**תרגיל 3.2.1**

מצא את כל הטיסות ש:

1. איחרו יותר משעתיים.
2. שטסו ליוסטון (IAH או HOU).
3. שאיחרו להגיע למעלה משעתיים אך לא איחרו לצאת.
4. שיצאו באיחור של למעלה משעה אך פיצו על לפחות חצי שעה מהאיחור בזמן אויר.
5. השתמשו בפונקציהbetween  כדי למצוא את הטיסות שיצאו בין השעה חצות ל-6 בבוקר. (שימו לב שהשעה חצות מיוצגת בנתונים כ-2400(

1. Since the arr\_delay variable is measured in minutes, find flights with an arrival delay of 120 or more minutes.

**filter**(flights, arr\_delay >= 120)

2. The flights that flew to Houston are those flights where the destination (dest) is either “IAH” or “HOU”.

**filter**(flights, dest == "IAH" | dest == "HOU")

However, using %in% is more compact and would scale to cases where there were more than two airports we were interested in.

**filter**(flights, dest %in% **c**("IAH", "HOU"))

3. Flights that arrived more than two hours late, but didn’t leave late will have an arrival delay of more than 120 minutes (arr\_delay > 120) and a non-positive departure delay (dep\_delay <= 0).

**filter**(flights, arr\_delay > 120, dep\_delay <= 0)

4. Were delayed by at least an hour, but made up over 30 minutes in flight. If a flight was delayed by at least an hour, then dep\_delay >= 60. If the flight didn’t make up any time in the air, then its arrival would be delayed by the same amount as its departure, meaning dep\_delay == arr\_delay, or alternatively, dep\_delay - arr\_delay == 0. If it makes up over 30 minutes in the air, then the arrival delay must be at least 30 minutes less than the departure delay, which is stated as dep\_delay - arr\_delay > 30.

**filter**(flights, dep\_delay >= 60, dep\_delay - arr\_delay > 30)

5. Finding flights that departed between midnight and 6 a.m. is complicated by the way in which times are represented in the data.  
In dep\_time, midnight is represented by 2400, not 0. You can verify this by checking the minimum and maximum of dep\_time.

**summary**(flights$dep\_time)

This is an example of why it is always good to check the summary statistics of your data. Unfortunately, this means we cannot simply check that dep\_time < 600, because we also have to consider the special case of midnight.

**filter**(flights, dep\_time <= 600 | dep\_time == 2400)

Alternatively, we could use the [modulo operator](https://en.wikipedia.org/wiki/Modulo_operation), %%. The modulo operator returns the remainder of division. Let’s see how this affects our times.

**c**(600, 1200, 2400) %% 2400

Since 2400 %% 2400 == 0 and all other times are left unchanged, we can compare the result of the modulo operation to 600,

**filter**(flights, dep\_time %% 2400 <= 600)

This filter expression is more compact, but its readability depends on the familiarity of the reader with modular arithmetic.

תרגיל 3.3.1

1. כיצד ניתן להשתמש ב arrange כדי שתציג ערכים חסרים (NA) בהתחלה?
2. מיין את flights  כדי למצוא את הטיסות שיצאו באיחור הגדול ביותר.
3. מצא את הטיסות שטסו במהירות הגבוהה ביותר (אין משתנה המציין את המהירות של הטיסה בנתונים)

1. The arrange() function puts NA values last.

**arrange**(flights, dep\_time) %>% **tail**()

Using desc() does not change that.

**arrange**(flights, **desc**(dep\_time))

To put NA values first, we can add an indicator of whether the column has a missing value. Then we sort by the missing indicator column and the column of interest. For example, to sort the data frame by departure time (dep\_time) in ascending order but NA values first, run the following.

**arrange**(flights, **desc**(**is.na**(dep\_time)), dep\_time)

The flights will first be sorted by desc(is.na(dep\_time)). Since desc(is.na(dep\_time)) is either TRUE when dep\_time is missing, or FALSE, when it is not, the rows with missing values of dep\_time will come first, since TRUE > FALSE.

2. Find the most delayed flights by sorting the table by departure delay, dep\_delay, in descending order.

**arrange**(flights, **desc**(dep\_delay))

Similarly, the earliest departing flight can be found by sorting dep\_delay in ascending order.

**arrange**(flights, dep\_delay)

3. There are actually two ways to interpret this question: one that can be solved by using arrange(), and a more complex interpretation that requires creation of a new variable using mutate(), which we haven’t seen demonstrated before.

The colloquial interpretation of “fastest” flight can be understood to mean “the flight with the shortest flight time”. We can use arrange to sort our data by the air\_time variable to find the shortest flights:

**head**(**arrange**(flights, air\_time))

Another definition of the “fastest flight” is the flight with the highest average [ground speed](https://en.wikipedia.org/wiki/Ground_speed). The ground speed is not included in the data, but it can be calculated from the distance and air\_time of the flight.

**head**(**arrange**(flights, **desc**(distance / air\_time)))

**תרגיל 3.4.1**

1. חשוב על כמה שיותר דרכים לסנן את העמודות : dep\_time, dep\_delay, arr\_time, arr\_delay.
2. מה קורה אם בוחרים באותה עמודה מספר פעמים?
3. בחן את הקוד הבא:
4. **select**(flights, **contains**("TIME"))

האם התוצאה מפתיעה? בחן את הארגומנטים שלtidyselect::contains כדי לשנות זאת.

1. These are a few ways to select columns.

Specify columns names as unquoted variable names.

**select**(flights, dep\_time, dep\_delay, arr\_time, arr\_delay)

Specify column names as strings.

**select**(flights, "dep\_time", "dep\_delay", "arr\_time", "arr\_delay")

Specify the column numbers of the variables.

**select**(flights, 4, 6, 7, 9)

This works, but is not good practice for two reasons. First, the column location of variables may change, resulting in code that may continue to run without error, but produce the wrong answer. Second code is obfuscated, since it is not clear from the code which variables are being selected. What variable does column 6 correspond to? I just wrote the code, and I’ve already forgotten.

Specify the names of the variables with character vector and any\_of() or all\_of()

**select**(flights, **all\_of**(**c**("dep\_time", "dep\_delay", "arr\_time", "arr\_delay")))

**select**(flights, **any\_of**(**c**("dep\_time", "dep\_delay", "arr\_time", "arr\_delay")))

This is useful because the names of the variables can be stored in a variable and passed to all\_of() or any\_of().

variables <- **c**("dep\_time", "dep\_delay", "arr\_time", "arr\_delay")

**select**(flights, **all\_of**(variables))

These two functions replace the deprecated function one\_of().

Selecting the variables by matching the start of their names using starts\_with().

**select**(flights, **starts\_with**("dep\_"), **starts\_with**("arr\_"))

Selecting the variables using regular expressions with matches(). Regular expressions provide a flexible way to match string patterns and are discussed in the [Strings](https://r4ds.had.co.nz/strings.html) chapter.

**select**(flights, **matches**("^(dep|arr)\_(time|delay)$"))

Specify the names of the variables with a character vector and use the bang-bang operator (!!).

variables <- **c**("dep\_time", "dep\_delay", "arr\_time", "arr\_delay")

**select**(flights, !!variables)

This and the following answers use the features of **tidy evaluation** not covered in R4DS (an also in our course) but covered in the [Programming with dplyr](https://dplyr.tidyverse.org/articles/programming.html) vignette.

Specify the names of the variables in a character or list vector and use the bang-bang-bang operator.

variables <- **c**("dep\_time", "dep\_delay", "arr\_time", "arr\_delay")

**select**(flights, !!!variables)

Specify the unquoted names of the variables in a list using syms() and use the bang-bang-bang operator.

variables <- **syms**(**c**("dep\_time", "dep\_delay", "arr\_time", "arr\_delay"))

**select**(flights, !!!variables)

Some things that **don’t** work are:

* Matching the ends of their names using ends\_with() since this will incorrectly include other variables. For example,

**select**(flights, **ends\_with**("arr\_time"), **ends\_with**("dep\_time"))

Matching the names using contains() since there is not a pattern that can include all these variables without incorrectly including others.

**select**(flights, **contains**("\_time"), **contains**("arr\_"))

2. What happens if you include the name of a variable multiple times in a select() call?

The select() call ignores the duplication. Any duplicated variables are only included once, in the first location they appear. The select() function does not raise an error or warning or print any message if there are duplicated variables.

**select**(flights, year, month, day, year, year)

This behavior is useful because it means that we can use select() with everything() in order to easily change the order of columns without having to specify the names of all the columns.

**select**(flights, arr\_delay, **everything**())

3.

**select**(flights, **contains**("TIME"))

The default behavior for contains() is to ignore case. This may or may not surprise you. If this behavior does not surprise you, that could be why it is the default. Users searching for variable names probably have a better sense of the letters in the variable than their capitalization. A second, technical, reason is that dplyr works with more than R data frames. It can also work with a variety of [databases](https://db.rstudio.com/dplyr/). Some of these database engines have case insensitive column names, so making functions that match variable names case insensitive by default will make the behavior of select() consistent regardless of whether the table is stored as an R data frame or in a database.

To change the behavior add the argument ignore.case = FALSE.

**select**(flights, **contains**("TIME", ignore.case = FALSE))

**תרגיל 3.5.1**

1. המשתנים dep\_time and sched\_dep\_time נוחים למבט אך לא נוחים לחישוב משום שאינם בעלי מספרים רציפים. המר את העמודות לפורמט אחר כך שיספרו את מספר הדקות מחצות. (כדאי להשתמש באופרטורים %%, %/%.)
2. תוך שימוש בפונקצית rank, מצא את 10 הטיסות שהגיעו באיחור הגדול ביותר. כיצד ברצונך לטפל בשוויון בערכים? קרא את ה help -שלmin\_rank.

1. To get the departure times in the number of minutes, divide dep\_time by 100 to get the hours since midnight and multiply by 60 and add the remainder of dep\_time divided by 100. For example, 1504 represents 15:04 (or 3:04 PM), which is 904 minutes after midnight. To generalize this approach, we need a way to split out the hour-digits from the minute-digits. Dividing by 100 and discarding the remainder using the integer division operator, %/% gives us the following.

1504 %/% 100

Instead of %/% could also use / along with trunc() or floor(), but round() would not work. To get the minutes, instead of discarding the remainder of the division by 100, we only want the remainder. So we use the modulo operator, %%, discussed in the [Other Useful Functions](https://r4ds.had.co.nz/transform.html#select) section.

1504 %% 100

Now, we can combine the hours (multiplied by 60 to convert them to minutes) and minutes to get the number of minutes after midnight.

1504 %/% 100 \* 60 + 1504 %% 100

There is one remaining issue. Midnight is represented by 2400, which would correspond to 1440 minutes since midnight, but it should correspond to 0. After converting all the times to minutes after midnight, x %% 1440 will convert 1440 to zero while keeping all the other times the same.

Now we will put it all together. The following code creates a new data frame flights\_times with columns dep\_time\_mins and sched\_dep\_time\_mins. These columns convert dep\_time and sched\_dep\_time, respectively, to minutes since midnight.

flights\_times <- **mutate**(flights,

dep\_time\_mins = (dep\_time %/% 100 \* 60 + dep\_time %% 100) %% 1440,

sched\_dep\_time\_mins = (sched\_dep\_time %/% 100 \* 60 +

sched\_dep\_time %% 100) %% 1440

)

*# view only relevant columns*

**select**(

flights\_times, dep\_time, dep\_time\_mins, sched\_dep\_time,

sched\_dep\_time\_mins

)

Looking ahead to the [Functions](https://r4ds.had.co.nz/functions.html) chapter, this is precisely the sort of situation in which it would make sense to write a function to avoid copying and pasting code. We could define a function time2mins(), which converts a vector of times in from the format used in flights to minutes since midnight.

time2mins <- **function**(x) {

(x %/% 100 \* 60 + x %% 100) %% 1440

}

Using time2mins, the previous code simplifies to the following.

flights\_times <- **mutate**(flights,

dep\_time\_mins = **time2mins**(dep\_time),

sched\_dep\_time\_mins = **time2mins**(sched\_dep\_time)

)

*# show only the relevant columns*

**select**(

flights\_times, dep\_time, dep\_time\_mins, sched\_dep\_time,

sched\_dep\_time\_mins

)

2. The **dplyr** package provides multiple functions for ranking, which differ in how they handle tied values: row\_number(), min\_rank(), dense\_rank(). To see how they work, let’s create a data frame with duplicate values in a vector and see how ranking functions handle ties.

rankme <- **tibble**(

x = **c**(10, 5, 1, 5, 5)

)

rankme <- **mutate**(rankme,

x\_row\_number = **row\_number**(x),

x\_min\_rank = **min\_rank**(x),

x\_dense\_rank = **dense\_rank**(x)

)

**arrange**(rankme, x)

The function row\_number() assigns each element a unique value. The result is equivalent to the index (or row) number of each element after sorting the vector, hence its name.

The min\_rank() and dense\_rank() assign tied values the same rank, but differ in how they assign values to the next rank. For each set of tied values the min\_rank() function assigns a rank equal to the number of values less than that tied value plus one. In contrast, the dense\_rank() function assigns a rank equal to the number of distinct values less than that tied value plus one. To see the difference between dense\_rank() and min\_rank() compare the value of rankme$x\_min\_rank and rankme$x\_dense\_rank for x = 10.

If I had to choose one for presenting rankings to someone else, I would use min\_rank() since its results correspond to the most common usage of rankings in sports or other competitions. In the code below, I use all three functions, but since there are no ties in the top 10 flights, the results don’t differ.

flights\_delayed <- **mutate**(flights,

dep\_delay\_min\_rank = **min\_rank**(**desc**(dep\_delay)),

dep\_delay\_row\_number = **row\_number**(**desc**(dep\_delay)),

dep\_delay\_dense\_rank = **dense\_rank**(**desc**(dep\_delay))

)

flights\_delayed <- **filter**(flights\_delayed,

!(dep\_delay\_min\_rank > 10 | dep\_delay\_row\_number > 10 |

dep\_delay\_dense\_rank > 10))

flights\_delayed <- **arrange**(flights\_delayed, dep\_delay\_min\_rank)

**print**(**select**(flights\_delayed, month, day, carrier, flight, dep\_delay,

dep\_delay\_min\_rank, dep\_delay\_row\_number, dep\_delay\_dense\_rank),

n = Inf)

In addition to the functions covered here, the rank() function provides several more ways of ranking elements.

There are other ways to solve this problem that do not using ranking functions. To select the top 10, sort values with arrange() and select the top values with slice:

flights\_delayed2 <- **arrange**(flights, **desc**(dep\_delay))

flights\_delayed2 <- **slice**(flights\_delayed2, 1:10)

**select**(flights\_delayed2, month, day, carrier, flight, dep\_delay)

Alternatively, we could use the top\_n().

flights\_delayed3 <- **top\_n**(flights, 10, dep\_delay)

flights\_delayed3 <- **arrange**(flights\_delayed3, **desc**(dep\_delay))

**select**(flights\_delayed3, month, day, carrier, flight, dep\_delay)

The previous two approaches will always select 10 rows even if there are tied values. Ranking functions provide more control over how tied values are handled. Those approaches will provide the 10 rows with the largest values of dep\_delay, while ranking functions can provide all rows with the 10 largest values of dep\_delay. If there are no ties, these approaches are equivalent. If there are ties, then which is more appropriate depends on the use.

תרגיל 3.11.1

1. מצא חלופות שונות אשר יניבו את אותן תוצאות כמו הקוד הבא:

not\_cancelled %>% count(dest)  
not\_cancelled %>% count(tailnum, wt = distance)

1. מה הארגומנט sort עושה בפונקציה count. מתי תשתמש בו?
2. מה ההבדל בין האופן שבו פועלת הפונקציות +, log בתוך grouped data לבין פונקציות כגון mean, lead, lag? האם תוכל לאפיין את ההבדל בין שני הסוגים?
3. באיזו שעה של היום כדאי לטוס אם ברצונך להימנע מאיחור בהגעה?
4. מצא את כל היעדים אשר מגיעים אליהם לפחות 2 חברות תעופה. דרג את היעדים לפי מספר חברות התעופה שמגיעים אליהם.

1.

not\_cancelled <- flights %>%

**filter**(!**is.na**(dep\_delay), !**is.na**(arr\_delay))

The first expression is the following.

not\_cancelled %>% **count**(dest)

The count() function counts the number of instances within each group of variables. Instead of using the count() function, we can combine the group\_by() and summarise() verbs.

not\_cancelled %>%

**group\_by**(dest) %>%

**summarise**(n = **length**(dest))

An alternative method for getting the number of observations in a data frame is the function n().

not\_cancelled %>%

**group\_by**(dest) %>%

**summarise**(n = **n**())

Another alternative to count() is to use group\_by() followed by tally(). In fact, count() is effectively a short-cut for group\_by() followed by tally().

not\_cancelled %>%

**group\_by**(tailnum) %>%

**tally**()

The second expression also uses the count() function, but adds a wt argument.

not\_cancelled %>%

**count**(tailnum, wt = distance)

Like the previous example, we can also use the combination group\_by() and tally(). Any arguments to tally() are summed.

not\_cancelled %>%

**group\_by**(tailnum) %>%

**tally**(distance)

2. The sort argument to count() sorts the results in order of n. You could use this anytime you would run count() followed by arrange().

For example, the following expression counts the number of flights to a destination and sorts the returned data from highest to lowest.

flights %>%

**count**(dest, sort = TRUE)

3. Summary functions (mean()), offset functions (lead(), lag()), ranking functions (min\_rank(), row\_number()), operate within each group when used with group\_by() in mutate() or filter(). Arithmetic operators (+, -), logical operators (<, ==), modular arithmetic operators (%%, %/%), logarithmic functions (log) are not affected by group\_by.

Summary functions like mean(), median(), sum(), std() and others covered in the section [Useful Summary Functions](https://r4ds.had.co.nz/transform.html#summarise-funs) calculate their values within each group when used with mutate() or filter() and group\_by().

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(x\_mean = **mean**(x)) %>%

**group\_by**(group) %>%

**mutate**(x\_mean\_2 = **mean**(x))

Arithmetic operators +, -, \*, /, ^ are not affected by group\_by().

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(y = x + 2) %>%

**group\_by**(group) %>%

**mutate**(z = x + 2)

The modular arithmetic operators %/% and %% are not affected by group\_by()

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(y = x %% 2) %>%

**group\_by**(group) %>%

**mutate**(z = x %% 2)

The logarithmic functions log(), log2(), and log10() are not affected by group\_by().

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(y = **log**(x)) %>%

**group\_by**(group) %>%

**mutate**(z = **log**(x))

The offset functions lead() and lag() respect the groupings in group\_by(). The functions lag() and lead() will only return values within each group.

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**group\_by**(group) %>%

**mutate**(lag\_x = **lag**(x),

The cumulative and rolling aggregate functions cumsum(), cumprod(), cummin(), cummax(), and cummean() calculate values within each group.

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(x\_cumsum = **cumsum**(x)) %>%

**group\_by**(group) %>%

**mutate**(x\_cumsum\_2 = **cumsum**(x))

Logical comparisons, <, <=, >, >=, !=, and == are not affected by group\_by().

**tibble**(x = 1:9,

y = 9:1,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(x\_lte\_y = x <= y) %>%

**group\_by**(group) %>%

**mutate**(x\_lte\_y\_2 = x <= y)

Ranking functions like min\_rank() work within each group when used with group\_by().

**tibble**(x = 1:9,

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**mutate**(rnk = **min\_rank**(x)) %>%

**group\_by**(group) %>%

**mutate**(rnk2 = **min\_rank**(x))

Though not asked in the question, note that arrange() ignores groups when sorting values.

**tibble**(x = **runif**(9),

group = **rep**(**c**("a", "b", "c"), each = 3)) %>%

**group\_by**(group) %>%

**arrange**(x)

However, the order of values from arrange() can interact with groups when used with functions that rely on the ordering of elements, such as lead(), lag(), or cumsum().

**tibble**(group = **rep**(**c**("a", "b", "c"), each = 3),

x = **runif**(9)) %>%

**group\_by**(group) %>%

**arrange**(x) %>%

**mutate**(lag\_x = **lag**(x))

4. Let’s group by the hour of the flight. The earlier the flight is scheduled, the lower its expected delay. This is intuitive as delays will affect later flights. Morning flights have fewer (if any) previous flights that can delay them.

flights %>%

**group\_by**(hour) %>%

**summarise**(arr\_delay = **mean**(arr\_delay, na.rm = TRUE)) %>%

**arrange**(arr\_delay)

5. To restate this question, we are asked to rank airlines by the number of destinations that they fly to, considering only those airports that are flown to by two or more airlines. There are two steps to calculating this ranking. First, find all airports serviced by two or more carriers. Then, rank carriers by the number of those destinations that they service.

flights %>%

*# find all airports with > 1 carrier*

**group\_by**(dest) %>%

**mutate**(n\_carriers = **n\_distinct**(carrier)) %>%

**filter**(n\_carriers > 1) %>%

*# rank carriers by numer of destinations*

**group\_by**(carrier) %>%

**summarize**(n\_dest = **n\_distinct**(dest)) %>%

**arrange**(**desc**(n\_dest))

The carrier "EV" flies to the most destinations, considering only airports flown to by two or more carriers. What airline does the "EV" carrier code correspond to?

**filter**(airlines, carrier == "EV")

Unless you know the airplane industry, it is likely that you don’t recognize [ExpressJet](https://en.wikipedia.org/wiki/ExpressJet); I certainly didn’t. It is a regional airline that partners with major airlines to fly from hubs (larger airports) to smaller airports. This means that many of the shorter flights of major carriers are operated by ExpressJet. This business model explains why ExpressJet services the most destinations.

Among the airlines that fly to only one destination from New York are Alaska Airlines and Hawaiian Airlines.

**filter**(airlines, carrier %in% **c**("AS", "F9", "HA"))