Project Coding in R

Ken Ye

9/20/2023

1. Announcements and References

This lab will be focused on various data scraping, subsetting and manipulation exercises to help you for your projects. We'll work through these examples as a class.

References:

- Interpreting interaction terms in regression:
- https://stats.idre.ucla.edu/r/faq/how-can-i-explain-a-continuous-by-continuous-interaction/
- R For Data Science: https://r4ds.had.co.nz/
- STA 523 at Duke: http://www2.stat.duke.edu/~cr173/Sta523_Fa18/
- Vignette on Web Scraping in R: https://cran.r-project.org/web/packages/rvest/vignettes/selectorgadget.html

2. Data Transformation and Visualization

This part of the lab follows Ch. 5 from https://r4ds.had.co.nz/. For full details and explanations of functions, please see this book, *R for Data Science*. We will only work with the code portions here. We will cover `dplyr` and `ggplot2`.

The dataset of interest will be `nycflights13`, which contains information about all 336,776 flights that departed from NYC in 2013.

First, in the console, install the packages `nycflights13` and `tidyverse`.

```
library("nycflights13")
library("tidyverse")
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      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
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

The `flights` data is stored as a **tibble**, which is a dataframe with some nice properties that work well in the `tidyverse` packages. Under each column name, there is information about the data type of each variable. The data types for this data include:

- `int`: integer
- `dbl`: double precision, real number
- `chr`: character vectors or strings
- `dttm`: date-time

We are going to focus on 5 important `dplyr` functions that can be used to manipulate and subset data. These functions are:

1. `filter()`: pick observations by their values (works on rows)
2. `arrange()`: reorder the rows of a dataframe or tibble
3. `select()`: pick variables by their names (works on columns)
4. `mutate()`: create new variables with functions of existing variables
5. `summarise()`: create summary values by collpasing many values

Another useful function to know is `group_by()`. Instead of operating on the entire dataframe, `group_by()` changes the function so that it only works on a group-by-group basis.

For all of these functions, the syntax is as follows:

- 1st Argument: Data Frame
- Following Argument(s): describe what to do with the dataframe, use variable names without quotations here
- Result: new data frame

filter()

This function can be used to subset observations, based on their values (i.e. selecting by row). Below is code to select all flights on January 1st:

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      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
## # i 832 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

`dplyr` never modifies the input, so if we want to have a new data frame to save the result, we need to use the assignment operator to assign the filtered dataframe to a new name:

```
jan1 <- filter(flights, month == 1, day == 1)
```

Suppose we want to select all rows for flights that left early. How would we use the `filter()` function to do this?

```
flights %>% filter(dep_delay < 0)
```

```
## # A tibble: 183,575 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      544            545        -1     1004           1022
##  2  2013     1     1      554            600        -6      812            837
##  3  2013     1     1      554            558        -4      740            728
##  4  2013     1     1      555            600        -5      913            854
##  5  2013     1     1      557            600        -3      709            723
##  6  2013     1     1      557            600        -3      838            846
##  7  2013     1     1      558            600        -2      753            745
##  8  2013     1     1      558            600        -2      849            851
##  9  2013     1     1      558            600        -2      853            856
## 10  2013     1     1      558            600        -2      924            917
## # i 183,565 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Q: Is the above output the same as `filter(flights, dep_delay < 0)`? Check.

```
filter(flights,dep_delay < 0)
```

```
## # A tibble: 183,575 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      544            545        -1     1004           1022
##  2  2013     1     1      554            600        -6      812            837
##  3  2013     1     1      554            558        -4      740            728
##  4  2013     1     1      555            600        -5      913            854
##  5  2013     1     1      557            600        -3      709            723
##  6  2013     1     1      557            600        -3      838            846
##  7  2013     1     1      558            600        -2      753            745
##  8  2013     1     1      558            600        -2      849            851
##  9  2013     1     1      558            600        -2      853            856
## 10  2013     1     1      558            600        -2      924            917
## # i 183,565 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Yes, they are the same.

When filtering, we need to use comparison operators to do so. What are the symbols for the 6 primary comparison operators in `R` and what does each do?

```
# >
# >=
# <
# <=
# !=
# ==
```

Be careful with floating point numbers when testing for equality!

```
sqrt(2) ^ 2 == 2
```

```
## [1] FALSE
```

```
1 / 49 * 49 == 1
```

```
## [1] FALSE
```

Why are these results not as expected?

```
## Finite precision for computers
```

When we want to test equality with floating point numbers, we can use the `near()` function instead:

```
near(sqrt(2) ^ 2,  2)
```

```
## [1] TRUE
```

```
near(1 / 49 * 49, 1)
```

```
## [1] TRUE
```

Boolean operators can be used to use multiple arguments. The Boolean operator for *and* is `&`, for *or* is `|` and for *not* is `!`.

One common logical operation is exclusive or, `XOR`. Visually, what does this logical operation look like for a venn diagram with variables `x` and `y`?

```
## XOR is x or y but not both, middle of venn diagram is only thing not shaded
```

This code finds all flights that departed in November or December:

```
filter(flights, month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    11     1        5           2359         6      352            345
##  2  2013    11     1       35           2250       105      123           2356
##  3  2013    11     1      455            500        -5      641            651
```

```
##  4  2013    11     1      539            545        -6      856            827
##  5  2013    11     1      542            545        -3      831            855
##  6  2013    11     1      549            600       -11      912            923
##  7  2013    11     1      550            600       -10      705            659
##  8  2013    11     1      554            600        -6      659            701
##  9  2013    11     1      554            600        -6      826            827
## 10  2013    11     1      554            600        -6      749            751
## # i 55,393 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

How would we select all flights that had an arrival delay of less than 10 minutes and occurred in September?

```
flights %>% filter(arr_delay < 10 & month == 9)
```

```
## # A tibble: 22,501 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     9     1        9           2359        10      343            340
##  2  2013     9     1      508            516        -8      717            800
##  3  2013     9     1      537            545        -8      849            855
##  4  2013     9     1      537            545        -8      906            921
##  5  2013     9     1      549            600       -11      815            850
##  6  2013     9     1      552            600        -8      843            905
##  7  2013     9     1      553            600        -7      809            834
##  8  2013     9     1      554            600        -6      700            716
##  9  2013     9     1      554            600        -6      803            823
## 10  2013     9     1      554            600        -6      657            715
## # i 22,491 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Another really useful comparison function is `%in%`. For example, `x %in% y` selects every row where `x` is one of the values in `y`. For example, the two following lines of code are equivalent:

```
filter(flights, month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    11     1        5           2359         6      352            345
##  2  2013    11     1       35           2250       105      123           2356
##  3  2013    11     1      455            500        -5      641            651
##  4  2013    11     1      539            545        -6      856            827
##  5  2013    11     1      542            545        -3      831            855
##  6  2013    11     1      549            600       -11      912            923
##  7  2013    11     1      550            600       -10      705            659
##  8  2013    11     1      554            600        -6      659            701
##  9  2013    11     1      554            600        -6      826            827
## 10  2013    11     1      554            600        -6      749            751
```

```
## # i 55,393 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
filter(flights, month %in% c(11, 12))
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    11     1        5           2359         6      352            345
##  2  2013    11     1       35           2250       105      123           2356
##  3  2013    11     1      455            500        -5      641            651
##  4  2013    11     1      539            545        -6      856            827
##  5  2013    11     1      542            545        -3      831            855
##  6  2013    11     1      549            600       -11      912            923
##  7  2013    11     1      550            600       -10      705            659
##  8  2013    11     1      554            600        -6      659            701
##  9  2013    11     1      554            600        -6      826            827
## 10  2013    11     1      554            600        -6      749            751
## # i 55,393 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

De Morgan's law can be used for complicated expressions. What is De Morgan's Law (from probability) and how would we write it in terms of R code?

```
# !(x & y) same as !x | !y
# !(x | y) same as !x & !y
```

We can use De Morgan's law to select flights that weren't delayed (on arrival or departure) by more than two hours:

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
```

```
## # A tibble: 316,050 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      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
## # i 316,040 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

```
## # A tibble: 316,050 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      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
## # i 316,040 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Most real life data includes missing values. What are some possible reasons that data could be missing and what might the implications be for later modeling?

Missing values are represented as `NA` in `R`. `NA` values are "contagious", so almost any operation involving an `NA` will also be `NA`.

```
NA > 5
```

```
## [1] NA
```

```
10 == NA
```

```
## [1] NA
```

```
NA + 10
```

```
## [1] NA
```

```
NA / 2
```

```
## [1] NA
```

What do you expect the result to be for the following code:

```
NA == NA
```

```
NA == NA
```

```
## [1] NA
```

We can use the `is.na()` function to determine if a value is missing or is `NA`.

The `filter()` function only includes rows that are `TRUE`, `FALSE` and `NA` rows are excluded.

arrange()

`arrange()` changes the order of rows instead of selecting them.

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

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      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
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

The `desc()` function can be used to re-order by a specific column in descending order:

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     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     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Missing values are always sorted to the end.

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1     2
## 2     5
## 3    NA
```

```
arrange(df, desc(x))
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1     5
## 2     2
## 3    NA
```

select()

The `select()` function can be used to select columns. This is especially useful when we don't want to include all variables in our analysis.

We can select columns by name:

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

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

or within a range:

```
select(flights, year:day)
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
```

```
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

We can also select all columns except some specific ones:

```
select(flights, -(year:day))
```

```
## # A tibble: 336,776 x 16
##    dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
##       <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
##  1      517            515         2      830            819        11 UA
##  2      533            529         4      850            830        20 UA
##  3      542            540         2      923            850        33 AA
##  4      544            545        -1     1004           1022       -18 B6
##  5      554            600        -6      812            837       -25 DL
##  6      554            558        -4      740            728        12 UA
##  7      555            600        -5      913            854        19 B6
##  8      557            600        -3      709            723       -14 EV
##  9      557            600        -3      838            846        -8 B6
## 10      558            600        -2      753            745         8 AA
## # i 336,766 more rows
## # i 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Here are some useful helper functions to use with `select()`:

- `starts_with("abc")`
- `ends_with("xyz")`
- `contains("ijk")`
- `matches("(.)\\1")` - selects variables based on a regular expression, this regular expression matches variables that contain repeated characters
- `num_range("x", 1:3)`- matches `x1`, `x2`, and `x3`

mutate()

It is often really useful to add new columns to the data that are functions of existing columns and the `mutate()` function can be used for this. `mutate()` always adds new columns to the end of the data frame.

```
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)

mutate(flights_sml,
  gain = dep_delay - arr_delay,
```

```
  speed = distance / air_time * 60
)
```

```
## # A tibble: 336,776 x 9
##     year month   day dep_delay arr_delay distance air_time  gain speed
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227    -9  370.
##  2  2013     1     1         4        20     1416      227   -16  374.
##  3  2013     1     1         2        33     1089      160   -31  408.
##  4  2013     1     1        -1       -18     1576      183    17  517.
##  5  2013     1     1        -6       -25      762      116    19  394.
##  6  2013     1     1        -4        12      719      150   -16  288.
##  7  2013     1     1        -5        19     1065      158   -24  404.
##  8  2013     1     1        -3       -14      229       53    11  259.
##  9  2013     1     1        -3        -8      944      140     5  405.
## 10  2013     1     1        -2         8      733      138   -10  319.
## # i 336,766 more rows
```

If you only want to keep the new variables, use `transmute()`:

```
transmute(flights,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
## # A tibble: 336,776 x 3
##     gain hours gain_per_hour
##    <dbl> <dbl>         <dbl>
##  1    -9 3.78          -2.38
##  2   -16 3.78          -4.23
##  3   -31 2.67         -11.6
##  4    17 3.05           5.57
##  5    19 1.93           9.83
##  6   -16 2.5           -6.4
##  7   -24 2.63          -9.11
##  8    11 0.883         12.5
##  9     5 2.33           2.14
## 10   -10 2.3           -4.35
## # i 336,766 more rows
```

The main requirement for functions that can be used with `mutate()` is that the function is vectorized. This means that the function has to take a vector of values as input and return a vector with the same number of values as an output.

Some examples are: - arithmetic operators: `+`, `-`, `*`, `/`, `^` - logs: `log()`, `log2()`, `log10()` - cumulative and rolling aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()` and `cummean()`

```
x <- 1:10
cumsum(x)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

```
cummean(x)
```

```
##  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

- logical comparisons

How would we add a column to the `flights` tibble that is the total delay?

```
flights %>% mutate(total_delay = dep_delay + arr_delay)
```

```
## # A tibble: 336,776 x 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
## # i 336,766 more rows
## # i 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 <dttm>, total_delay <dbl>
```

summarise()

The `summarise()` function collapses a data frame into a single row.

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

It is more useful to pair `summarise()` with the `group_by()` function.

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
```

```
##     year month   day delay
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5
##  2  2013     1     2 13.9
##  3  2013     1     3 11.0
##  4  2013     1     4  8.95
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
##  9  2013     1     9  2.28
## 10  2013     1    10  2.84
## # i 355 more rows
```

We can use pipes to chain together multiple functions. What does the following code do step by step?

```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

```
delays <- flights %>%
  group_by(dest) %>%  # group by destination
  summarise(
    count = n(),  # count number of times
    dist = mean(distance, na.rm = TRUE), # calculate mean distance
    delay = mean(arr_delay, na.rm = TRUE) # calculate mean delay
  ) %>%
  filter(count > 20, dest != "HNL") # keep those with count > 20 and destination not HNL
```

The `na.rm = TRUE` argument drops all rows with `NA` values from the computation when we use `group_by()`. `count()`, `mean()` and `median()` are some useful arguments to the `group_by()` function.

```
flights %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5
##  2  2013     1     2 13.9
##  3  2013     1     3 11.0
##  4  2013     1     4  8.95
```

```
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
##  9  2013     1     9  2.28
## 10  2013     1    10  2.84
## # i 355 more rows
```

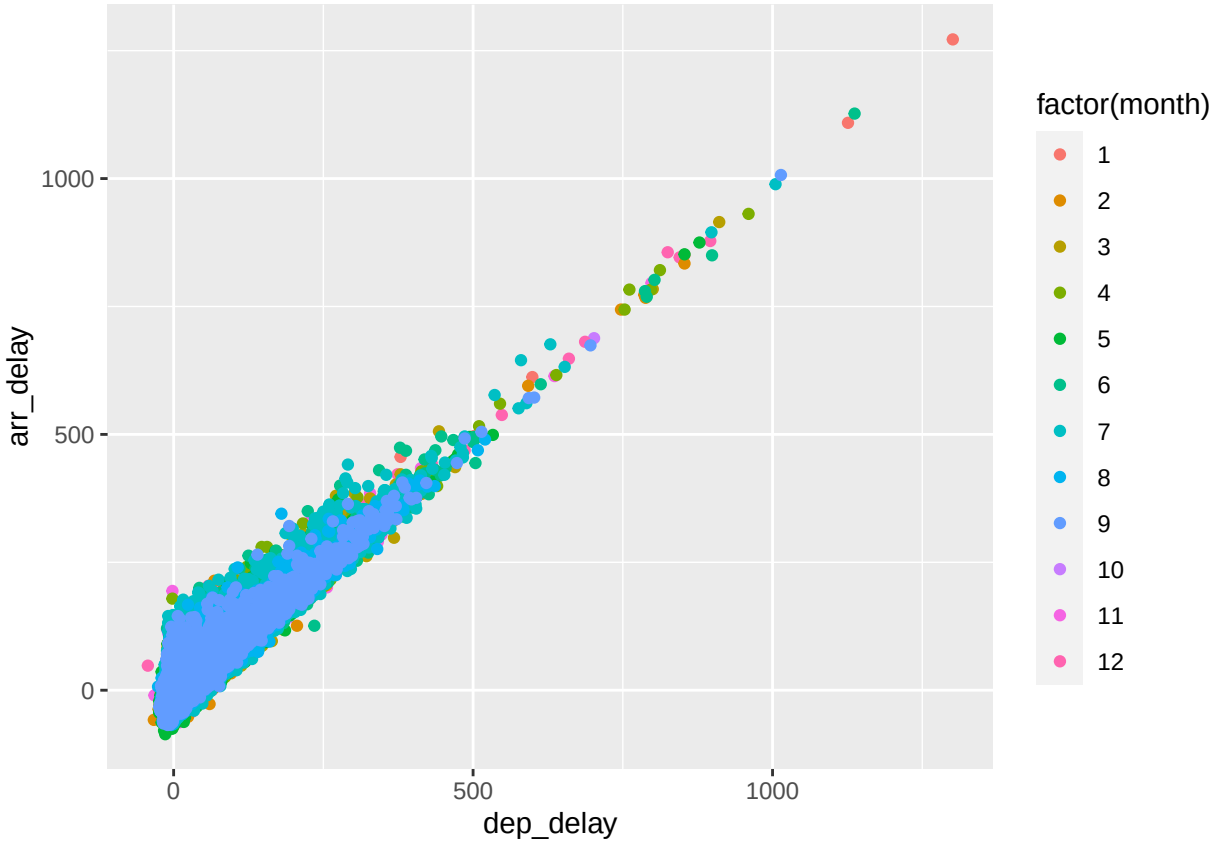
3. Visualization with ggplot

Check out Ch.3 of https://r4ds.had.co.nz/data-visualisation.html for some great discussion about `ggplot` and the grammar of graphics. For this part of the lab we will build on our data subsetting skills to create some nice plots using `ggplot`.

We'll walk through a few example plots:

```
ggplot(flights, aes(x = dep_delay, y = arr_delay, col = factor(month))) +
  geom_point()
```

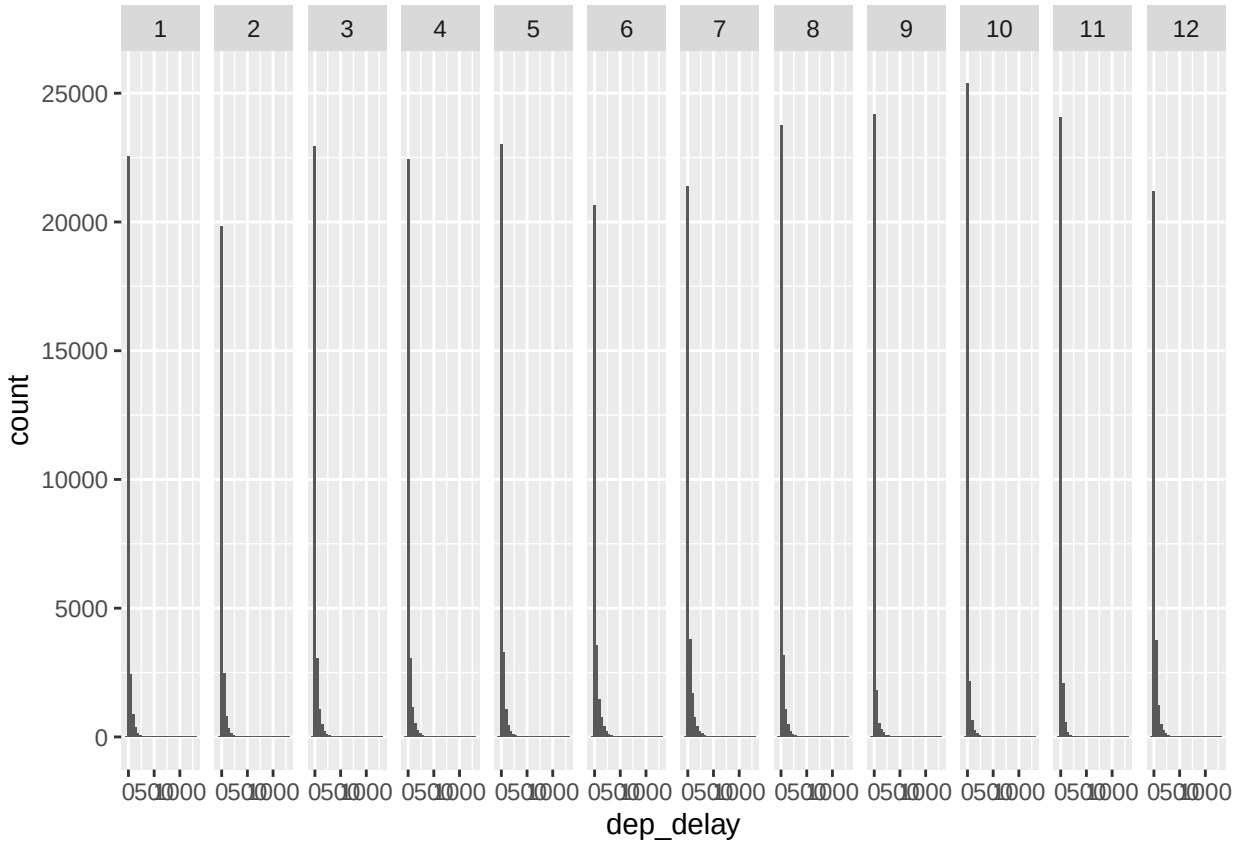
```
## Warning: Removed 9430 rows containing missing values (`geom_point()`).
```




```
ggplot(flights, aes(x = dep_delay)) +
  geom_histogram() +
  facet_grid(~factor(month))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 8255 rows containing non-finite values (`stat_bin()`).
```

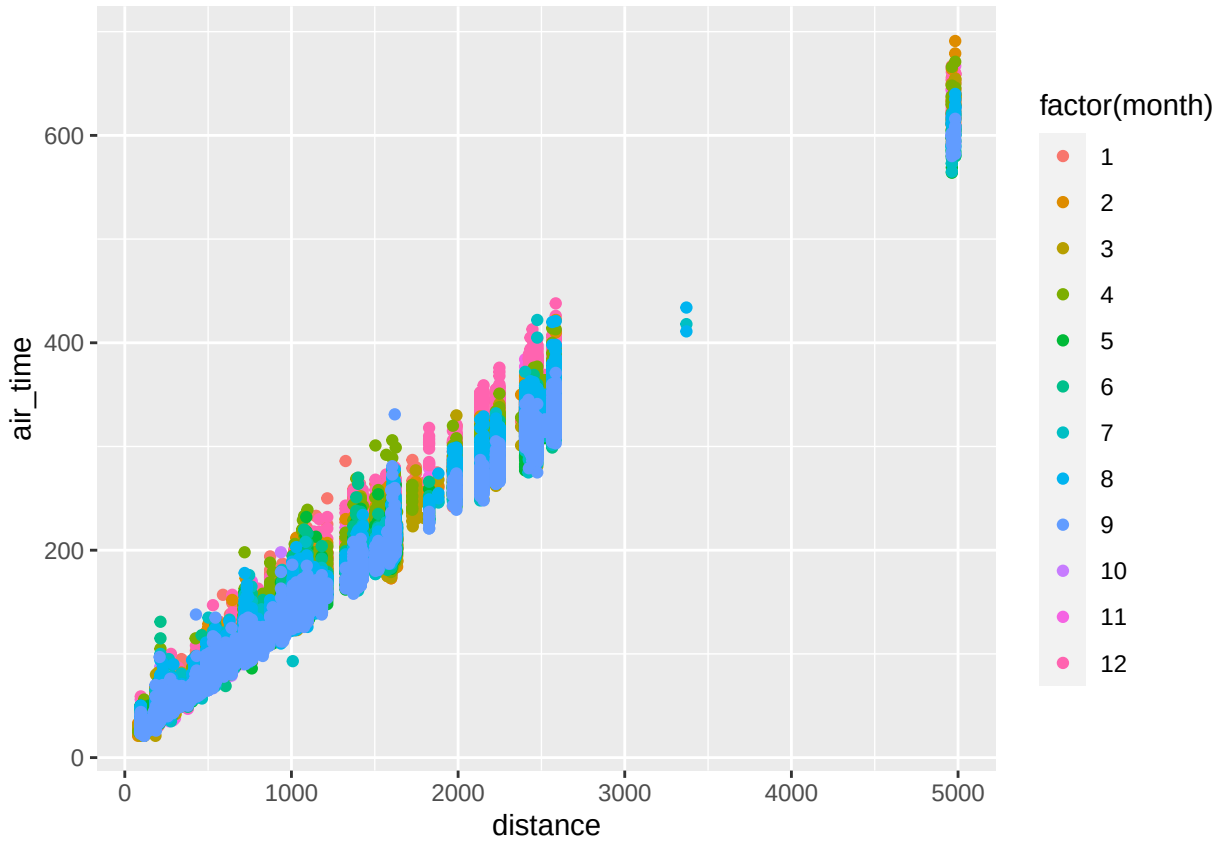


1. Select all of the flights that occur within the first 10 days of each month. Then, create a scatter plot using two of the numeric/continuous variables in the `flights` data set. Color your plot points by the day of the month.

```
first10d <- flights |>
  filter(day <= 10)

ggplot(first10d,
       aes(x = distance,
           y = air_time,
           col = factor(month))) +
  geom_point()
```

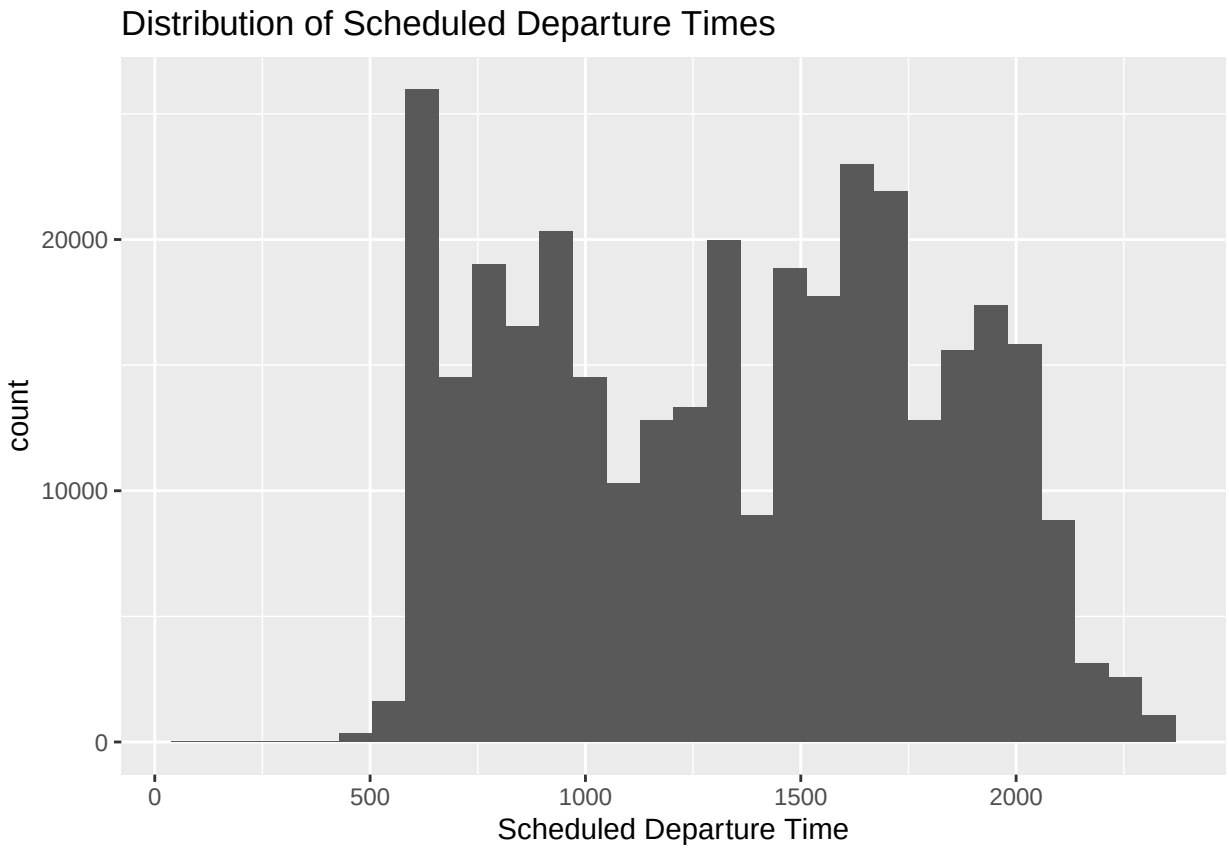
```
## Warning: Removed 3982 rows containing missing values (`geom_point()`).
```



2. Make a histogram of the scheduled departure time for the `flights` data. Change the label for the x-axis to be "Scheduled Departure Time" (hint look at `?labs()`). Add a title to your plot.

```
ggplot(flights,
       aes(x = sched_dep_time)) +
  geom_histogram() +
  labs(x = "Scheduled Departure Time",
    title = "Distribution of Scheduled Departure Times")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



3. Model Diagnostics

Randomly subset the `flights` data to only include 5000 rows.

```
set.seed(1)
flights_sample <- flights |>
  sample_n(5000)

head(flights_sample)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1    29      715            715         0     1012           1027
## 2  2013     2    15     2045           2048        -3     2345           2356
## 3  2013     9    25     1432           1417        15     1659           1646
## 4  2013     3     7     2138           2135         3      106             50
## 5  2013     1    29       NA           1900        NA       NA           2015
## 6  2013     8    16     1242           1241         1     1550           1518
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Using the `lm()` function, build a regression model trying to predict the `arr_delay` for the `flights` data. Use the following variables as predictors: `month`, `day`, `dep_time`, `arr_time`, `air_time` and `distance`.

```r
lm <- lm(arr_delay ~ month + day + dep_time + arr_time + air_time + distance, data = flights)

summary(lm)
```

```
##
## Call:
## lm(formula = arr_delay ~ month + day + dep_time + arr_time +
##     air_time + distance, data = flights)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -109.67  -21.61   -9.05    7.76 1300.87
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -2.117e+01  3.367e-01  -62.889   <2e-16 ***
## month       -2.078e-02  2.140e-02   -0.971    0.332
## day          4.196e-04  8.300e-03    0.051    0.960
## dep_time     3.625e-02  1.998e-04  181.418   <2e-16 ***
## arr_time    -2.074e-02  1.836e-04 -112.956   <2e-16 ***
## air_time     7.190e-01  5.730e-03  125.479   <2e-16 ***
## distance    -9.338e-02  7.290e-04 -128.094   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 41.68 on 327339 degrees of freedom
##   (9430 observations deleted due to missingness)
## Multiple R-squared:  0.1281, Adjusted R-squared:  0.1281
## F-statistic:  8015 on 6 and 327339 DF,  p-value: < 2.2e-16
```

Without running any additional code, how does the model fit look? Are there any variables that we can drop from the model?

The model has a low R-squared value, indicating a poor fit to the data.

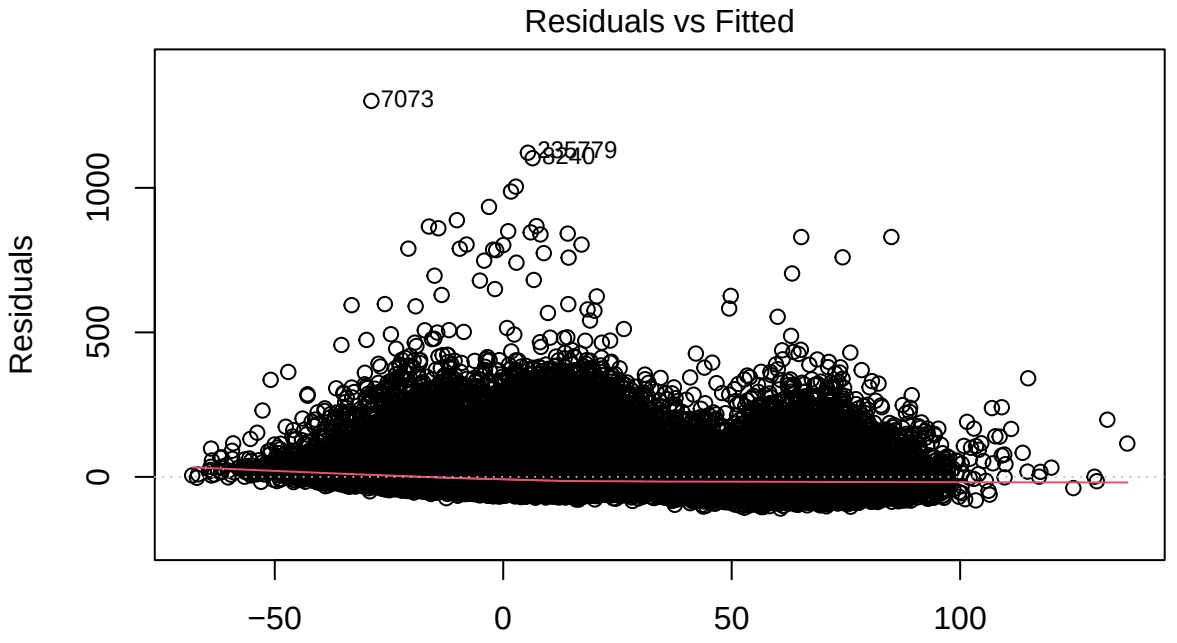
The variables month and day are not statistically significant and can be considered for removal from the model to simplify it without losing much explanatory power.

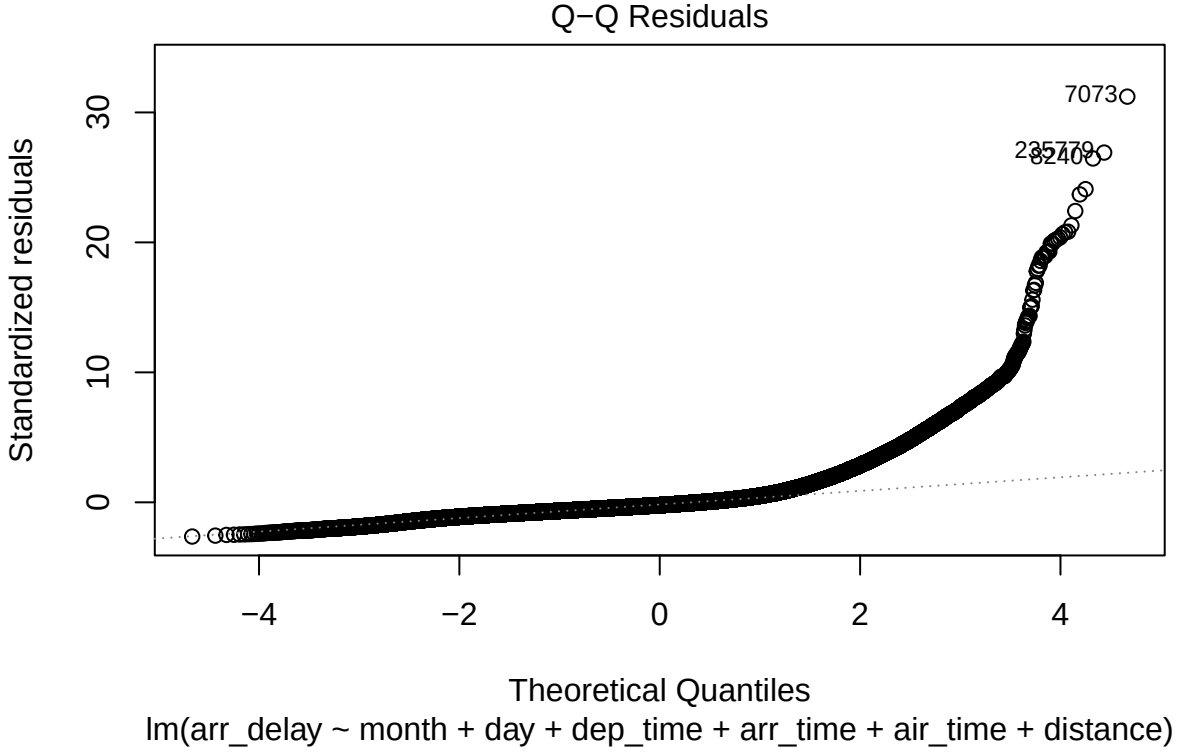
Despite the low R-squared value, the model is statistically significant, suggesting that it does have some predictive power.

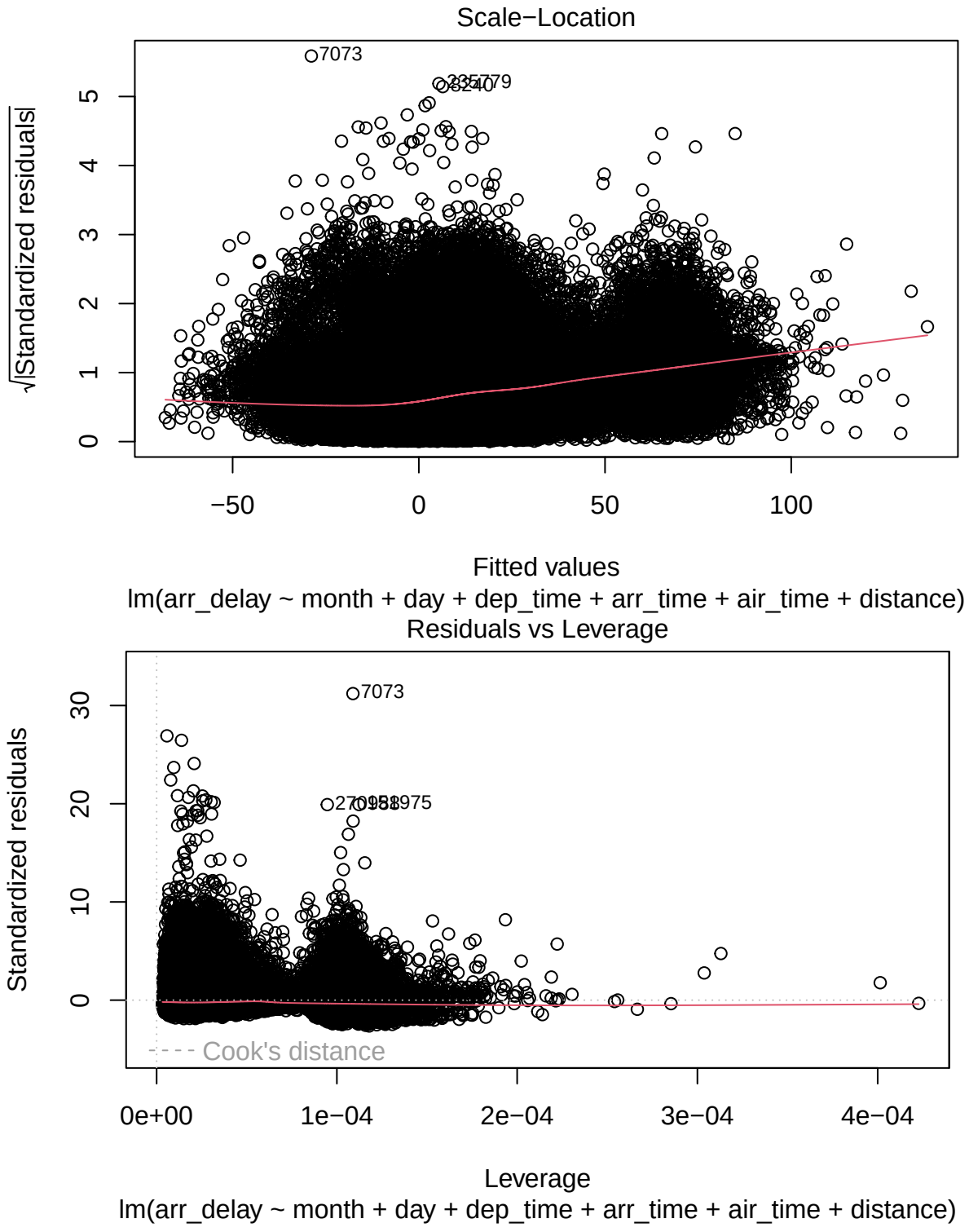
Further analysis and potentially adding other relevant variables could help in improving the model fit.

Let's look at some model diagnostics, using the `plot()` function on our `lm` object. What conclusions should we draw from the various diagnostic plots?

```
plot(lm)
```


Residuals vs Fitted
7073
235779
82401
Residuals
1000
500
0
−50
0
50
100
Fitted values
lm(arr_delay ~ month + day + dep_time + arr_time + air_time + distance)


Q−Q Residuals
7073
235779
82401
Standardized residuals
30
20
10
0
−4
−2
0
2
4
Theoretical Quantiles
lm(arr_delay ~ month + day + dep_time + arr_time + air_time + distance)



In the Residuals vs Fitted graph, the residuals doesn't seem to be randomly scattered, and several especially high values are highlighted by R.

In the Q-Q Residuals graph, for quantiles > 1, the standardized residuals deviate largely from the y = x line, indicating violation of normality.

In the Scale-Location graph, similar to the Residuals vs Fitted graph, the residuals doesn't seem to be randomly scattered, and several especially high values are highlighted by R.

In the Residuals vs Leverage graph, several points with high residual are highlighted as influential by R.

What is your conclusion about this regression model and why? How might we improve our model fit?

The current regression model has a low R-squared value of 0.1281, indicating it only explains about 12.81% of the variance in the arrival delay. This suggests the model is not fitting the data well. Moreover, the month and day variables are not statistically significant predictors, as indicated by their high p-values.

To improve the model, we can consider the following steps:

Removing insignificant variables: Start by removing the month and day variables to simplify the model.

Adding new variables or interaction terms: Incorporate other potentially relevant variables or interaction terms to capture more complexity.

Variable transformation: Experiment with different transformations of the existing variables to address potential non-linearity.