3 Exercise 3

Select all variables whose name ends in “time”.

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

```
## # A tibble: 336,776 × 5
```

	dep_time	sched_dep_time	arr_time	sched_arr_time	air_time
	<int>	<int>	<int>	<int>	<dbl>
## 1	517	515	830	819	227
## 2	533	529	850	830	227
## 3	542	540	923	850	160
## 4	544	545	1004	1022	183
## 5	554	600	812	837	116
## 6	554	558	740	728	150
## 7	555	600	913	854	158
## 8	557	600	709	723	53
## 9	557	600	838	846	140
## 10	558	600	753	745	138

```
## # ... with 336,766 more rows
```

2.1.4 Exercise 4

Select all variables whose name contains the word “delay”.

```
flights %>% select(matches("delay"))
```

```
## # A tibble: 336,776 × 2
```

	dep_delay	arr_delay
	<dbl>	<dbl>
## 1	2	11
## 2	4	20
## 3	2	33
## 4	-1	-18
## 5	-6	-25
## 6	-4	12
## 7	-5	19
## 8	-3	-14
## 9	-3	-8
## 10	-2	8

```
## # ... with 336,766 more rows
```

2.1.5 Exercise 5

Select the `tailnum` variable and rename it into `tail_num`.

```
flights %>% select(tail_num = tailnum)
```

```
## # A tibble: 336,776 × 1
##   tail_num
##   <chr>
## 1 N14228
## 2 N24211
## 3 N619AA
## 4 N804JB
## 5 N668DN
## 6 N39463
## 7 N516JB
## 8 N829AS
## 9 N593JB
## 10 N3ALAA
## # ... with 336,766 more rows
```

2.1.6 Exercise 6

Select all the variables and rename the `tailnum` variable into `tail_num`.

```
flights %>% rename(tail_num = tailnum)
```

```
## # A tibble: 336,776 × 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     517             515           2     830           819
## 2  2013     1     1     533             529           4     850           830
## 3  2013     1     1     542             540           2     923           850
## 4  2013     1     1     544             545          -1    1004          1022
## 5  2013     1     1     554             600          -6     812           837
## 6  2013     1     1     554             558          -4     740           728
## 7  2013     1     1     555             600          -5     913           854
## 8  2013     1     1     557             600          -3     709           723
## 9  2013     1     1     557             600          -3     838           846
## 10 2013     1     1     558             600          -2     753           745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tail_num <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

2.2 filter() and its friends

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)

## Loading required package: tidyverse

## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'tidyverse'

require(nycflights13)

## Loading required package: nycflights13
```

2.2.1 Exercise 1

Select all flights which delayed more than 1000 minutes at departure.

```
flights %>% filter(dep_delay > 1000)

## # A tibble: 5 × 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     9     641             900      1301     1242         1530
## 2  2013     1    10    1121            1635      1126     1239         1810
## 3  2013     6    15    1432            1935      1137     1607         2120
## 4  2013     7    22     845             1600      1005     1044         1815
## 5  2013     9    20    1139            1845      1014     1457         2210
## # ... with 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 <dtm>
```

2.2.2 Exercise 2

Select all flights which delayed more than 1000 minutes at departure or at arrival.

```
flights %>% filter(dep_delay > 1000 | arr_delay > 1000)
```

```
## # A tibble: 5 × 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     9     641             900      1301    1242         1530
## 2  2013     1    10    1121            1635      1126    1239         1810
## 3  2013     6    15    1432            1935      1137    1607         2120
## 4  2013     7    22     845            1600      1005    1044         1815
## 5  2013     9    20    1139            1845      1014    1457         2210
## # ... with 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 <dtm>
```

```
# alternatively
# flights %>% filter(dep_delay > 1000, arr_delay > 1000)
```

2.2.3 Exercise 3

Select all flights which took off from “EWR” and landed in “IAH” on Christmas Day.

```
flights %>% filter(origin == "EWR" & dest == "IAH" & month == 12 & day == 25)
```

```
## # A tibble: 8 × 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    12    25     524             515         9     805         814
## 2  2013    12    25     753             747         6    1038        1048
## 3  2013    12    25    1018            1015         3    1310        1316
## 4  2013    12    25    1442            1345        57    1730        1646
## 5  2013    12    25    1530            1529         1    1836        1826
## 6  2013    12    25    1628            1630        -2    1944        1925
## 7  2013    12    25    1843            1804        39    2141        2113
## 8  2013    12    25    2003            2006        -3    2304        2314
## # ... with 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 <dtm>
```

```
# alternatively
# flights %>% filter(origin == "EWR", dest == "IAH", month == 12, day == 25)
```

2.2.4 Exercise 4

Select the first five flights in this dataset.

```
flights %>% slice(1:5)
```

```
## # A tibble: 5 × 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     517             515         2      830           819
## 2  2013     1     1     533             529         4      850           830
## 3  2013     1     1     542             540         2      923           850
## 4  2013     1     1     544             545        -1     1004          1022
## 5  2013     1     1     554             600        -6      812           837
## # ... with 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 <dtm>
```

2.2.5 Exercise 5

Select the last ten flights in this dataset.

```
flights %>% slice((n()-9):n())

## # A tibble: 10 × 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     9    30    2240             2250        -10    2347           7
## 2  2013     9    30    2241             2246         -5    2345           1
## 3  2013     9    30    2307             2255         12    2359          2358
## 4  2013     9    30    2349             2359        -10     325           350
## 5  2013     9    30      NA             1842         NA      NA          2019
## 6  2013     9    30      NA             1455         NA      NA          1634
## 7  2013     9    30      NA             2200         NA      NA          2312
## 8  2013     9    30      NA             1210         NA      NA          1330
## 9  2013     9    30      NA             1159         NA      NA          1344
## 10 2013     9    30      NA              840         NA      NA          1020
## # ... with 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 <dtm>
```

2.2.6 Exercise 6

Extract information about distance for all flights which delayed more than 1000 minutes at departure.

```
flights %>%
  filter(dep_delay > 1000) %>%
  select(distance)
```

```
## # A tibble: 5 × 1
##   distance
##   <dbl>
## 1     4983
## 2      719
## 3      483
## 4      589
## 5     2586
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

2.3 arrange()

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(nycflights13)

## Loading required package: nycflights13
```

2.3.1 Exercise 1

Sort the flights in chronological order.

```
flights %>% arrange(year, month, day)

## # A tibble: 336,776 × 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     517             515           2     830             819
## 2  2013     1     1     533             529           4     850             830
## 3  2013     1     1     542             540           2     923             850
## 4  2013     1     1     544             545          -1    1004            1022
## 5  2013     1     1     554             600          -6     812             837
## 6  2013     1     1     554             558          -4     740             728
## 7  2013     1     1     555             600          -5     913             854
## 8  2013     1     1     557             600          -3     709             723
## 9  2013     1     1     557             600          -3     838             846
## 10 2013     1     1     558             600          -2     753             745
## # ... with 336,766 more rows, and 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 <dtm>
```


2.3.2 Exercise 2

Sort the flights by decreasing arrival delay.

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

## # A tibble: 336,776 × 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     9     641             900      1301    1242         1530
## 2  2013     6    15    1432            1935      1137    1607         2120
## 3  2013     1    10    1121            1635      1126    1239         1810
## 4  2013     9    20    1139            1845      1014    1457         2210
## 5  2013     7    22     845            1600      1005    1044         1815
## 6  2013     4    10    1100            1900       960    1342         2211
## 7  2013     3    17    2321             810       911     135         1020
## 8  2013     7    22    2257             759       898     121         1026
## 9  2013    12     5     756            1700       896    1058         2020
## 10 2013     5     3    1133            2055       878    1250         2215
## # ... with 336,766 more rows, and 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 <dtm>
```

2.3.3 Exercise 3

Sort the flights by origin (in alphabetical order) and decreasing arrival delay.

```
flights %>% arrange(origin, desc(arr_delay))

## # A tibble: 336,776 × 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    10    1121            1635      1126    1239         1810
## 2  2013    12     5     756            1700       896    1058         2020
## 3  2013     5     3    1133            2055       878    1250         2215
## 4  2013    12    19     734            1725       849    1046         2039
## 5  2013    12    17     705            1700       845    1026         2020
## 6  2013    11     3     603            1645       798     829         1913
## 7  2013     2    24    1921             615       786    2135         842
## 8  2013    10    14    2042             900       702    2255         1127
## 9  2013     7    21    1555             615       580    1955         910
## 10 2013     7     7    2123            1030       653     17         1345
## # ... with 336,766 more rows, and 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 <dtm>
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

2.4 mutate() and its friends

Note: all the exercises of this section are based on the `flights` dataset. Times are in minutes and distances are in miles.

```
require(tidyverse)
require(nycflights13)

## Loading required package: nycflights13
```

2.4.1 Exercise 1

Add the following new variables to the `flights` dataset:

- the gained time in minutes, defined as the difference between delay at departure and delay at arrival;
- the speed in miles per hour (`distance / air_time * 60`).

Show only the following variables: delay at departure, delay at arrival, distance, air time and the two new variables (gained time and speed).

```
flights %>% mutate(gained_time = arr_delay - dep_delay, speed = distance/air_time*60)

## # A tibble: 336,776 × 21
##   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     517             515           2     830           819
## 2  2013     1     1     533             529           4     850           830
## 3  2013     1     1     542             540           2     923           850
## 4  2013     1     1     544             545          -1    1004          1022
## 5  2013     1     1     554             600          -6     812           837
## 6  2013     1     1     554             558          -4     740           728
## 7  2013     1     1     555             600          -5     913           854
## 8  2013     1     1     557             600          -3     709           723
## 9  2013     1     1     557             600          -3     838           846
## 10 2013     1     1     558             600          -2     753           745
## # ... with 336,766 more rows, and 13 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 <dtm>,
## #   gained_time <dbl>, speed <dbl>
```

2.4.2 Exercise 2

Redo the previous calculations keeping only the new variables.

```
flights %>%
  transmute(gained_time = arr_delay - dep_delay, speed = distance/air_time*60)

## # A tibble: 336,776 × 2
##   gained_time    speed
##   <dbl>      <dbl>
## 1         9 370.0441
## 2        16 374.2731
## 3        31 408.3750
## 4       -17 516.7213
## 5       -19 394.1379
## 6        16 287.6000
## 7        24 404.4304
## 8       -11 259.2453
## 9         -5 404.5714
## 10        10 318.6957
## # ... with 336,766 more rows
```

2.4.3 Exercise 3

After sorting flights in chronological order, for each flight calculate the difference between its delay at arrival and the delay at arrival of the immediately previous flight. Have R to show only the delay variables (delay at departure, delay at arrival and the new variable).

```
flights %>%
  arrange(year, month, day) %>%
  mutate(lead_arr_delay = lead(arr_delay), delta_delay = lead_arr_delay - arr_delay) %>%
  select(dep_delay, arr_delay, delta_delay)

## # A tibble: 336,776 × 3
##   dep_delay arr_delay delta_delay
##   <dbl>      <dbl>      <dbl>
## 1         2         11           9
## 2         4         20          13
## 3         2         33         -51
## 4        -1        -18          -7
## 5        -6        -25          37
## 6        -4         12           7
## 7        -5         19         -33
## 8        -3        -14           6
## 9        -3         -8          16
## 10       -2          8         -10
## # ... with 336,766 more rows
```

2.4.4 Exercise 4

For each flight calculate the ‘min ranking’ in terms of delay at arrival.

```
flights %>%
  mutate(min_rank_arr_delay = min_rank(arr_delay))

## # A tibble: 336,776 × 20
##   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     517           515           2     830           819
## 2  2013     1     1     533           529           4     850           830
## 3  2013     1     1     542           540           2     923           850
## 4  2013     1     1     544           545          -1    1004          1022
## 5  2013     1     1     554           600          -6     812           837
## 6  2013     1     1     554           558          -4     740           728
## 7  2013     1     1     555           600          -5     913           854
## 8  2013     1     1     557           600          -3     709           723
## 9  2013     1     1     557           600          -3     838           846
## 10 2013     1     1     558           600          -2     753           745
## # ... with 336,766 more rows, and 12 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 <dtm>,
## #   min_rank_arr_delay <int>
```

2.4.5 Exercise 5

For each flight calculate the ‘first ranking’ in terms of delay at arrival.

```
flights %>%
  mutate(first_rank_arr_delay = row_number(arr_delay))

## # A tibble: 336,776 × 20
##   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     517           515           2     830           819
## 2  2013     1     1     533           529           4     850           830
## 3  2013     1     1     542           540           2     923           850
## 4  2013     1     1     544           545          -1    1004          1022
## 5  2013     1     1     554           600          -6     812           837
## 6  2013     1     1     554           558          -4     740           728
## 7  2013     1     1     555           600          -5     913           854
## 8  2013     1     1     557           600          -3     709           723
## 9  2013     1     1     557           600          -3     838           846
## 10 2013     1     1     558           600          -2     753           745
## # ... with 336,766 more rows, and 12 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 <dtm>,
## #   first_rank_arr_delay <int>
```

2.4.6 Exercise 6

Create a variable which indicates if a flight took off on time, i.e. departure delay is more than -4 and less than 4 minutes late.

```
flights %>%
  filter(arr_delay > -4 & arr_delay <4) %>%
  mutate(dep_on_time = 1)

## # A tibble: 37,061 × 20
##   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     558             600        -2     849             851
## 2  2013     1     1     558             600        -2     853             856
## 3  2013     1     1     622             630        -8    1017            1014
## 4  2013     1     1     623             627        -4     933             932
## 5  2013     1     1     627             630        -3    1018            1018
## 6  2013     1     1     628             630        -2    1137            1140
## 7  2013     1     1     658             700        -2    1027            1025
## 8  2013     1     1     659             700        -1    1008            1007
## 9  2013     1     1     728             732        -4    1041            1038
## 10 2013     1     1     732             735        -3     857             858
## # ... with 37,051 more rows, and 12 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 <dtm>,
## #   dep_on_time <dbl>

## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

2.5 summarise()

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(nycflights13)

## Loading required package: nycflights13
```

2.5.1 Exercise 1

Calculate minimum, mean and maximum delay at arrival.

```
flights %>% summarise(min_arr_delay = min(arr_delay, na.rm = T),
                      mean_arr_delay = mean(arr_delay, na.rm = T),
                      max_arr_delay = max(arr_delay, na.rm = T))

## # A tibble: 1 × 3
##   min_arr_delay mean_arr_delay max_arr_delay
##         <dbl>         <dbl>         <dbl>
## 1          -86         6.895377         1272
```

2.5.2 Exercise 2

Calculate minimum, mean and maximum delay at arrival for flights in January.

```
flights %>%
  filter(month == 1) %>%
  summarise(min_arr_delay = min(arr_delay, na.rm = T),
            mean_arr_delay = mean(arr_delay, na.rm = T),
            max_arr_delay = max(arr_delay, na.rm = T))

## # A tibble: 1 × 3
##   min_arr_delay mean_arr_delay max_arr_delay
##         <dbl>         <dbl>         <dbl>
## 1          -70         6.129972         1272
```

2.5.3 Exercise 3

Calculate the number of flights are contained in the dataset

```
flights %>%
  summarise(n_flights = n())

## # A tibble: 1 × 1
##   n_flights
##       <int>
## 1     336776

## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

Chapter 3

Grouping data

3.1 group_by()

Note: all the exercises of this section are based on the `flights` dataset.

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

3.1.1 Exercise 1

Calculate number of flights, minimum, mean and maximum delay at arrival for flights by month.

```
flights %>%
  group_by(month) %>%
  summarise(n_flights = n(),
            min_arr_delay = min(arr_delay, na.rm = T),
            mean_arr_delay = mean(arr_delay, na.rm = T),
            max_arr_delay = max(arr_delay, na.rm = T))
```



```
## # A tibble: 12 × 5
##   month n_flights min_arr_delay mean_arr_delay max_arr_delay
##   <int>   <int>         <dbl>         <dbl>         <dbl>
## 1     1     27004          -70         6.1299720       1272
## 2     2     24951          -70         5.6130194        834
## 3     3     28834          -68         5.8075765        915
## 4     4     28330          -68        11.1760630        931
## 5     5     28796          -86         3.5215088        875
## 6     6     28243          -64        16.4813296       1127
## 7     7     29425          -66        16.7113067        989
## 8     8     29327          -68         6.0406524        490
## 9     9     27574          -68        -4.0183636       1007
## 10    10     28889          -61        -0.1670627        688
```

```
## 11    11    27268          -67      0.4613474      796
## 12    12    28135          -68     14.8703553      878
```

3.1.2 Exercise 2

Calculate number of flights, mean delay at departure and arrival for flights by origin.

```
flights %>%
  group_by(origin) %>%
  summarise(n_flights = n(),
            min_arr_delay = min(arr_delay, na.rm = T),
            mean_arr_delay = mean(arr_delay, na.rm = T),
            max_arr_delay = max(arr_delay, na.rm = T))

## # A tibble: 3 × 5
##   origin n_flights min_arr_delay mean_arr_delay max_arr_delay
##   <chr>   <int>         <dbl>         <dbl>         <dbl>
## 1   EWR    120835         -86          9.107055         1109
## 2   JFK    111279         -79          5.551481         1272
## 3   LGA    104662         -68          5.783488          915
```

3.1.3 Exercise 3

Calculate the number of flights that go to each possible destination.

```
flights %>%
  group_by(dest) %>%
  summarise(n_flights = n())

## # A tibble: 105 × 2
##   dest n_flights
##   <chr>   <int>
## 1   ABQ       254
## 2   ACK       265
## 3   ALB       439
## 4   ANC         8
## 5   ATL    17215
## 6   AUS     2439
## 7   AVL       275
## 8   BDL       443
## 9   BGR       375
## 10  BHM       297
## # ... with 95 more rows
```


3.1.4 Exercise 4

Calculate the number of flights for each day. Save the result in a data frame called `per_day`.

```
per_day <- flights %>%
  group_by(year, day, month) %>%
  summarise(n_flights = n())
```

3.1.5 Exercise 5

By exploiting `per_day`, calculate the number of flights for each month. Save the result in a data frame called `per_month`.

```
per_month <- flights %>%
  group_by(month) %>%
  summarise(n_flights = n())
```

3.1.6 Exercise 6

Calculate the mean daily number of flights per month.

```
per_month %>%
  group_by(month) %>%
  summarise(mean_n_flights = mean(n_flights))
```

```
## # A tibble: 12 × 2
##   month mean_n_flights
##   <int>         <dbl>
## 1     1         27004
## 2     2         24951
## 3     3         28834
## 4     4         28330
## 5     5         28796
## 6     6         28243
## 7     7         29425
## 8     8         29327
## 9     9         27574
## 10    10         28889
## 11    11         27268
## 12    12         28135
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```


Chapter 4

Do

4.1 do

Note: all the exercises of this section are based on the `flights` dataset.

```
library(dplyr)
library(nycflights13)
```

4.1.1 Exercise 1

Calculate quartiles (25-, 50- and 75-percentiles) of delay at arrival per origin. Put all three quartiles in a unique column.

```
flights %>% group_by(origin) %>%
  do(data.frame(p = (1:3)/4, quantile = quantile(.$arr_delay, probs = (1:3)/4, na.rm = TRUE)))

## Source: local data frame [9 x 3]
## Groups: origin [3]
##
##   origin      p quantile
##   <chr> <dbl>   <dbl>
## 1   EWR  0.25    -16
## 2   EWR  0.50     -4
## 3   EWR  0.75     16
## 4   JFK  0.25    -18
## 5   JFK  0.50     -6
## 6   JFK  0.75     13
## 7   LGA  0.25    -17
## 8   LGA  0.50     -5
## 9   LGA  0.75     12
```

4.1.2 Exercise 2

Redo the previous exercise putting the three quartiles in three different columns (hint: use `summarise()`).

```
flights %>% group_by(origin) %>%
  do(data.frame(p1 = quantile(.$arr_delay, probs = 1/4, na.rm = TRUE),
    p2 = quantile(.$arr_delay, probs = 2/4, na.rm = TRUE),
    p3 = quantile(.$arr_delay, probs = 3/4, na.rm = TRUE)))

## Source: local data frame [3 x 4]
## Groups: origin [3]
##
##   origin    p1    p2    p3
##   <chr> <dbl> <dbl> <dbl>
## 1   EWR   -16    -4    16
## 2   JFK   -18    -6    13
## 3   LGA  -17    -5    12
```

4.1.3 Exercise 3

Calculate mean and standard deviation of delay at arrival per origin. Put both statistics in a unique column.

```
fun <- function(x, ...) c(mean = mean(x, ...), sd = sd(x, ...))

flights %>% group_by(origin) %>%
  do(data.frame(stats = c("mean", "sd") , value = fun(.$arr_delay, na.rm = TRUE)))

## Source: local data frame [6 x 3]
## Groups: origin [3]
##
##   origin stats    value
##   <chr> <fctr>    <dbl>
## 1   EWR  mean  9.107055
## 2   EWR  sd  45.529183
## 3   JFK  mean  5.551481
## 4   JFK  sd  44.277448
## 5   LGA  mean  5.783488
## 6   LGA  sd  43.862273
```

4.1.4 Exercise 4

Redo the previous exercise putting mean and standard deviation in two different columns (hint: use `summarise()`).

```
flights %>% group_by(origin) %>%  
  do(data.frame(arr_delay_mean = mean(.$arr_delay, na.rm = TRUE),  
                arr_delay_sd = sd(.$arr_delay, na.rm = TRUE)))  
  
## Source: local data frame [3 x 3]  
## Groups: origin [3]  
##  
##   origin arr_delay_mean arr_delay_sd  
##   <chr>      <dbl>      <dbl>  
## 1    EWR      9.107055    45.52918  
## 2    JFK      5.551481    44.27745  
## 3    LGA      5.783488    43.86227  
  
## Warning in .doLoadActions(where, attach): trying to execute load actions without  
## 'methods' package
```


Chapter 5

Combining data

5.1 Joins: `inner_join()`, `left_join()`, `right_join()`, etc.

Note: all the exercises of this section are based on `flights`, `airlines`, `airports` or `planes` datasets.

```
library(dplyr)
library(nycflights13)
```

5.1.1 Exercise 1

Keep only the following variables of the `flights` dataset: `month`, `day`, `hour`, `origin`, `destination` and `carrier`. Save this dataset in a data frame and call it `flights_red`. Through a proper join command, add the carrier name to `flights_red` (this piece of information is available in `airlines`).

```
flights_red <- flights %>%
  select(month, day, hour, origin, dest, carrier)

right_join(flights_red, airlines)

## Joining, by = "carrier"

## # A tibble: 336,776 × 7
##   month   day hour origin dest carrier      name
##   <int> <int> <dbl> <chr> <chr>   <chr>    <chr>
## 1     1     1     8   JFK   MSP     9E Endeavor Air Inc.
## 2     1     1    15   JFK   IAD     9E Endeavor Air Inc.
## 3     1     1    14   JFK   BUF     9E Endeavor Air Inc.
## 4     1     1    15   JFK   SYR     9E Endeavor Air Inc.
```

```
## 5      1      1     15    JFK    ROC      9E Endeavor Air Inc.
## 6      1      1     15    JFK    BWI      9E Endeavor Air Inc.
## 7      1      1     15    JFK    ORD      9E Endeavor Air Inc.
## 8      1      1     15    JFK    IND      9E Endeavor Air Inc.
## 9      1      1     16    JFK    BNA      9E Endeavor Air Inc.
## 10     1      1     16    JFK    BOS      9E Endeavor Air Inc.
## # ... with 336,766 more rows
```

5.1.2 Exercise 2

Through a proper join command, add name, latitude, longitude and altitude of the origin airport to `flights_red` (these pieces of information are available in `airports`). Do the same also for the destination airport. (If you are able to, try to keep variables about both origin and destination airports in the same final dataset).

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

```
## # A tibble: 336,776 × 13
##   month   day   hour origin dest carrier      name      lat
##   <int> <int> <dbl> <chr> <chr> <chr>      <chr>    <dbl>
## 1      1     1     5    EWR   IAH    UA Newark Liberty Intl 40.69250
## 2      1     1     5    LGA   IAH    UA      La Guardia 40.77725
## 3      1     1     5    JFK   MIA    AA John F Kennedy Intl 40.63975
## 4      1     1     5    JFK   BQN    B6 John F Kennedy Intl 40.63975
## 5      1     1     6    LGA   ATL    DL      La Guardia 40.77725
## 6      1     1     5    EWR   ORD    UA Newark Liberty Intl 40.69250
## 7      1     1     6    EWR   FLL    B6 Newark Liberty Intl 40.69250
## 8      1     1     6    LGA   IAD    EV      La Guardia 40.77725
## 9      1     1     6    JFK   MCO    B6 John F Kennedy Intl 40.63975
## 10     1     1     6    LGA   ORD    AA      La Guardia 40.77725
## # ... with 336,766 more rows, and 5 more variables: lon <dbl>, alt <int>,
## #   tz <dbl>, dst <chr>, tzone <chr>
```

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

```
## # A tibble: 336,776 × 13
##   month   day   hour origin dest carrier      name
##   <int> <int> <dbl> <chr> <chr> <chr>      <chr>
## 1      1     1     5    EWR   IAH    UA George Bush Intercontinental
## 2      1     1     5    LGA   IAH    UA George Bush Intercontinental
## 3      1     1     5    JFK   MIA    AA Miami Intl
## 4      1     1     5    JFK   BQN    B6 <NA>
## 5      1     1     6    LGA   ATL    DL Hartsfield Jackson Atlanta Intl
## 6      1     1     5    EWR   ORD    UA Chicago Ohare Intl
## 7      1     1     6    EWR   FLL    B6 Fort Lauderdale Hollywood Intl
```



```
## 8      1      1      6    LGA    IAD      EV      Washington Dulles Intl
## 9      1      1      6    JFK    MCO      B6              Orlando Intl
## 10     1      1      6    LGA    ORD      AA              Chicago Ohare Intl
## # ... with 336,766 more rows, and 6 more variables: lat <dbl>, lon <dbl>,
## #   alt <int>, tz <dbl>, dst <chr>, tzone <chr>
```

5.1.3 Exercise 3

Through the `inner_join()` function, redo the same for the destination airport but keep only the flights whose information is available in both datasets (`flights` and `airports`).

```
flights_red %>% inner_join(airports, c("dest" = "faa"))

## # A tibble: 329,174 × 13
##   month   day hour origin dest carrier      name
##   <int> <int> <dbl> <chr> <chr>   <chr>   <chr>
## 1     1     1     5   EWR   IAH     UA   George Bush Intercontinental
## 2     1     1     5   LGA   IAH     UA   George Bush Intercontinental
## 3     1     1     5   JFK   MIA     AA           Miami Intl
## 4     1     1     6   LGA   ATL     DL Hartsfield Jackson Atlanta Intl
## 5     1     1     5   EWR   ORD     UA           Chicago Ohare Intl
## 6     1     1     6   EWR   FLL     B6   Fort Lauderdale Hollywood Intl
## 7     1     1     6   LGA   IAD     EV      Washington Dulles Intl
## 8     1     1     6   JFK   MCO     B6              Orlando Intl
## 9     1     1     6   LGA   ORD     AA           Chicago Ohare Intl
## 10    1     1     6   JFK   PBI     B6           Palm Beach Intl
## # ... with 329,164 more rows, and 6 more variables: lat <dbl>, lon <dbl>,
## #   alt <int>, tz <dbl>, dst <chr>, tzone <chr>
```

5.1.4 Exercise 4

Redo the exercise 3 by using `full_join()` instead of `inner_join()`. What is the difference in the result?

```
flights_red %>% full_join(airports, c("dest" = "faa"))

## # A tibble: 338,133 × 13
##   month   day hour origin dest carrier      name
##   <int> <int> <dbl> <chr> <chr>   <chr>   <chr>
## 1     1     1     5   EWR   IAH     UA   George Bush Intercontinental
## 2     1     1     5   LGA   IAH     UA   George Bush Intercontinental
## 3     1     1     5   JFK   MIA     AA           Miami Intl
## 4     1     1     5   JFK   BQN     B6              <NA>
## 5     1     1     6   LGA   ATL     DL Hartsfield Jackson Atlanta Intl
```

```
## 6      1      1      5    EWR    ORD      UA      Chicago Ohare Intl
## 7      1      1      6    EWR    FLL      B6    Fort Lauderdale Hollywood Intl
## 8      1      1      6    LGA    IAD      EV      Washington Dulles Intl
## 9      1      1      6    JFK    MCO      B6      Orlando Intl
## 10     1      1      6    LGA    ORD      AA      Chicago Ohare Intl
## # ... with 338,123 more rows, and 6 more variables: lat <dbl>, lon <dbl>,
## #   alt <int>, tz <dbl>, dst <chr>, tzone <chr>
```

there are a few more rows due to the fact that full_join keeps all rows even those with n

5.1.5 Exercise 5

Through the `anti_join()` function, extract all the flights from `flights` whose information about destination airport is not available in `airports`.

```
flights_red %>% anti_join(airports, c("dest" = "faa"))
```

```
## # A tibble: 7,602 × 6
##   month  day  hour origin dest carrier
##   <int> <int> <dbl> <chr> <chr> <chr>
## 1     1     1     23   JFK   PSE    B6
## 2     1     2     23   JFK   PSE    B6
## 3     1     3     23   JFK   PSE    B6
## 4     1     4     23   JFK   PSE    B6
## 5     1     5     23   JFK   PSE    B6
## 6     1     6     23   JFK   PSE    B6
## 7     1     7     23   JFK   PSE    B6
## 8     1     8     23   JFK   PSE    B6
## 9     1     9     23   JFK   PSE    B6
## 10    1    10     23   JFK   PSE    B6
## # ... with 7,592 more rows
```

5.1.6 Exercise 6

Sort the `planes` dataset by increasing year. Then create two datasets: the first will deal with planes older than 2000; the second will deal with planes of 2000 or newer. Finally create a unique dataset where the first rows will deal with the newest planes, whereas the last rows will deal with the oldest planes.

```
planes_old <- planes %>%
  arrange(year) %>%
  slice(year <= 2000)

planes_young <- planes %>%
```

```

  arrange(year) %>%
  slice(year > 2000)

planes_old %>% bind_rows(planes_young)

## # A tibble: 3,252 × 9
##   tailnum year      type manufacturer model engines seats
##   <chr> <int>      <chr>      <chr>    <chr>   <int> <int>
## 1  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 2  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 3  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 4  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 5  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 6  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 7  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 8  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 9  N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## 10 N381AA  1956 Fixed wing multi engine    DOUGLAS DC-7BF      4    102
## # ... with 3,242 more rows, and 2 more variables: speed <int>, engine <chr>

## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package

```


Chapter 6

Tidy data with tidyr

6.1 tidyr

```
library(tidyverse)
```

6.1.1 Exercise 1

Consider the following dataset:

```
heartrate_wide <- data.frame(
  name = c("Aldo", "Giovanni", "Giacomo"),
  surname = c("Baglio", "Storti", "Poretti"),
  morning = c(67, 80, 64),
  afternoon = c(56, 90, 50)
)
heartrate_wide
```

	name	surname	morning	afternoon
## 1	Aldo	Baglio	67	56
## 2	Giovanni	Storti	80	90
## 3	Giacomo	Poretti	64	50

It represents the heart rate measured on three patients in the morning and in the afternoon. The dataset is in the wide format: change it to the long format through a proper `tidyr` function. Save the result in a data frame and call it `heartrate_long`.

```
heartrate_long <- gather(heartrate_wide, key = "when", value = "value" , 3:4)
```

```
## Error in is.data.frame(x): object 'heartrate_wide' not found
```

6.1.2 Exercise 2

Starting from `heartrate_long`, come back to a dataset in a wide format through a proper `tidyr` function. The result should be obviously equal to `heartrate_wide`.

```
spread(heartrate_long, key = when, value = "value")

## Error in match(x, table, nomatch = 0L): object 'heartrate_long' not found
```

6.1.3 Exercise 3

Consider the dataset `heartrate_wide` and unite name and surname of the patients in a unique column through a proper `tidyr` function. Save the result in a new data frame called `heartrate_united`.

```
heartrate_united <- heartrate_wide %>%
  unite(name_surname, name, surname)

## Error in eval(expr, envir, enclos): object 'heartrate_wide' not found
```

6.1.4 Exercise 4

Starting from `heartrate_united`, come back to a dataset where name and surname are in two different columns through a proper `tidyr` function. The result should be obviously equal to `heartrate_wide`.

```
heartrate_united %>%
  separate(name_surname, c("name", "surname"))

## Error in eval(expr, envir, enclos): object 'heartrate_united' not found

## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

Chapter 7

Handling Missing values

7.1

```
library(tidyverse)
```

7.1.1 Exercise 1

Consider the following dataset:

```
heartrate <- data.frame(  
  name = c("Aldo", "Giovanni", "Giacomo", "Aldo", "Giovanni", "Giacomo", "Giovanni", "Giacomo"),  
  surname = c("Baglio", "Storti", "Poretti", "Baglio", "Storti", "Poretti", "Storti", "Poretti"),  
  when = c("morning", "morning", "morning", "afternoon", "afternoon", "afternoon", "evening", "evening"),  
  heartrate = c(67, 80, 64, 56, 90, 50, 60, 85)  
)
```

It represents the heart rate measured on three patients in the morning, in the afternoon and in the evening. Make explicit any implicit missing value. How many missing values do you see?

```
heartrate %>% complete(surname, when, fill=list(name="Aldo"))
```

```
## # A tibble: 9 × 4  
##   surname    when    name heartrate  
##   <fctr>    <fctr>  <fctr>    <dbl>  
## 1 Baglio  afternoon  Aldo      56  
## 2 Baglio  evening   Aldo      NA  
## 3 Baglio  morning   Aldo      67  
## 4 Poretti afternoon Giacomo    50  
## 5 Poretti evening   Giacomo    85  
## 6 Poretti morning   Giacomo    64  
## 7 Storti  afternoon Giovanni    90
```

```
## 8 Storti evening Giovanni 60
## 9 Storti morning Giovanni 80
```

```
# one missing value
# alternatively:
# heartrate %>%
# complete(surname, when) %>%
# fill(name)
```

7.1.2 Exercise 2

Import data in the file `marks.Rdta`. Missing values have been recorded as “?”. What’s the percentage of missing values in the data? Replace them with NA and drop them.

```
marks_NA <- na_if(marks, ".")
marks_NA %>%
  filter(is.na(marks)) %>%
  summarise(n())/30
```

```
##           n()
## 1 0.06666667
```

```
# 0.067% missing, only one variable, we may drop missing obs
marks_NA %>% drop_na()
```

```
## # A tibble: 28 × 1
##   marks
##   <chr>
## 1    25
## 2    21
## 3    26
## 4    23
## 5    23
## 6    24
## 7    22
## 8    24
## 9    23
## 10   26
## # ... with 18 more rows
```

7.1.3 Exercise 3

Import the data `heartrate_NA.Rdta`. Consider all the missing values you find and replace them using the function `fill()` when possible.


```
heartrate_NA %>%  
  na_if( " ") %>%  
  fill(name, surname)
```

```
## # A tibble: 9 × 4  
##       name surname      when heartrate  
##   <chr>   <chr>    <chr>    <dbl>  
## 1    Aldo  Baglio  morning      67  
## 2    Aldo  Baglio afternoon    56  
## 3    Aldo  Baglio  evening      67  
## 4 Giovanni Storti  morning      80  
## 5 Giovanni Storti afternoon    90  
## 6 Giovanni Storti  evening      60  
## 7 Giacomo Poretti morning      64  
## 8 Giacomo Poretti afternoon    50  
## 9 Giacomo Poretti evening      85
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without  
## 'methods' package
```


Chapter 8

Dates with lubridate

8.1 lubridate

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(lubridate)

## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##      date

require(nycflights13)

## Loading required package: nycflights13
```

8.1.1 Exercise 1

Using the `flights` data, build the variable `dep_date` based on the variables `year`, `month` and `day`. First use the function `unite()` and then the parsing function `ydm()`. Select only the new variable and save the new data frame called `flights_date`.

```
flights_date <- flights %>%
  unite(date, year, month, day) %>%
  mutate(date = ymd(date)) %>%
  select(date)
```

8.1.2 Exercise 2

Using the dataset, shift all flights by two months. Save it in a separate data frame called `flights_date_2`.

```
flights_date_2 <- flights_date %>% mutate(date2 = date + months(2))
```

8.1.3 Exercise 3

Take the new date (2 months ahead) and subtract the original variable `date`. `flights_date_2`.

```
flights_date_2 %>% mutate(date2-date)
```

```
## # A tibble: 336,776 × 3
##       date      date2 `date2 - date`
##   <date>    <date>      <time>
## 1 2013-01-01 2013-03-01    59 days
## 2 2013-01-01 2013-03-01    59 days
## 3 2013-01-01 2013-03-01    59 days
## 4 2013-01-01 2013-03-01    59 days
## 5 2013-01-01 2013-03-01    59 days
## 6 2013-01-01 2013-03-01    59 days
## 7 2013-01-01 2013-03-01    59 days
## 8 2013-01-01 2013-03-01    59 days
## 9 2013-01-01 2013-03-01    59 days
## 10 2013-01-01 2013-03-01    59 days
## # ... with 336,766 more rows
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```

Chapter 9

Manipulating strings with stringr

9.1

```
library(tidyverse)
library(stringr)
```

9.1.1 Exercise 1

Import the data `aire_milano_strings.txt` which is a tab delimited file. Find how China has been codified (notice that the file is in Italian) and manipulate that string as you find more comfortable for you. Save the results in a new tibble.

```
## Parsed with column specification:
## cols(
##   Residenza = col_character(),
##   MotivoIscrizioneEstero = col_character(),
##   Num = col_integer()
## )

aire %>% filter(str_detect(Residenza, c("Cina"))) # not recorded as Cina

## # A tibble: 0 × 3
## # ... with 3 variables: Residenza <chr>, MotivoIscrizioneEstero <chr>,
## #   Num <int>

aire %>% filter(str_detect(Residenza, c("cina"))) # not recorded as cina

## # A tibble: 0 × 3
## # ... with 3 variables: Residenza <chr>, MotivoIscrizioneEstero <chr>,
## #   Num <int>
```

```

aire %>% filter(str_detect(Residenza, c("Cin")))

## # A tibble: 5 × 3
##       Residenza      MotivoIscrizioneEstero  Num
##       <chr>          <chr> <int>
## 1 Cinese, Rep. Popolare      all'emigrazione    535
## 2 Cinese, Rep. Popolare per acquisto cittadinanza    14
## 3 Cinese, Rep. Popolare      per nascita    119
## 4 Cinese, Rep. Popolare per residenza all'estero    26
## 5 Cinese, Rep. Popolare      trasferimento da AIRE    10

aire_clean <- aire %>% mutate(Residenza, Residenza = str_replace(Residenza, c("Cinese, Rep.

```

9.1.2 Exercise 2

Using the data modified in exercise 1, find all the countries whose names contain non-alphanumeric characters. Identify what kind of characters they contain.

```

str_extract(aire_clean$Residenza, "[[:punct:]]")

```

9.1.3 Exercise 3

Consider now the column with information on the reason for migrating. Count how many different reasons there are and notice that citizenship was recorded in two slightly different ways: “acquisto cittadinanza” and “per acquisto cittadinanza”. Replace one of them so that they are the same.

```

# find all distinct reasons
aire_clean %>% distinct(MotivoIscrizioneEstero)

## # A tibble: 8 × 1
##       MotivoIscrizioneEstero
##       <chr>
## 1      all'emigrazione
## 2      per nascita
## 3 per residenza all'estero
## 4      acquisto cittadinanza
## 5      per matrimonio
## 6      trasferimento da AIRE
## 7 per acquisto cittadinanza
## 8      per sentenza

aire_clean <- aire_clean %>% mutate(MotivoIscrizioneEstero = str_replace(MotivoIscrizioneEstero, "per acquisto cittadinanza", "acquisto cittadinanza"))

# now you only have 7 different levels
aire_clean %>% distinct(MotivoIscrizioneEstero)

```

```
## # A tibble: 7 × 1
##   MotivoIscrizioneEsterio
##   <chr>
## 1 all'emigrazione
## 2 per nascita
## 3 per residenza all'estero
## 4 per acquisto cittadinanza
## 5 per matrimonio
## 6 trasferimento da AIRE
## 7 per sentenza
```

```
## Warning in .doLoadActions(where, attach): trying to execute load actions without
## 'methods' package
```


Chapter 10

Case study

10.1 Recap exercise

In this section you will work on a real data set. Using all the tools provided throughout the course, you will manipulate data for better analysing it. In the data folder you find the following three files:

1. rating_final.csv
2. chefmezcuisine.csv
3. userprofile.csv

These are the files you will work on.

10.1.1 Exercise 1

1. First of all you need to import the three files into R using the correct `readr` function. In order to find the correct function and to set the right options, you'd better explore the files by opening them in csv (check which is the separator, if there are column names, etc).

```
require(tidyverse)

# chefmozcuisine
chefmozcuisine <- read_delim("/home/emanuela/dev/qtraining/050-dplyr-datamanage/exercises/data/chefmozcuisine.csv",
                             delim=";", as_is_strings=TRUE,
                             show_col_types=FALSE)

## Parsed with column specification:
## cols(
##   placeID = col_integer(),
##   Rcuisine = col_character()
## )
```

```

# userprofile
userprofile <- read_delim("/home/emanuela/dev/qtraining/050-dplyr-datamanager/exercises/data/

## Parsed with column specification:
## cols(
##   userID = col_character(),
##   latitude = col_double(),
##   longitude = col_double(),
##   smoker = col_character(),
##   drink_level = col_character(),
##   dress_preference = col_character(),
##   ambience = col_character(),
##   transport = col_character(),
##   marital_status = col_character(),
##   hijos = col_character(),
##   birth_year = col_integer(),
##   interest = col_character(),
##   personality = col_character(),
##   religion = col_character(),
##   activity = col_character(),
##   color = col_character(),
##   weight = col_integer(),
##   budget = col_character(),
##   height = col_double()
## )

# rating_final
rating_final <- read_delim("/home/emanuela/dev/qtraining/050-dplyr-datamanager/exercises/data/

## Parsed with column specification:
## cols(
##   userID = col_character(),
##   placeID = col_integer(),
##   rating = col_integer(),
##   food_rating = col_integer(),
##   service_rating = col_integer()
## )

```

2. In order to understand what you are working on, check how many columns and rows each dataset is composed of, and check what type of variables you are working with. If the variables type has not correctly been parsed, parse it manually. If you find many variables, focus on those that you think may be interesting for understanding different people's tastes (for example age of the users, job, etc).

```

# chefmozcuisine
# how many rows? how many variables?
chefmozcuisine

```

```
## # A tibble: 916 × 2
##   placeID      Rcuisine
##   <int>      <chr>
## 1  135110      Spanish
## 2  135109      Italian
## 3  135107 Latin_American
## 4  135106      Mexican
## 5  135105      Fast_Food
## 6  135104      Mexican
## 7  135103      Burgers
## 8  135103 Dessert-Ice_Cream
## 9  135103      Fast_Food
## 10 135103      Hot_Dogs
## # ... with 906 more rows
```

```
# how many different types of cuisine?
chefmozcuisine %>% distinct(Rcuisine)
```

```
## # A tibble: 59 × 1
##       Rcuisine
##       <chr>
## 1      Spanish
## 2      Italian
## 3 Latin_American
## 4      Mexican
## 5      Fast_Food
## 6      Burgers
## 7 Dessert-Ice_Cream
## 8      Hot_Dogs
## 9      Steaks
## 10     Asian
## # ... with 49 more rows
```

```
# userprofile
# how many rows? how many variables?
userprofile
```

```
## # A tibble: 138 × 19
##   userID latitude longitude smoker  drink_level dress_preference  ambience
##   <chr>    <dbl>    <dbl> <chr>      <chr>      <chr>      <chr>
## 1  U1001  22.14000 -100.9788 false    abstemious    informal    family
## 2  U1002  22.15009 -100.9833 false    abstemious    informal    family
## 3  U1003  22.11985 -100.9465 false  social drinker    formal    family
## 4  U1004  18.86700  -99.1830 false    abstemious    informal    family
## 5  U1005  22.18348 -100.9599 false    abstemious  no preference    family
## 6  U1006  22.15000 -100.9830  true  social drinker  no preference    friends
```

```
## 7  U1007 22.11846 -100.9383 false casual drinker      informal solitary
## 8  U1008 22.12299 -100.9238 false social drinker      formal solitary
## 9  U1009 22.15943 -100.9904 false abstemious          formal family
## 10 U1010 22.19089 -100.9987 false social drinker      no preference friends
## # ... with 128 more rows, and 12 more variables: transport <chr>,
## # marital_status <chr>, hijos <chr>, birth_year <int>, interest <chr>,
## # personality <chr>, religion <chr>, activity <chr>, color <chr>,
## # weight <int>, budget <chr>, height <dbl>
```

```
# let us explore some interesting variables that may be interesting in future analysis
userprofile %>% distinct(activity)
```

```
## # A tibble: 5 × 1
##   activity
##   <chr>
## 1 student
## 2 professional
## 3 ?
## 4 unemployed
## 5 working-class
```

```
userprofile %>% distinct(ambience)
```

```
## # A tibble: 4 × 1
##   ambience
##   <chr>
## 1 family
## 2 friends
## 3 solitary
## 4 ?
```

```
userprofile %>% distinct(smoker)
```

```
## # A tibble: 3 × 1
##   smoker
##   <chr>
## 1 false
## 2 true
## 3 ?
```

```
# rating_final
# how many rows?
rating_final
```

```
## # A tibble: 1,161 × 5
##   userID placeID rating food_rating service_rating
##   <chr>   <int> <int>      <int>      <int>
## 1  U1077  135085     2         2         2
## 2  U1077  135038     2         2         1
## 3  U1077  132825     2         2         2
## 4  U1077  135060     1         2         2
## 5  U1068  135104     1         1         2
## 6  U1068  132740     0         0         0
## 7  U1068  132663     1         1         1
## 8  U1068  132732     0         0         0
## 9  U1068  132630     1         1         1
## 10 U1067  132584     2         2         2
## # ... with 1,151 more rows
```

```
rating_final %>% distinct(rating)
```

```
## # A tibble: 3 × 1
##   rating
##   <int>
## 1     2
## 2     1
## 3     0
```

```
rating_final %>% distinct(food_rating)
```

```
## # A tibble: 3 × 1
##   food_rating
##   <int>
## 1         2
## 2         1
## 3         0
```

```
rating_final %>% distinct(service_rating)
```

```
## # A tibble: 3 × 1
##   service_rating
##   <int>
## 1           2
## 2           1
## 3           0
```

```
# ratings are either 0, 1 or 2
```

notice that placeID is an integer value. However, it is the ID hence you will calculate n

```
rating_final <- rating_final %>%
  mutate(placeID = as.character(placeID))
```

```
chefmozcuisine <- chefmozcuisine %>%
  mutate(placeID = as.character(placeID))
```

- Based on the dataset `userprofile`, create a new dataset with only relevant variables. Among these, keep the variables: `userID`, `birth_year`, `budget`, `marital_status`, `personality`, `smoker` and `activity`. If you think there are other relevant variables, you may include them in your dataset as well. Call the new dataset “`userprofile_reduced`”.

```
userprofile_reduced <- userprofile %>%
  select(userID, birth_year, budget, marital_status, personality, smoker, activity)
```

```
userprofile_reduced
```

```
## # A tibble: 138 × 7
##   userID birth_year budget marital_status personality smoker
##   <chr>    <int> <chr>      <chr>      <chr>    <chr>
## 1  U1001    1989 medium    single    thrifty-protector false
## 2  U1002    1990  low     single    hunter-ostentatious false
## 3  U1003    1989  low     single    hard-worker false
## 4  U1004    1940 medium    single    hard-worker false
## 5  U1005    1992 medium    single    thrifty-protector false
## 6  U1006    1989 medium    single    hard-worker true
## 7  U1007    1989  low     single    thrifty-protector false
## 8  U1008    1989  low     single    hard-worker false
## 9  U1009    1991 medium    single    thrifty-protector false
## 10 U1010    1987 medium    married    hard-worker false
## # ... with 128 more rows, and 1 more variables: activity <chr>
```

- Focus on the data frame `userprofile_reduced`. By exploring the different values recorded for `budget`, you may notice there are missing values. What are they recorded by? Replace all missing values with `NA`. Do the same for all the variables in `userprofile_reduced`.

```
userprofile_reduced %>%
  distinct(budget)
```

```
## # A tibble: 4 × 1
##   budget
##   <chr>
## 1 medium
## 2  low
## 3   ?
## 4  high
```

```
# "?" is for missing values
userprofile_final <- userprofile_reduced %>% na_if("?")
```

5. Note that for all users we have the year of birth but we do not have the age. Replace the year of birth with a variable called age.

```
require(lubridate)

## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##      date

userprofile_final <- userprofile_final %>%
  mutate(age = year(today())-birth_year) %>%
  select(-birth_year)
```

6. All three data frames are now ready to use. Merge the three datasets so that you keep all rows and columns of rating and you add all the variables of chefmezcuisine.csv and userprofile.csv. Call the new data frame rating_all.

```
rating_all <- rating_final %>%
  left_join(userprofile_final) %>%
  left_join(chefmozcuisine)

## Joining, by = "userID"

## Joining, by = "placeID"
```

7. Find the mean of all rating variables. Group data by placeID and then sort the tibble so that places with the highest average rating are at the top. Show id of such places and type of cuisine.

```
rating_all %>%
  group_by(placeID) %>%
  summarise_at(vars(rating_mean = rating, food_rating_mean = food_rating,
                    service_rating_mean = service_rating), funs(mean)) %>%
  arrange(desc(rating_mean), desc(food_rating_mean), desc(service_rating_mean)) %>%
  left_join(chefmozcuisine) %>%
  select(c(placeID, rating_mean, food_rating_mean, service_rating_mean, Rcuisine))
```

```
## Joining, by = "placeID"
```

```
## # A tibble: 147 × 5
##   placeID rating_mean food_rating_mean service_rating_mean Rcuisine
##   <chr>      <dbl>          <dbl>          <dbl>          <chr>
## 1  134986    2.000000          2.00          2.000000 International
## 2  135034    2.000000          2.00          1.600000 Japanese
## 3  132955    2.000000          1.80          1.800000 Bar_Pub_Brewery
## 4  132922    1.833333          1.50          1.833333 Cafeteria
## 5  132755    1.800000          2.00          1.600000 Mexican
## 6  135013    1.750000          2.00          1.750000 <NA>
## 7  135074    1.750000          1.75          1.750000 Contemporary
## 8  134976    1.750000          1.75          1.000000 Mexican
## 9  134976    1.750000          1.75          1.000000 Mediterranean
## 10 134976    1.750000          1.75          1.000000 Burgers
## # ... with 137 more rows
```

8. Find mean and standard deviation of all rating variables. Do you notice differences with regards to ratings of students as compared to people that are employed? Do you find differences in smokers and non smokers? Do you notice large differences in any other group of users?

```
rating_all %>%
  group_by(activity) %>%
  summarise_at(vars(rating, food_rating, service_rating), funs(n(), mean, sd))
```

```
## # A tibble: 5 × 10
##   activity rating_n food_rating_n service_rating_n rating_mean
##   <chr>      <int>          <int>          <int>          <dbl>
## 1 professional    135          135          135    1.385185
## 2 student        1114          1114          1114    1.188510
## 3 unemployed      15           15           15    0.000000
## 4 working-class    4            4            4    1.500000
## 5 <NA>            63           63           63    1.365079
## # ... with 5 more variables: food_rating_mean <dbl>, service_rating_mean <dbl>,
## #   rating_sd <dbl>, food_rating_sd <dbl>, service_rating_sd <dbl>
```

```
rating_all %>%
  group_by(smoker) %>%
  summarise_at(vars(rating, food_rating, service_rating), funs(n(), mean, sd))
```

```
## # A tibble: 3 × 10
##   smoker rating_n food_rating_n service_rating_n rating_mean food_rating_mean
##   <chr>      <int>          <int>          <int>          <dbl>          <dbl>
## 1 false    1034          1034          1034    1.210832    1.223404
```



```
## 2   true      259      259      259   1.150579   1.108108
## 3   <NA>      38      38      38   1.394737   1.315789
## # ... with 4 more variables: service_rating_mean <dbl>, rating_sd <dbl>,
## #   food_rating_sd <dbl>, service_rating_sd <dbl>
```