Professor McDonald   
FNCE 5352 – Financial Programming and Modeling   
March 13, 2019  
Solutions to March 5 Assignment

**5.2.4 Exercise 1**

Find all flights that

1. Had an arrival delay of two or more hours
2. Flew to Houston (IAH or HOU)
3. Were operated by United, American, or Delta
4. Departed in summer (July, August, and September)
5. Arrived more than two hours late, but didn’t leave late
6. Were delayed by at least an hour, but made up over 30 minutes in flight
7. Departed between midnight and 6 am (inclusive)

The answer to each part follows.

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)

*#> # A tibble: 10,200 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 811 630 101 1047*

*#> 2 2013 1 1 848 1835 853 1001*

*#> 3 2013 1 1 957 733 144 1056*

*#> 4 2013 1 1 1114 900 134 1447*

*#> 5 2013 1 1 1505 1310 115 1638*

*#> 6 2013 1 1 1525 1340 105 1831*

*#> # … with 1.019e+04 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

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

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

*#> # A tibble: 9,313 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 517 515 2 830*

*#> 2 2013 1 1 533 529 4 850*

*#> 3 2013 1 1 623 627 -4 933*

*#> 4 2013 1 1 728 732 -4 1041*

*#> 5 2013 1 1 739 739 0 1104*

*#> 6 2013 1 1 908 908 0 1228*

*#> # … with 9,307 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

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"))

*#> # A tibble: 9,313 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 517 515 2 830*

*#> 2 2013 1 1 533 529 4 850*

*#> 3 2013 1 1 623 627 -4 933*

*#> 4 2013 1 1 728 732 -4 1041*

*#> 5 2013 1 1 739 739 0 1104*

*#> 6 2013 1 1 908 908 0 1228*

*#> # … with 9,307 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

1. In the flights dataset, the column carrier indicates the airline, but it uses two-character carrier codes. We can find the carrier codes for the airlines in the airlines dataset. Since the carrier code dataset only has 16 rows, and the names of the airlines in that dataset are not exactly “United”, “American”, or “Delta”, it is easiest to manually look up their carrier codes in that data.

airlines

*#> # A tibble: 16 x 2*

*#> carrier name*

*#> <chr> <chr>*

*#> 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.*

*#> # … with 10 more rows*

The carrier code for Delta is "DL", for American is "AA", and for United is "UA". Using these carriers codes, we check whether carrier is one of those.

**filter**(flights, carrier %in% **c**("AA", "DL", "UA"))

*#> # A tibble: 139,504 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 517 515 2 830*

*#> 2 2013 1 1 533 529 4 850*

*#> 3 2013 1 1 542 540 2 923*

*#> 4 2013 1 1 554 600 -6 812*

*#> 5 2013 1 1 554 558 -4 740*

*#> 6 2013 1 1 558 600 -2 753*

*#> # … with 1.395e+05 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

1. The variable month has the month, and it is numeric. So, the summer flights are those that departed in months 7 (July), 8 (August), and 9 (September).

**filter**(flights, month >= 7, month <= 9)

*#> # A tibble: 86,326 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 7 1 1 2029 212 236*

*#> 2 2013 7 1 2 2359 3 344*

*#> 3 2013 7 1 29 2245 104 151*

*#> 4 2013 7 1 43 2130 193 322*

*#> 5 2013 7 1 44 2150 174 300*

*#> 6 2013 7 1 46 2051 235 304*

*#> # … with 8.632e+04 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

The %in% operator is an alternative. If the : operator is used to specify the integer range, the expression is readable and compact.

**filter**(flights, month %in% 7:9)

*#> # A tibble: 86,326 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 7 1 1 2029 212 236*

*#> 2 2013 7 1 2 2359 3 344*

*#> 3 2013 7 1 29 2245 104 151*

*#> 4 2013 7 1 43 2130 193 322*

*#> 5 2013 7 1 44 2150 174 300*

*#> 6 2013 7 1 46 2051 235 304*

*#> # … with 8.632e+04 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

We could also use the | operator. However, the | does not scale to many choices. Even with only three choices, it is quite verbose.

**filter**(flights, month == 7 | month == 8 | month == 9)

*#> # A tibble: 86,326 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 7 1 1 2029 212 236*

*#> 2 2013 7 1 2 2359 3 344*

*#> 3 2013 7 1 29 2245 104 151*

*#> 4 2013 7 1 43 2130 193 322*

*#> 5 2013 7 1 44 2150 174 300*

*#> 6 2013 7 1 46 2051 235 304*

*#> # … with 8.632e+04 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

We can also use the between() function as shown in [Exercise 5.2.2](https://jrnold.github.io/r4ds-exercise-solutions/transform.html#exercise-5.2.2).

1. 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)

*#> # A tibble: 29 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 27 1419 1420 -1 1754*

*#> 2 2013 10 7 1350 1350 0 1736*

*#> 3 2013 10 7 1357 1359 -2 1858*

*#> 4 2013 10 16 657 700 -3 1258*

*#> 5 2013 11 1 658 700 -2 1329*

*#> 6 2013 3 18 1844 1847 -3 39*

*#> # … with 23 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

1. 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)

*#> # A tibble: 1,844 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 2205 1720 285 46*

*#> 2 2013 1 1 2326 2130 116 131*

*#> 3 2013 1 3 1503 1221 162 1803*

*#> 4 2013 1 3 1839 1700 99 2056*

*#> 5 2013 1 3 1850 1745 65 2148*

*#> 6 2013 1 3 1941 1759 102 2246*

*#> # … with 1,838 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

1. 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. 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)

*#> # A tibble: 9,373 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 517 515 2 830*

*#> 2 2013 1 1 533 529 4 850*

*#> 3 2013 1 1 542 540 2 923*

*#> 4 2013 1 1 544 545 -1 1004*

*#> 5 2013 1 1 554 600 -6 812*

*#> 6 2013 1 1 554 558 -4 740*

*#> # … with 9,367 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

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 how this affects our times.

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

*#> [1] 600 1200 0*

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)

*#> # A tibble: 9,373 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 517 515 2 830*

*#> 2 2013 1 1 533 529 4 850*

*#> 3 2013 1 1 542 540 2 923*

*#> 4 2013 1 1 544 545 -1 1004*

*#> 5 2013 1 1 554 600 -6 812*

*#> 6 2013 1 1 554 558 -4 740*

*#> # … with 9,367 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

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

### 5.2.4 Exercise 3

How many flights have a missing dep\_time? What other variables are missing? What might these rows represent?

Find the rows of flights with a missing departure time (dep\_time) using the is.na() function.

**filter**(flights, **is.na**(dep\_time))

*#> # A tibble: 8,255 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 NA 1630 NA NA*

*#> 2 2013 1 1 NA 1935 NA NA*

*#> 3 2013 1 1 NA 1500 NA NA*

*#> 4 2013 1 1 NA 600 NA NA*

*#> 5 2013 1 2 NA 1540 NA NA*

*#> 6 2013 1 2 NA 1620 NA NA*

*#> # … with 8,249 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

Notably, the arrival time (arr\_time) is also missing for these rows. These seem to be cancelled flights.

### 5.3.1 Exercise 1

How could you use arrange() to sort all missing values to the start? (Hint: use is.na()).

The arrange() function puts NA values last.

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

**tail**()

*#> # A tibble: 6 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 9 30 NA 1842 NA NA*

*#> 2 2013 9 30 NA 1455 NA NA*

*#> 3 2013 9 30 NA 2200 NA NA*

*#> 4 2013 9 30 NA 1210 NA NA*

*#> 5 2013 9 30 NA 1159 NA NA*

*#> 6 2013 9 30 NA 840 NA NA*

*#> # … with 12 more variables: sched\_arr\_time <int>, arr\_delay <dbl>,*

*#> # carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,*

*#> # air\_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,*

*#> # time\_hour <dttm>*

Using desc() does not change that.

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

*#> # A tibble: 336,776 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 10 30 2400 2359 1 327*

*#> 2 2013 11 27 2400 2359 1 515*

*#> 3 2013 12 5 2400 2359 1 427*

*#> 4 2013 12 9 2400 2359 1 432*

*#> 5 2013 12 9 2400 2250 70 59*

*#> 6 2013 12 13 2400 2359 1 432*

*#> # … with 3.368e+05 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

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 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)

*#> # A tibble: 336,776 x 19*

*#> year month day dep\_time sched\_dep\_time dep\_delay arr\_time*

*#> <int> <int> <int> <int> <int> <dbl> <int>*

*#> 1 2013 1 1 NA 1630 NA NA*

*#> 2 2013 1 1 NA 1935 NA NA*

*#> 3 2013 1 1 NA 1500 NA NA*

*#> 4 2013 1 1 NA 600 NA NA*

*#> 5 2013 1 2 NA 1540 NA NA*

*#> 6 2013 1 2 NA 1620 NA NA*

*#> # … with 3.368e+05 more rows, and 12 more variables: sched\_arr\_time <int>,*

*#> # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,*

*#> # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,*

*#> # minute <dbl>, time\_hour <dttm>*

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.

### 5.5.2 Exercise 2

Compare air\_time with arr\_time - dep\_time. What do you expect to see? What do you see? What do you need to do to fix it?

I expect that air\_time is the difference between the arrival (arr\_time) and departure times (dep\_time). In other words, air\_time = arr\_time - dep\_time.

To check that this relationship, I’ll first need to convert the times to a form more amenable to arithmetic operations using the same calculations as the [previous exercise](https://jrnold.github.io/r4ds-exercise-solutions/transform.html#exercise-5.5.1).

flights\_airtime <-

**mutate**(flights,

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

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

air\_time\_diff = air\_time - arr\_time + dep\_time

)

So, does air\_time = arr\_time - dep\_time? If so, there should be no flights with non-zero values of air\_time\_diff.

**nrow**(**filter**(flights\_airtime, air\_time\_diff != 0))

*#> [1] 327150*

It turns out that there are many flights for which air\_time != arr\_time - dep\_time. Other than data errors, I can think of two reasons why air\_time would not equal arr\_time - dep\_time.

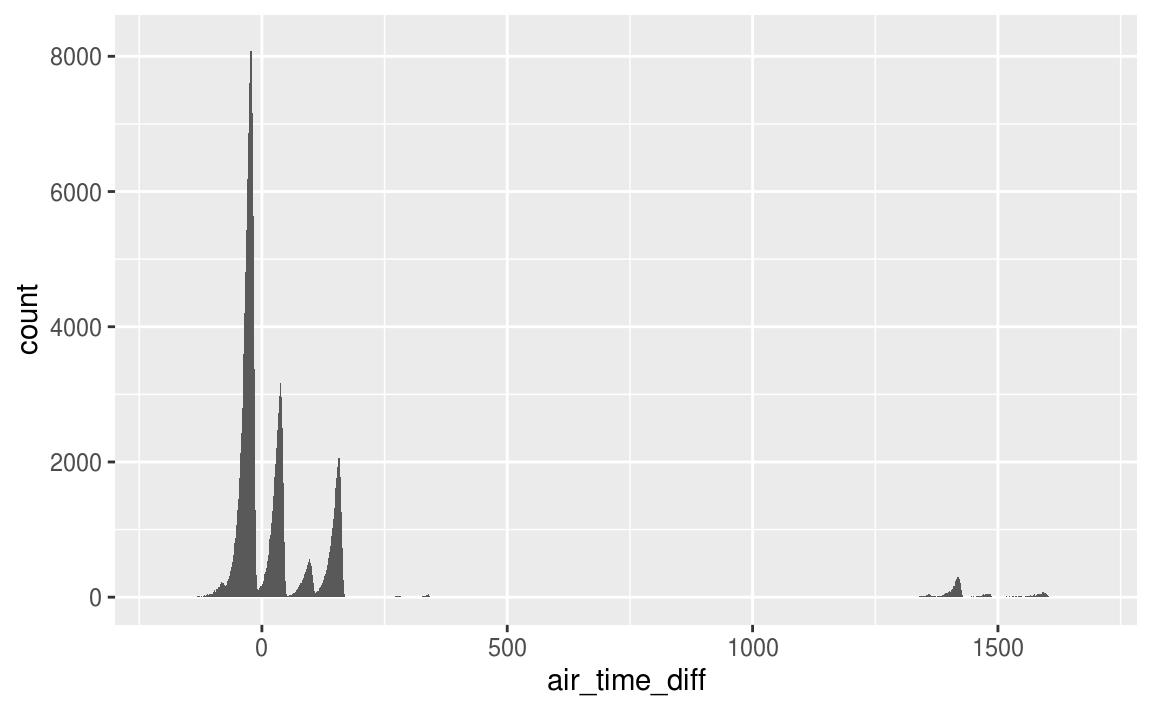
1. The flight passes midnight, so arr\_time < dep\_time. In these cases, the difference in airtime should be by 24 hours (1,440 minutes).
2. The flight crosses time zones, and the total air time will be off by hours (multiples of 60). All flights in flights departed from New York City and are domestic flights in the US. This means that flights will all be to the same or more westerly time zones. Given the time-zones in the US, the differences due to time-zone should be 60 minutes (Central) 120 minutes (Mountain), 180 minutes (Pacific), 240 minutes (Alaska), or 300 minutes (Hawaii).

Both of these explanations have clear patterns that I would expect to see if they were true. In particular, in both cases, since time-zones and crossing midnight only affects the hour part of the time, all values of air\_time\_diff should be divisible by 60. I’ll visually check this hypothesis by plotting the distribution of air\_time\_diff. If those two explanations are correct, distribution of air\_time\_diffshould comprise only spikes at multiples of 60.

**ggplot**(flights\_airtime, **aes**(x = air\_time\_diff)) +

**geom\_histogram**(binwidth = 1)

*#> Warning: Removed 9430 rows containing non-finite values (stat\_bin).*

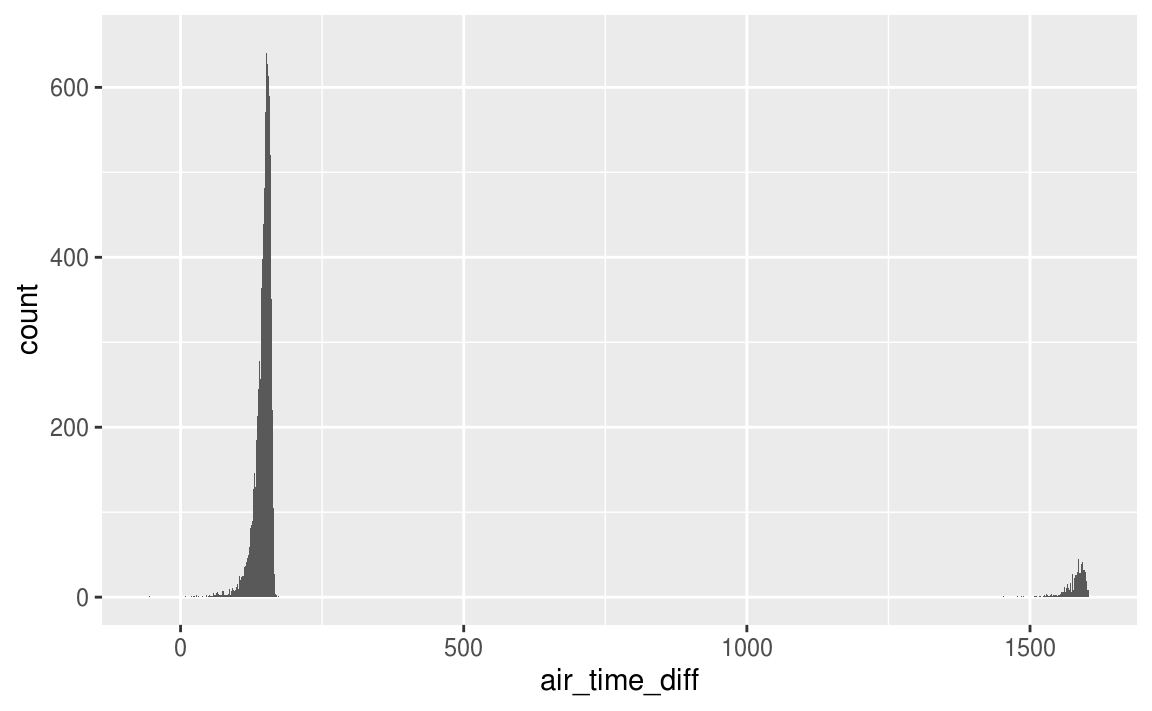
This is not the case. While, the distribution of air\_time\_diff has modes at multiples of 60 as hypothesized, it shows that there are many flights in which the difference between air time and local arrival and departure times is not divisible by 60.

Let’s also look at flights with Los Angeles as a destination. The discrepancy should be 180 minutes.

**ggplot**(**filter**(flights\_airtime, dest == "LAX"), **aes**(x = air\_time\_diff)) +

**geom\_histogram**(binwidth = 1)

*#> Warning: Removed 148 rows containing non-finite values (stat\_bin).*



To fix these time-zone issues, I would want to convert all the times to a date-time to handle overnight flights, and from local time to a common time zone, most likely [UTC](https://en.wikipedia.org/wiki/Coordinated_Universal_Time), to handle flights crossing time-zones. The tzone column of nycflights13::airports gives the time-zone of each airport. See the [“Dates and Times”](https://r4ds.had.co.nz/dates-and-times.html) for an introduction on working with date and time data.

But that still leaves the other differences unexplained. So what else might be going on? There seem to be too many problems for this to be data entry problems, so I’m probably missing something. So, I’ll reread the documentation to make sure that I understand the definitions of arr\_time, dep\_time, andair\_time. The documentation contains a link to the source of the flights data, <https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236>. This documentation shows that the flights data does not contain the variables TaxiIn, TaxiOff, WheelsIn, and WheelsOff. It appears that the air\_time variable refers to flight time, which is defined as the time between wheels-off (take-off) and wheels-in (landing). But the flight time does not include time spent on the runway taxiing to and from gates. With this new understanding of the data, I now know that the relationship between air\_time, arr\_time, and dep\_time is air\_time <= arr\_time - dep\_time, supposing that the time zones of arr\_time and dep\_time are in the same time zone.

### 5.5.2 Exercise 5

What does 1:3 + 1:10 return? Why?

The code given in the question returns the following.

1:3 + 1:10

*#> Warning in 1:3 + 1:10: longer object length is not a multiple of shorter*

*#> object length*

*#> [1] 2 4 6 5 7 9 8 10 12 11*

This is equivalent to the following.

**c**(1 + 1, 2 + 2, 3 + 3, 1 + 4, 2 + 5, 3 + 6, 1 + 7, 2 + 8, 3 + 9, 1 + 10)

*#> [1] 2 4 6 5 7 9 8 10 12 11*

When adding two vectors recycles the shorter vector’s values to get vectors of the same length.

The code also produces a warning that the shorter vector is not a multiple of the longer vector. A warning is provided since often, but not always, this indicates a bug in the code.