Solution PDF: Explore and analyze data with R

Eric Wanjau

Flights Data Exploration Challenge

A significant part of a a data scientist's role is to explore, analyze, and visualize data. In this challenge, you'll explore a real-world dataset containing flights data from the US Department of Transportation.

Let's start by loading the required packages.

```
# Load the packages in the tidyverse into the current R session
suppressPackageStartupMessages({
  library(tidyverse)
  library(summarytools)
  library(glue)
  library(patchwork)
  })
```

Now, we can import the into R and start doing some data science on it!

```
# Load and view the data
df_flights <- read_csv("https://raw.githubusercontent.com/MicrosoftDocs/ml-basics/master/challenges/data

df_flights %>%
  slice_head(n = 7)
```

```
## # A tibble: 7 x 20
##    Year Month DayofMonth DayOfWeek Carrier OriginAirportID OriginAirportName
##   <dbl> <dbl>      <dbl>     <dbl> <chr>             <dbl> <chr>
## 1  2013     9         16         1 DL                15304 Tampa International
## 2  2013     9         23         1 WN                14122 Pittsburgh Internati~
## 3  2013     9          7         6 AS                14747 Seattle/Tacoma Inter~
## 4  2013     7         22         1 OO                13930 Chicago O'Hare Inter~
## 5  2013     5         16         4 DL                13931 Norfolk International
## 6  2013     7         28         7 UA                12478 John F. Kennedy Inte~
## 7  2013    10          6         7 WN                13796 Metropolitan Oakland~
## # ... with 13 more variables: OriginCity <chr>, OriginState <chr>,
## #   DestAirportID <dbl>, DestAirportName <chr>, DestCity <chr>,
## #   DestState <chr>, CRSDepTime <dbl>, DepDelay <dbl>, DepDel15 <dbl>,
## #   CRSArrTime <dbl>, ArrDelay <dbl>, ArrDel15 <dbl>, Cancelled <dbl>
```

The dataset contains observations of US domestic flights in 2013, and consists of the following fields:

- **Year**: The year of the flight (all records are from 2013)
- **Month**: The month of the flight

- **DayofMonth**: The day of the month on which the flight departed
- **DayOfWeek**: The day of the week on which the flight departed - from 1 (Monday) to 7 (Sunday)
- **Carrier**: The two-letter abbreviation for the airline.
- **OriginAirportID**: A unique numeric identifier for the departure aiport
- **OriginAirportName**: The full name of the departure airport
- **OriginCity**: The departure airport city
- **OriginState**: The departure airport state
- **DestAirportID**: A unique numeric identifier for the destination aiport
- **DestAirportName**: The full name of the destination airport
- **DestCity**: The destination airport city
- **DestState**: The destination airport state
- **CRSDepTime**: The scheduled departure time
- **DepDelay**: The number of minutes departure was delayed (flight that left ahead of schedule have a negative value)
- **DelDelay15**: A binary indicator that departure was delayed by more than 15 minutes (and therefore considered "late")
- **CRSArrTime**: The scheduled arrival time
- **ArrDelay**: The number of minutes arrival was delayed (flight that arrived ahead of schedule have a negative value)
- **ArrDelay15**: A binary indicator that arrival was delayed by more than 15 minutes (and therefore considered "late")
- **Cancelled**: A binary indicator that the flight was cancelled

Your challenge is to explore the flight data to analyze possible factors that affect delays in departure or arrival of a flight.

1. Start by cleaning the data.
   - Identify any null or missing data, and impute appropriate replacement values.
   - Identify and eliminate any outliers in the **DepDelay** and **ArrDelay** columns.
2. Explore the cleaned data.
   - View summary statistics for the numeric fields in the dataset.
   - Determine the distribution of the **DepDelay** and **ArrDelay** columns.
   - Use statistics, aggregate functions, and visualizations to answer the following questions:
     - *What are the average (mean) departure and arrival delays?*
     - *How do the carriers compare in terms of arrival delay performance?*
     - *Is there a noticable difference in arrival delays for different days of the week?*
     - *Which departure airport has the highest average departure delay?*
     - *Do **late** departures tend to result in longer arrival delays than on-time departures?*

   - *Which route (from origin airport to destination airport) has the most **late** arrivals?*
   - *Which route has the highest average arrival delay?*

Sometimes, when we have a lot of columns in our data, it may difficult to get a grip of the data at first sight using `slice_head`

`glimpse` produces a transposed version where columns run down the page, and data runs across. This makes it possible to see every column in a data frame. Into the bargain, it also shows the dimension of the tibble and underlying data types of the columns.

```
# Get a glimpse of your data
df_flights %>%
  glimpse()
```

```
## Rows: 271,940
## Columns: 20
## $ Year              <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013~
## $ Month             <dbl> 9, 9, 9, 7, 5, 7, 10, 7, 10, 5, 6, 7, 8, 7, 10, 4, 1~
## $ DayofMonth        <dbl> 16, 23, 7, 22, 16, 28, 6, 28, 8, 12, 9, 21, 4, 17, 2~
## $ DayOfWeek         <dbl> 1, 1, 6, 1, 4, 7, 7, 7, 2, 7, 7, 7, 7, 3, 7, 7, 4, 5~
## $ Carrier           <chr> "DL", "WN", "AS", "OO", "DL", "UA", "WN", "EV", "AA"~
## $ OriginAirportID   <dbl> 15304, 14122, 14747, 13930, 13931, 12478, 13796, 122~
## $ OriginAirportName <chr> "Tampa International", "Pittsburgh International", "~
## $ OriginCity        <chr> "Tampa", "Pittsburgh", "Seattle", "Chicago", "Norfol~
## $ OriginState       <chr> "FL", "PA", "WA", "IL", "VA", "NY", "CA", "DC", "IL"~
## $ DestAirportID     <dbl> 12478, 13232, 11278, 11042, 10397, 14771, 12191, 145~
## $ DestAirportName   <chr> "John F. Kennedy International", "Chicago Midway Int~
## $ DestCity          <chr> "New York", "Chicago", "Washington", "Cleveland", "A~
## $ DestState         <chr> "NY", "IL", "DC", "OH", "GA", "CA", "TX", "VA", "TX"~
## $ CRSDepTime        <dbl> 1539, 710, 810, 804, 545, 1710, 630, 2218, 1010, 175~
## $ DepDelay          <dbl> 4, 3, -3, 35, -1, 87, -1, 4, 8, 40, 3, 10, 1, 95, -1~
## $ DepDel15          <dbl> 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0~
## $ CRSArrTime        <dbl> 1824, 740, 1614, 1027, 728, 2035, 1210, 2301, 1240, ~
## $ ArrDelay          <dbl> 13, 22, -7, 33, -9, 183, -3, 15, -10, 10, -8, -4, -4~
## $ ArrDel15          <dbl> 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0~
## $ Cancelled         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

Clean missing values

Once you have imported your data, it is always a good idea to clean it. Sadly, this is often chronically underestimated, yet it's a fundamental step required for the subsequent operations in data analysis.

Let's find how many null values there are for each column.

```
# Find how many null values there are for each column.
colSums(is.na(df_flights))
```

```
##              Year             Month        DayofMonth         DayOfWeek
##                 0                 0                 0                 0
##           Carrier   OriginAirportID OriginAirportName        OriginCity
##                 0                 0                 0                 0
##       OriginState     DestAirportID   DestAirportName          DestCity
##                 0                 0                 0                 0
```

```
##         DestState        CRSDepTime          DepDelay          DepDel15
##                 0                 0                 0              2761
##        CRSArrTime          ArrDelay          ArrDel15         Cancelled
##                 0                 0                 0                 0
```

Hmm, looks like there are some NA (missing values) `late departure` indicators. Departures are considered late if the delay is 15 minutes or more, so let's see the delays for the ones with an NA late indicator:

Question 1.

Starting with `df_flights`, **select** columns `DepDelay` and `DepDel15` then **filter** to obtain rows where the value of `DepDel15` is `NA`. Assign the results in a variable name `flights_depdel`.

```
BEGIN QUESTION
name: Question 1
manual: false
```

```
# Select columns DepDelay and DepDel15
# and then Filter the tibble to obtain observations where there is a missing value for DepDel15

flights_depdel <- df_flights %>%
  select(DepDelay, DepDel15) %>%
  filter(is.na(DepDel15))
```

```
. = " # BEGIN TEST CONFIG

success_message: Excellent. You have successfully selected columns **DepDelay** and **DepDel15** and the

failure_message: Oops! Let's give it another try. Ensure you have selected columns **DepDelay** and **De
" # END TEST CONFIG


suppressPackageStartupMessages({
  library(testthat)
  library(ottr)
})

## Test ##
test_that('the first column has no NA while the second has 2761 NAs', {
    expect_equal(sum(is.na(flights_depdel$DepDelay)), 0)
    expect_equal(sum(is.na(flights_depdel$DepDel15)), 2761)
})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

success_message: Fantastic. Your tibble dimensions are also correct.

failure_message: Almost there! Ensure you have selected columns **DepDelay** and **DepDel15** and then i
" # END TEST CONFIG
```

```r
suppressPackageStartupMessages({
  library(testthat)
  library(ottr)
})

## Test ##
test_that('data dimensions correct', {
  expect_output(glimpse(flights_depdel), "Rows: 2,761\nColumns: 2")

})
```

```
## Test passed
```

Good job! Now, let's `glimpse` at `flights_depdel`.

```r
flights_depdel %>%
  glimpse()
```

```
## Rows: 2,761
## Columns: 2
## $ DepDelay <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ DepDel15 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

From the first few observations, it looks like the observations in `DepDel15` (A binary indicator that departure was delayed by more than 15 minutes) all have a corresponding delay of 0 in `DepDelay`(The number of minutes departure was delayed). Let's check by looking at the summary statistics for the `DepDelay` records:

```r
# Get summary statistics using summary function
df_flights %>%
  filter(rowSums(is.na(.)) > 0) %>%
  select(DepDelay) %>%
  summary()
```

```
##     DepDelay
##  Min.   :0
##  1st Qu.:0
##  Median :0
##  Mean   :0
##  3rd Qu.:0
##  Max.   :0
```

The min, max, and mean are all 0; so it seems that none of these were actually *late* departures.

Question 2.

Starting with `df_flights`, replace the missing values in the column **DepDel15** with a `0`. Assign this to a variable name `df_flights`.

```
BEGIN QUESTION
name: Question 2
manual: false
```

```r
# Replace missing values in DepDel15 with 0
df_flights <- df_flights %>%
  mutate(DepDel15 = replace_na(DepDel15, 0))
```

```r
. = " # BEGIN TEST CONFIG

success_message: Good job! No more missing values in column DepDel15.

failure_message: Almost there! Ensure you have replaced the missing values in the column DepDel15 with a
" # END TEST CONFIG


## Test ##
test_that('data has no missing values', {
  expect_false(anyNA(df_flights), FALSE)

})
```

```
## Test passed
```

```r
. = " # BEGIN TEST CONFIG

success_message: Fantastic. Your tibble dimensions are also correct.

failure_message: Almost there! Ensure you are starting with tibble **df_flights** then replaced the miss
" # END TEST CONFIG


## Test ##
test_that('data dimensions correct', {
  expect_output(glimpse(df_flights), "Rows: 271,940")
  expect_output(glimpse(df_flights), "Columns: 20")

})
```

```
## Test passed
```

Good job! No missing values now. Let's take this a little further.

Clean outliers

An outlier is a data point that differs significantly from other observations. Let's create a function that shows the distribution and summary statistics for a given column.

```r
# Function to show summary stats and distribution for a column
show_distribution <- function(var_data, binwidth) {

  # Get summary statistics by first extracting values from the column
  min_val <- min(pull(var_data))
  max_val <- max(pull(var_data))
  mean_val <- mean(pull(var_data))
```

```r
  med_val <- median(pull(var_data))
  mod_val <- statip::mfv(pull(var_data))

  # Print the stats
  stats <- glue::glue(
  'Minimum: {format(round(min_val, 2), nsmall = 2)}
   Mean: {format(round(mean_val, 2), nsmall = 2)}
   Median: {format(round(med_val, 2), nsmall = 2)}
   Mode: {format(round(mod_val, 2), nsmall = 2)}
   Maximum: {format(round(max_val, 2), nsmall = 2)}'
  )

  theme_set(theme_light())
  # Plot the histogram
  hist_gram <- ggplot(var_data) +
  geom_histogram(aes(x = pull(var_data)), binwidth = binwidth,
                fill = "midnightblue", alpha = 0.7, boundary = 0.4) +

  # Add lines for the statistics
  geom_vline(xintercept = min_val, color = 'gray33', linetype = "dashed", size = 1.3) +
  geom_vline(xintercept = mean_val, color = 'cyan', linetype = "dashed", size = 1.3) +
  geom_vline(xintercept = med_val, color = 'red', linetype = "dashed", size = 1.3 ) +
  geom_vline(xintercept = mod_val, color = 'yellow', linetype = "dashed", size = 1.3 ) +
  geom_vline(xintercept = max_val, color = 'gray33', linetype = "dashed", size = 1.3 ) +

  # Add titles and labels
  ggtitle('Data Distribution') +
  xlab('')+
  ylab('Frequency') +
  theme(plot.title = element_text(hjust = 0.5))

  # Plot the box plot
  bx_plt <- ggplot(data = var_data) +
  geom_boxplot(mapping = aes(x = pull(var_data), y = 1),
              fill = "#E69F00", color = "gray23", alpha = 0.7) +

    # Add titles and labels
  xlab("Value") +
  ylab("") +
  theme(plot.title = element_text(hjust = 0.5))


  # To return multiple outputs, use a `list`
  return(

    list(stats,
         hist_gram / bx_plt)) # End of returned outputs

} # End of function
```

Question 3. Starting with the `df_flights` data, only keep the `DepDelay` column. Assign this to a variable name `df_col`.

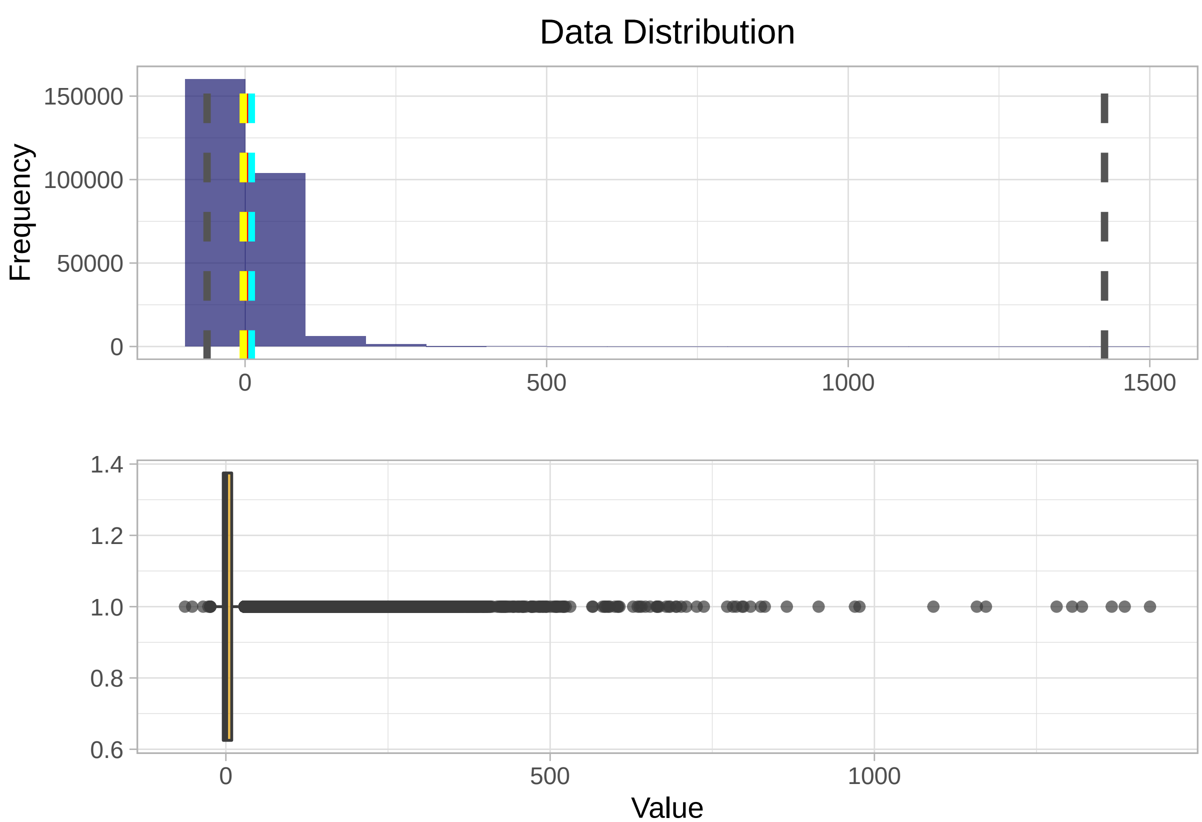
Once you have this figured out, call the function `show_distribution` with the arguments names and corresponding values as follows: `var_data = df_col` and `binwidth = 100`

From the function output, what's the distribution of **DepDelay** (The number of minutes departure was delayed)?

```
BEGIN QUESTION
name: Question 3
manual: false
```

```
# Select DepDelay column
df_col = df_flights %>%
  select(DepDelay)

# Call the function show_distribution
show_distribution(var_data = df_col, binwidth = 100)
```

```
## [[1]]
## Minimum: -63.00
## Mean: 10.35
## Median: -1.00
## Mode: -3.00
## Maximum: 1425.00
##
## [[2]]
```



```r
. = " # BEGIN TEST CONFIG

success_message: Fantastic! You have successfully selected column **DepDelay**

failure_message: Almost there! Ensure you have selected column **DepDelay**
" # END TEST CONFIG

## Test ##
test_that('df_col corresponds to DepDelay', {

  expect_equal(colnames(df_col), "DepDelay")
  expect_output(glimpse(df_col), "Rows: 271,940\nColumns: 1")

})
```

```
## Test passed
```

```r
. = " # BEGIN TEST CONFIG

success_message: Your summary statistics are also looking great!

failure_message: Almost there! Ensure that you selected column DepDelay to obtain the desired summary s
" # END TEST CONFIG

## Test ##

test_that('the distribution of DepDelay is correct', {

 expect_equal(show_distribution(var_data = df_col, binwidth = 100)[[1]], "Minimum: -63.00\nMean: 10.35\n

})
```

```
## Test passed
```

Now, let's investigate the distribution of **ArrDelay** (The number of minutes arrival was delayed)

Question 4. Starting with the `df_flights` data, only keep the **`ArrDelay`** column. Assign this to a variable name `df_col`.

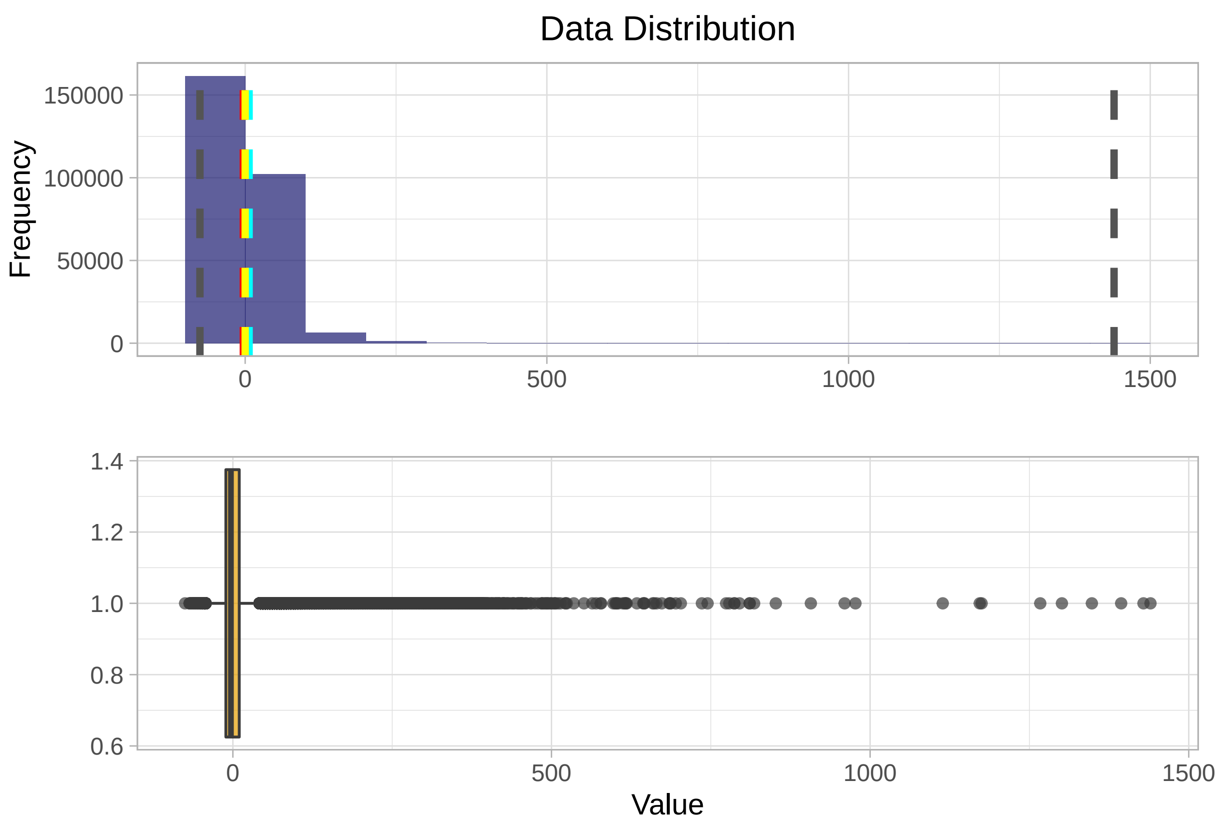
Once you have this figured out, call the function `show_distribution` with the arguments names and corresponding values as follows: `var_data = df_col` and `binwidth = 100`

From the function output, what's the distribution of **ArrDelay**?

```
BEGIN QUESTION
name: Question 4
manual: false
```

```r
# Select DepDelay column
df_col = df_flights %>%
  select(ArrDelay)

# Call the function show_distribution
show_distribution(var_data = df_col, binwidth = 100)
```

```
## [[1]]
## Minimum: -75.00
## Mean: 6.50
## Median: -3.00
## Mode: 0.00
## Maximum: 1440.00
##
## [[2]]
```




```
. = " # BEGIN TEST CONFIG

success_message: Fantastic! You have successfully selected column **ArrDelay**

failure_message: Almost there! Ensure you have selected column **ArrDelay**
" # END TEST CONFIG

## Test ##
test_that('df_col corresponds to ArrDelay', {

  expect_equal(colnames(df_col), "ArrDelay")
  expect_output(glimpse(df_col), "Rows: 271,940\nColumns: 1")

})
```

```
## Test passed
```

```r
. = " # BEGIN TEST CONFIG

success_message: Your summary statistics are also looking great!

failure_message: Almost there! Ensure that you selected column ArrDelay to obtain the desired summary s
" # END TEST CONFIG

## Test ##

test_that('the distribution of ArrDelay is correct', {

 expect_equal(show_distribution(var_data = df_col, binwidth = 100)[[1]], "Minimum: -75.00\nMean: 6.50\n

})
```

```
## Test passed
```

From both outputs, there are a outliers at the lower and upper ends of both variables. Let's trim the data so that we include only rows where the values for these fields are within the 1st and 90th percentile. We begin with the **ArrDelay** observation.

```r
# Trim outliers for ArrDelay based on 1% and 90% percentiles
# Produce quantiles corresponding to 1% and 90%
ArrDelay_01pcntile <- df_flights %>%
  pull(ArrDelay) %>%
  quantile(probs = 1/100, names = FALSE)

ArrDelay_90pcntile <- df_flights %>%
  pull(ArrDelay) %>%
  quantile(probs = 90/100, names = FALSE)

# Print 1st and 90th quantiles respectively
cat(ArrDelay_01pcntile, "\n", ArrDelay_90pcntile)
```

```
## -33
##  38
```

Now that we have quantiles corresponding to 1% and 90%, let's filter the `df_flights` data to only include rows whose Arrival delay falls within this range.

Question 5. Starting with the `df_flights` data, filter to only include rows whose **ArrDelay** falls within 1st and 90th quantiles. Assign this to a variable name `df_flights`.

```
BEGIN QUESTION
name: Question 5
manual: false
```

```r
# Filter data to remove outliers
df_flights <- df_flights %>%
  filter(ArrDelay > ArrDelay_01pcntile, ArrDelay < ArrDelay_90pcntile)
```

```
. = " # BEGIN TEST CONFIG

success_message: Well done! You have successfully filtered the data to include observations whose Arriva

failure_message: Almost there! Ensure you have filtered the **df_flights** data to only include rows whe
" # END TEST CONFIG

## Test ##
test_that('there are no outliers', {
  expect_equal(sum(df_flights$ArrDelay < ArrDelay_01pcntile), 0)
  expect_equal(sum(df_flights$ArrDelay > ArrDelay_90pcntile), 0)

})
```

```
## Test passed
```

Now, let's do the same for `DepDelay` column.

Question 6. Starting with the `df_flights` data, obtain quantiles corresponding to 1% and 90%. Assign these values to the variable names `DepDelay_01pcntile` and `DepDelay_90pcntile` respectively.

```
BEGIN QUESTION
name: Question 6
manual: false
```

```
# Trim outliers for DepDelay based on 1% and 90% percentiles
# Produce quantiles corresponding to 1% and 90%
DepDelay_01pcntile <- df_flights %>%
  pull(DepDelay) %>%
  quantile(probs = 1/100, names = FALSE)

DepDelay_90pcntile <- df_flights %>%
  pull(DepDelay) %>%
  quantile(probs = 90/100, names = FALSE)

# Print 1st and 90th quantiles respectively
cat(DepDelay_01pcntile, "\n", DepDelay_90pcntile)
```

```
## -12
##  17
```

```
. = " # BEGIN TEST CONFIG

success_message: That's it. You've got the correct values for the 1st and 90th percentiles.

failure_message: Let's give it another try! Ensure your **DepDelay** quantiles correspond to a probabil
" # END TEST CONFIG

## Test ##
test_that('quantile values are correct', {
  expect_equal(DepDelay_01pcntile, -12)
  expect_equal(DepDelay_90pcntile, 17)

})
```

```
## Test passed
```

Good job!!

Now that we have quantiles corresponding to 1% and 90%, let's filter the `df_flights` data to only include rows whose Departure delay falls within this range.

Question 7. Starting with the `df_flights` data, filter to only include rows whose **DepDelay** falls within 1st and 90th quantiles. Assign this to a variable name `df_flights`.

```
BEGIN QUESTION
name: Question 7
manual: false
```

```r
# Filter data to remove outliers
df_flights <- df_flights %>%
  filter(DepDelay > DepDelay_01pcntile, DepDelay < DepDelay_90pcntile)
```

```r
. = " # BEGIN TEST CONFIG

success_message: Well done! You have successfully filtered the data to include observations whose Depar

failure_message: Almost there! Ensure you have filtered the **df_flights** data to only include rows wh
" # END TEST CONFIG

## Test ##
test_that('there are no outliers', {
  expect_equal(sum(df_flights$DepDelay < DepDelay_01pcntile), 0)
  expect_equal(sum(df_flights$DepDelay > DepDelay_90pcntile), 0)

})
```

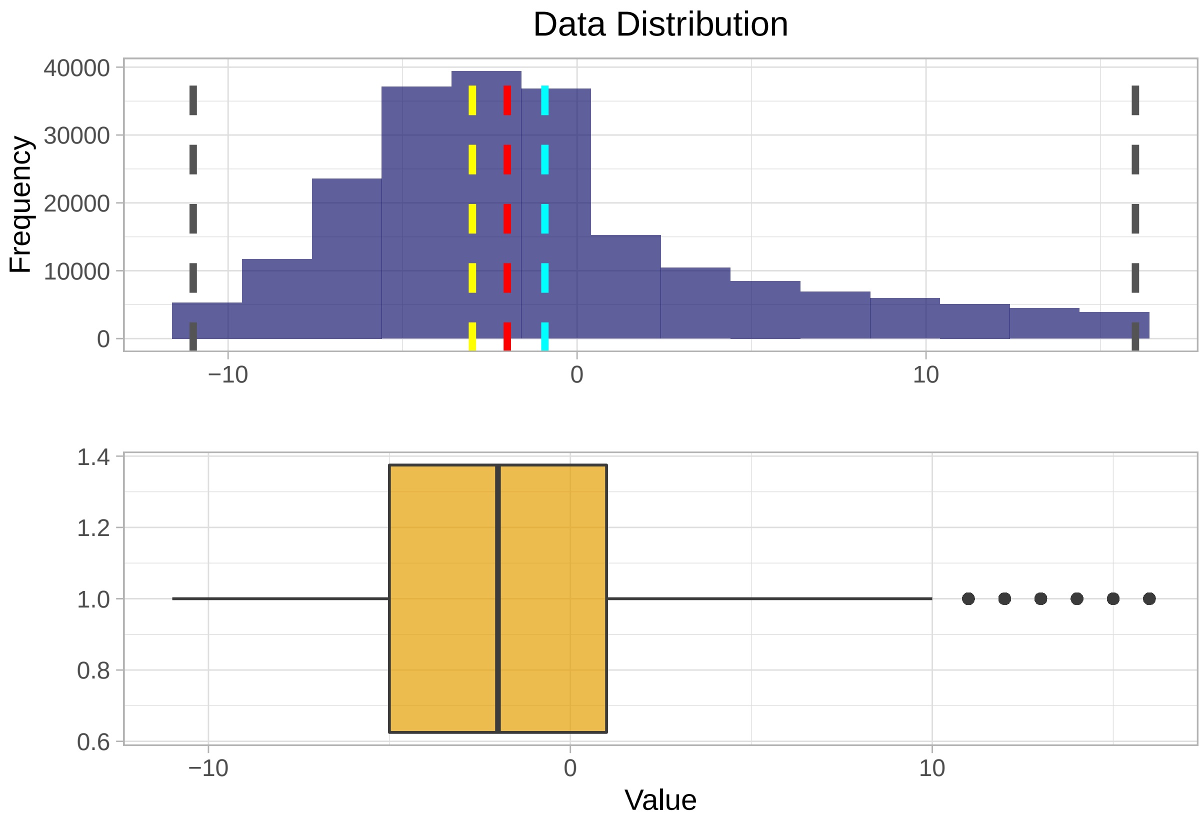
```
## Test passed
```

You rock!!

Now, we can check the distribution of our two variables with outliers removed.

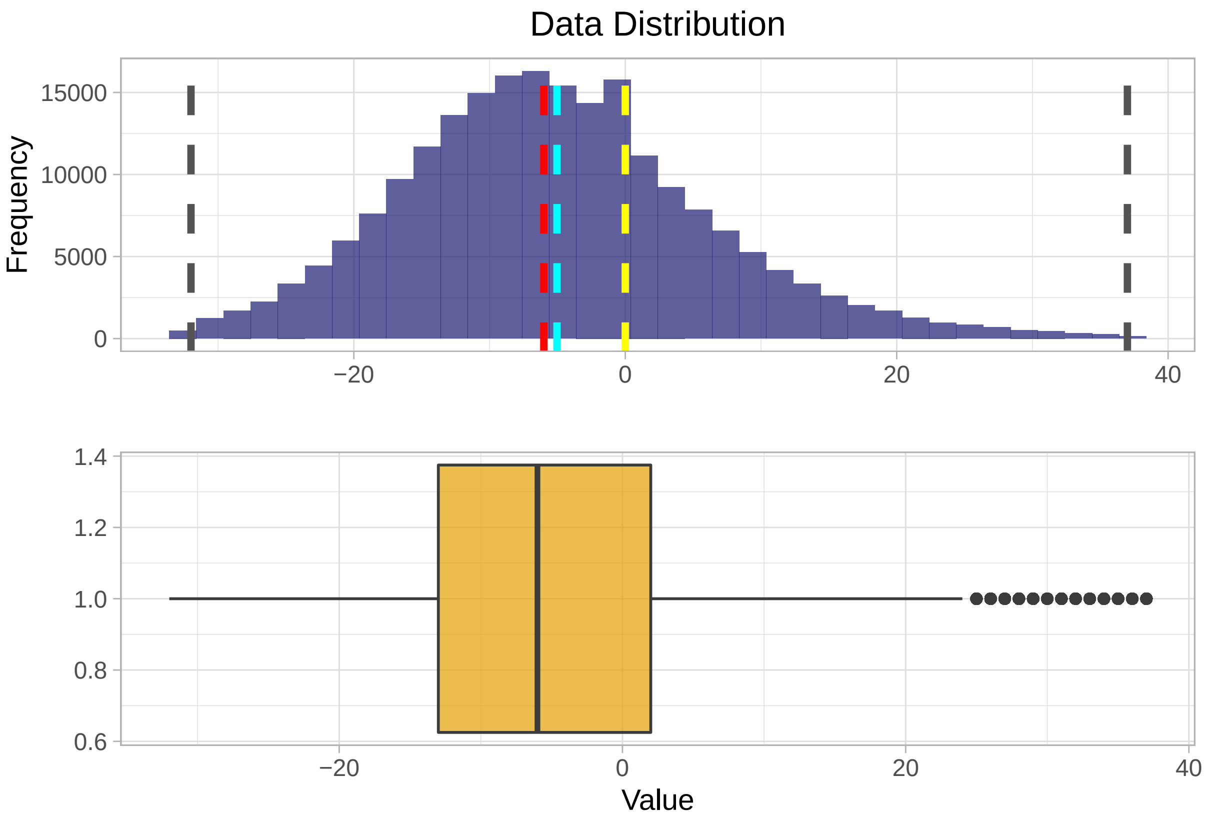
```r
# Distribution of DepDelay
show_distribution(var_data = select(df_flights, DepDelay), binwidth = 2)
```

```
## [[1]]
## Minimum: -11.00
## Mean: -0.92
## Median: -2.00
## Mode: -3.00
## Maximum: 16.00
##
## [[2]]
```




```r
# Distribution of ArrDelay
show_distribution(var_data = select(df_flights, ArrDelay), binwidth = 2)
```

```
## [[1]]
## Minimum: -32.00
## Mean: -5.03
## Median: -6.00
## Mode: 0.00
## Maximum: 37.00
##
## [[2]]
```



Much better!

Now that the data is all cleaned up, we can begin doing some exploratory analysis.

Explore the data

Let's start with an overall view of the summary statistics for the numeric columns.

```
# Obtain common summary statistics using summarytools package
df_flights %>%
  descr(stats = "common")
```

```
## Non-numerical variable(s) ignored: Carrier, OriginAirportName, OriginCity, OriginState, DestAirportNa
```

```
## Descriptive Statistics
## df_flights
## N: 214397
##
##                    ArrDel15    ArrDelay   Cancelled   CRSArrTime   CRSDepTime   DayofMonth
## ---------------- ----------- ----------- ----------- ------------ ------------ ------------
##             Mean        0.07       -5.03        0.01      1461.41      1278.22        15.79
##          Std.Dev        0.25       11.42        0.11       485.68       469.44         8.86
##              Min        0.00      -32.00        0.00         1.00         1.00         1.00
##           Median        0.00       -6.00        0.00      1445.00      1235.00        16.00
##              Max        1.00       37.00        1.00      2359.00      2359.00        31.00
```

```
##         N.Valid   214397.00   214397.00   214397.00    214397.00    214397.00    214397.00
##       Pct.Valid      100.00      100.00      100.00       100.00       100.00       100.00
##
## Table: Table continues below
##
##
##
##                   DayOfWeek    DepDel15    DepDelay   DestAirportID       Month   OriginAirportID
## --------------- ----------- ----------- ----------- --------------- ----------- -----------------
##            Mean        3.90        0.02       -0.92        12726.28        7.02          12757.83
##         Std.Dev        2.00        0.13        5.71         1506.25        2.01           1510.06
##             Min        1.00        0.00      -11.00        10140.00        4.00          10140.00
##          Median        4.00        0.00       -2.00        12892.00        7.00          12892.00
##             Max        7.00        1.00       16.00        15376.00       10.00          15376.00
##         N.Valid   214397.00   214397.00   214397.00       214397.00   214397.00         214397.00
##       Pct.Valid      100.00      100.00      100.00          100.00      100.00            100.00
##
## Table: Table continues below
##
##
##
##                        Year
## --------------- -----------
##            Mean     2013.00
##         Std.Dev        0.00
##             Min     2013.00
##          Median     2013.00
##             Max     2013.00
##         N.Valid   214397.00
##       Pct.Valid      100.00
```

What are the mean departure and arrival delays?

Question 8. Starting with the `df_flights` data, use `across()` within `summarize()` to find the mean across `**DepDelay**` and `**ArrDelay**` columns. Assign this to `df_delays` variable name. What are the mean delays?

```
BEGIN QUESTION
name: Question 8
manual: false
```

```r
# Summarise the departure and arrival delays by finding the mean
df_delays <- df_flights %>%
  summarise(across(contains("delay"), mean))

df_delays
```

```
## # A tibble: 1 x 2
##   DepDelay ArrDelay
##      <dbl>    <dbl>
## 1   -0.922    -5.03
```

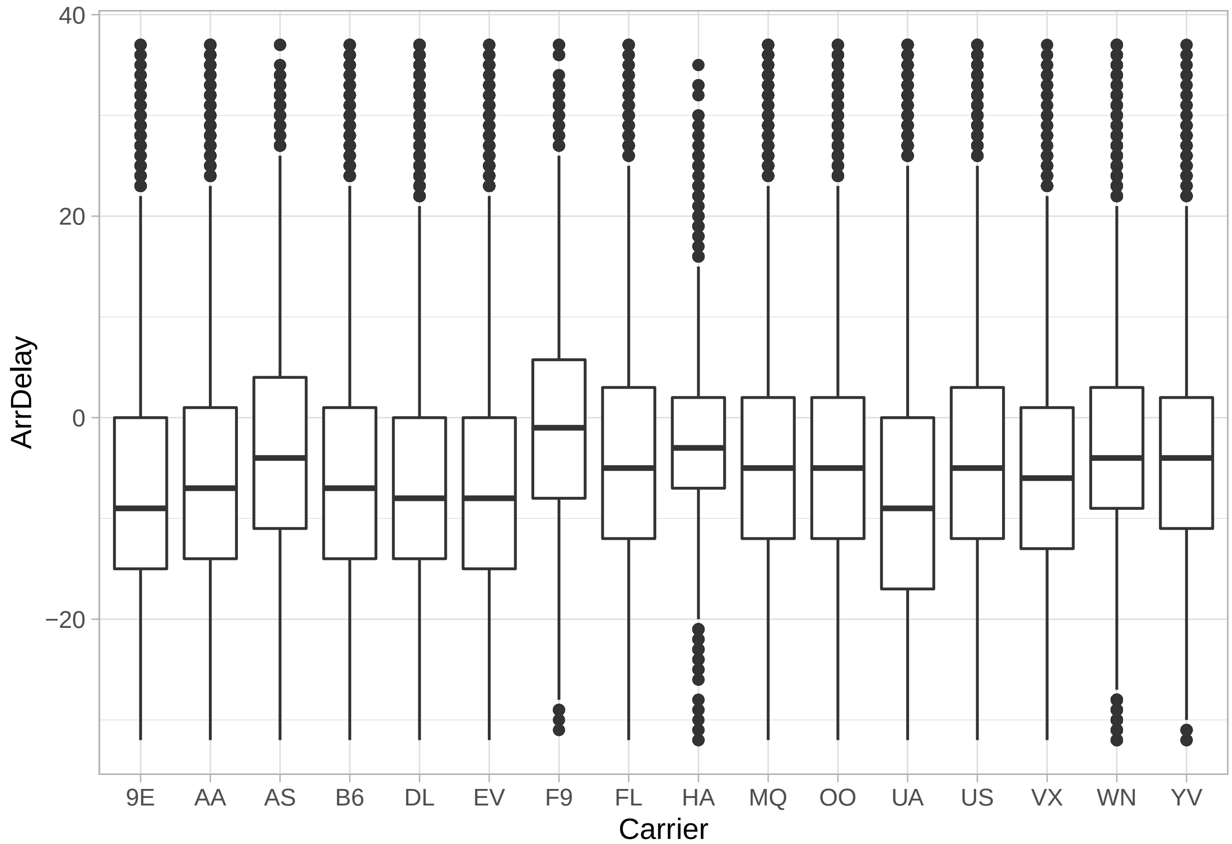
```r
. = " # BEGIN TEST CONFIG

success_message: Fantastic! You have successfully found the mean Delay time across **DepDelay** and **A

failure_message: Let's give it another shot! Ensure that starting with **df_flights** you are creating a

" # END TEST CONFIG

## Test ##
test_that('summary tibble has correct values', {
  expect_output(glimpse(df_delays), "Rows: 1\nColumns: 2", fixed = TRUE)
  expect_equal(df_delays$DepDelay, -0.921692)


})
```

```
## Test passed
```

How do the carriers compare in terms of arrival delay performance?

A box plot can be a good way for graphically depicting the distribution of groups of numerical data through their quantiles. The `geom` that takes care of box plots is `geom_boxplot`

```r
# Compare arrival delay across different carriers
df_flights %>%
  ggplot() +
  geom_boxplot(mapping = aes(x = Carrier, y = ArrDelay))
```

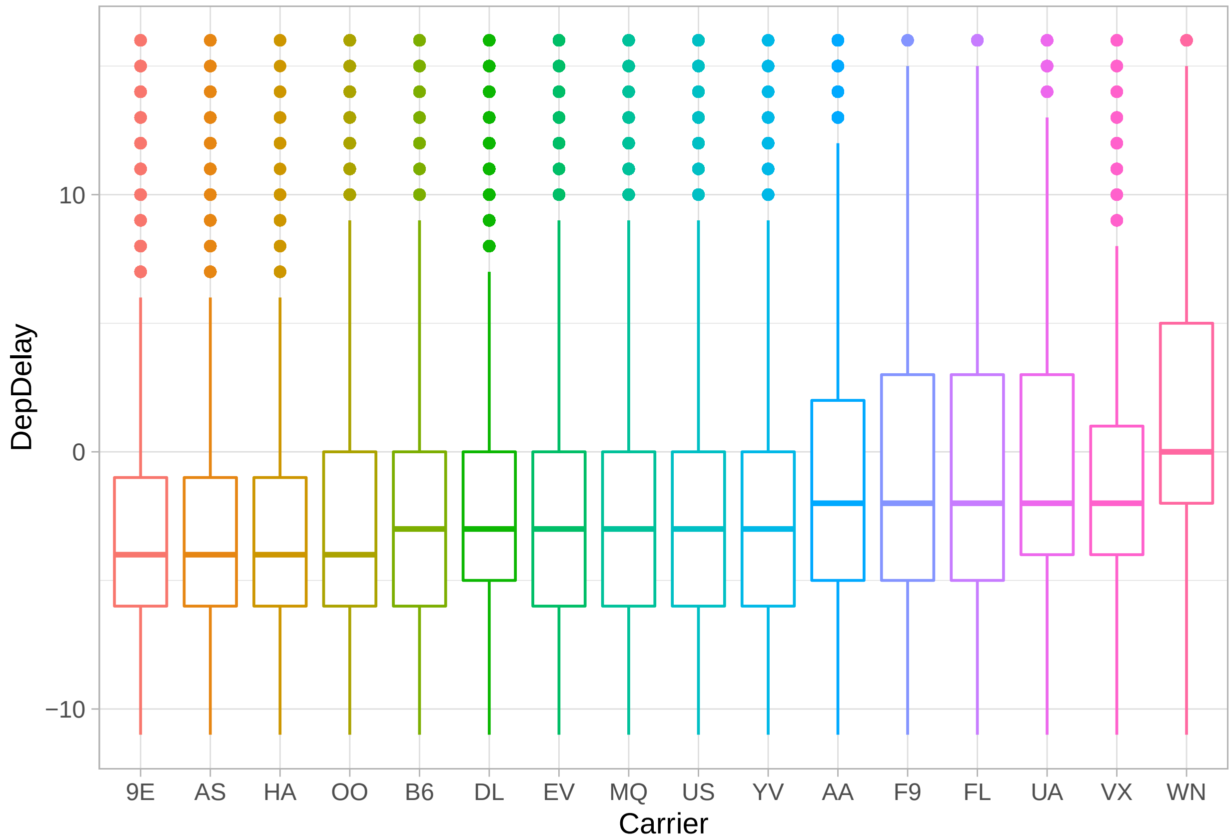


How do the carriers compare in terms of departure delay performance?

Let's do the same for the departure delay performance.

We can also try and rearrange the `Carrier` levels in ascending order of the delay time and sprinkle some color to the plots too.

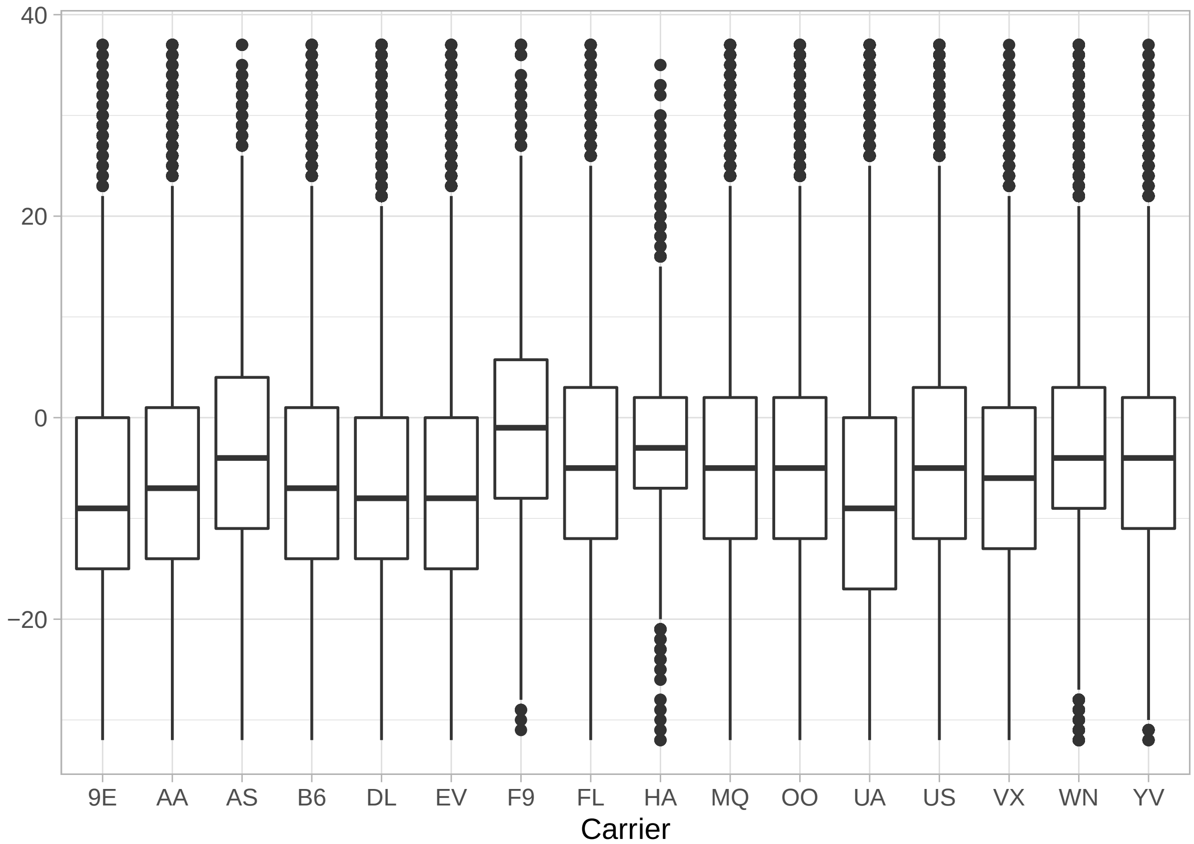
```
df_flights %>%
  mutate(Carrier = fct_reorder(Carrier, DepDelay)) %>%
  ggplot() +
  geom_boxplot(mapping = aes(x = Carrier, y = DepDelay, color = Carrier), show.legend = FALSE)
```



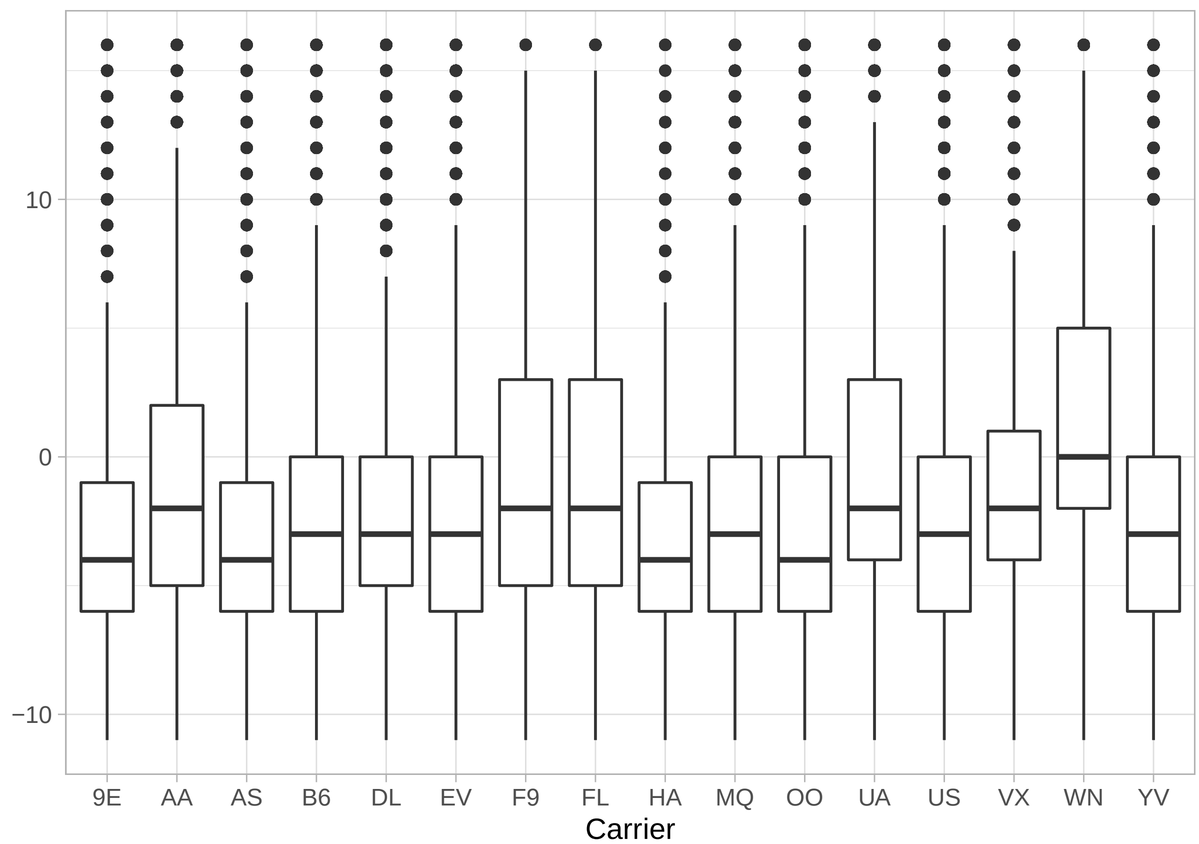
Alternatively, to create the above plots, we can use `purrr::map()` which allows us to apply a function to each column. See `?map` for more details.

```
map(df_flights %>% select(ArrDelay, DepDelay), ~ ggplot(df_flights) +
  geom_boxplot(mapping = aes(x = Carrier, y = .x)) + ylab(""))
```

```
## $ArrDelay
```



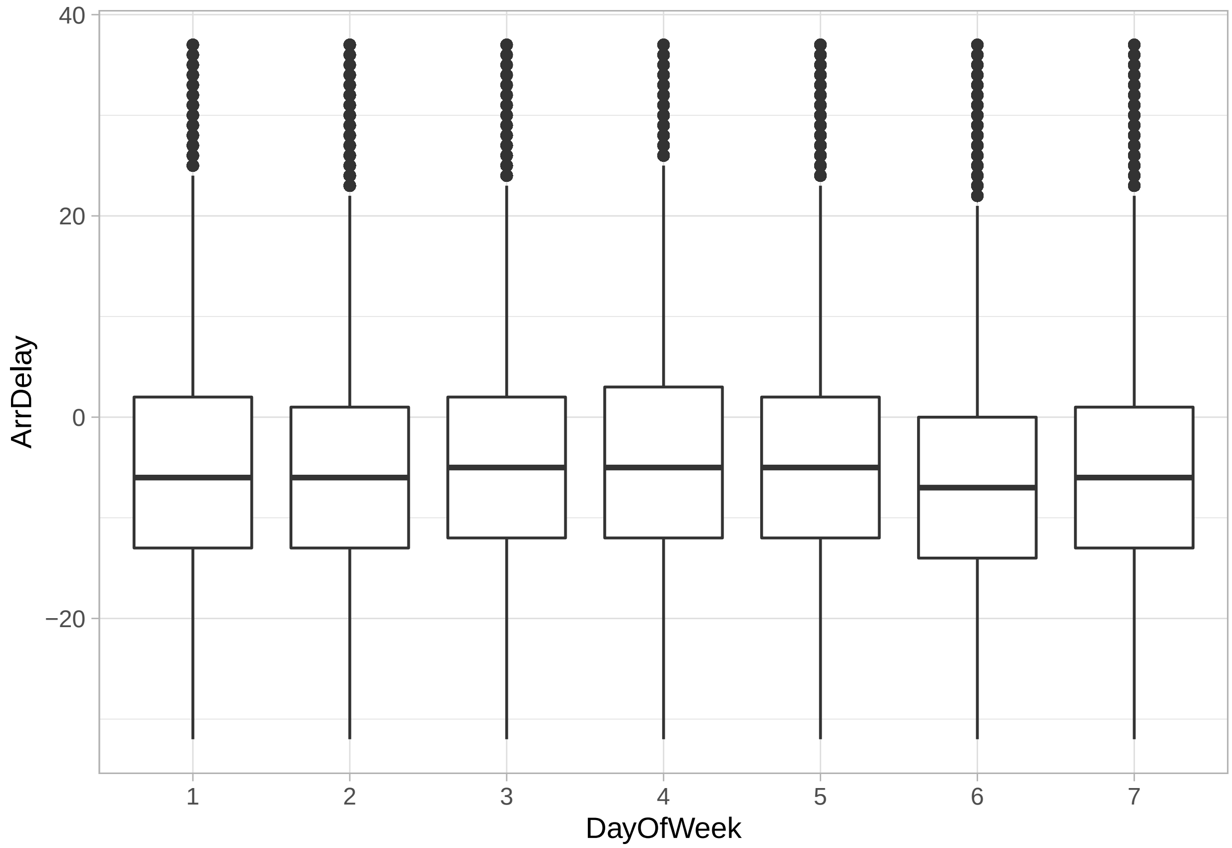

```
##
## $DepDelay
```



Are some days of the week more prone to arrival delays than others?

Again, let's make use of a box plot to visually inspect the distribution of arrival delays depending on the day of the week. To successfully accomplish this, we have to first encode the week days to `categorical` variables, that is, `factors` .

```
# Encode day of the week as a categorical and make boxplots
df_flights %>%
  mutate(DayOfWeek = factor(DayOfWeek)) %>%
  ggplot() +
  geom_boxplot(mapping = aes(x = DayOfWeek, y = ArrDelay), show.legend = FALSE)
```



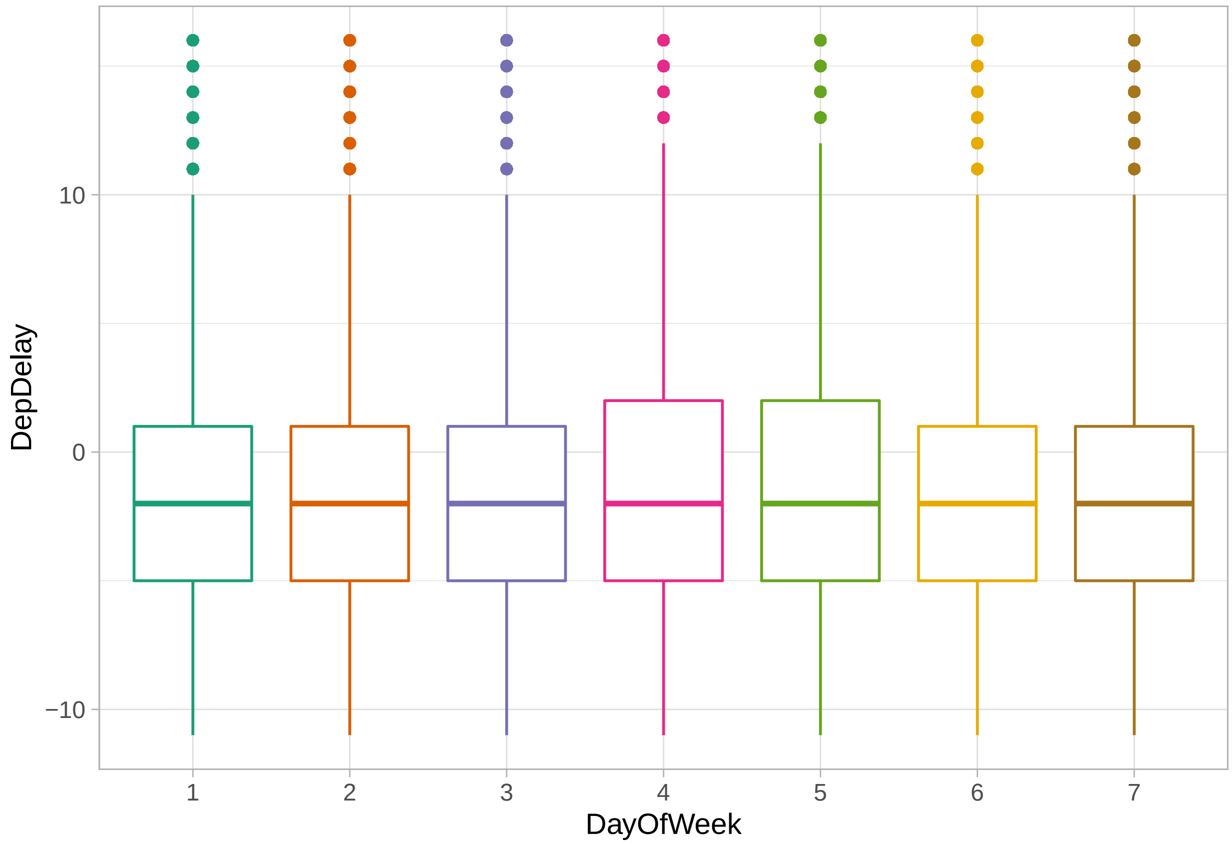
Are some days of the week more prone to departure delays than others?

Now, over to you.

Question 9. See whether you can investigate whether some days of the week (x axis) are more prone to departure delays (y axis) than others. Start by encoding day of the week as a categorical variable.

```
BEGIN QUESTION
name: Question 9
manual: false
```

```
# Encode day of the week as a categorical variable
df_flights <- df_flights %>%
  mutate(DayOfWeek = factor(DayOfWeek))

# Make a box plot of DayOfWeek and DepDelay
dep_delay_plot <- df_flights %>%
  ggplot() +
  geom_boxplot(mapping = aes(x = DayOfWeek, y = DepDelay, color = DayOfWeek), show.legend = FALSE) +
  scale_color_brewer(palette = "Dark2")

dep_delay_plot
```



What can you make out of this?

```
. = " # BEGIN TEST CONFIG
success_message: That's a great start! You have successfully encoded **DayOfWeek** as a categorical vari
failure_message: Almost there. Ensure you modified **DayOfWeek** column to a factor/category variable.

" # END TEST CONFIG

## Test ##
test_that('DayOfWeek is a factor variable', {
  expect_equal(class(df_flights$DayOfWeek), "factor")})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG
success_message: Great job! You now have yourself a beautiful box plot chart. As you can see, there does
failure_message: Let's give it another try. Ensure you have mapped the x aesthetic to **DayOfWeek** and

" # END TEST CONFIG

## Test ##
test_that('plot has expected aesthetic mappings', {

  expect_equal(dep_delay_plot$labels$x, "DayOfWeek")

  expect_equal(dep_delay_plot$labels$y, "DepDelay")
```

```
})
```

```
## Test passed
```

Great progress you are having!

Which departure airport has the highest average departure delay?

To answer this, we have to first **group** the data **by** `OriginAirportName` and then **summarize** the observations by the **mean** of their Departure delay `DepDelay` and then **arrange** this in **desc**ending order of the mean departure delays.

Let's put this into code.

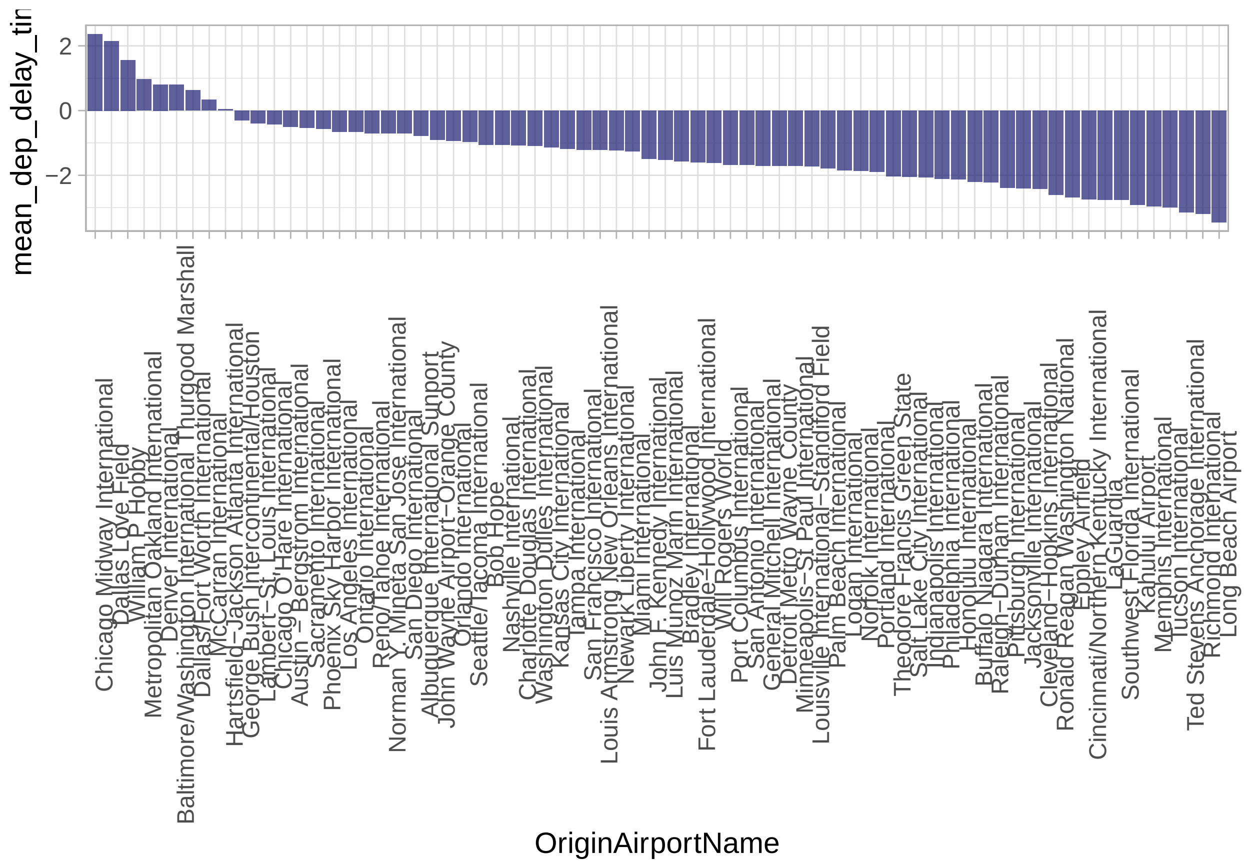
```
# Use group_by %>% summarize to find airports with highest avg DepDelay
mean_departure_delays <- df_flights %>%
  group_by(OriginAirportName) %>%
  summarize(mean_dep_delay_time = mean(DepDelay)) %>%
  arrange(desc(mean_dep_delay_time))

# Print the first 7 rows
mean_departure_delays %>%
  slice_head(n = 7)
```

```
## # A tibble: 7 x 2
##   OriginAirportName                                     mean_dep_delay_time
##   <chr>                                                               <dbl>
## 1 Chicago Midway International                                        2.37
## 2 Dallas Love Field                                                   2.15
## 3 William P Hobby                                                     1.56
## 4 Metropolitan Oakland International                                  0.965
## 5 Denver International                                                0.807
## 6 Baltimore/Washington International Thurgood Marshall                0.804
## 7 Dallas/Fort Worth International                                     0.625
```

Fantastic!

Let's represent this using bar plots.

```
mean_departure_delays %>%
  # Sort factor levels in descending order of delay time
  mutate(OriginAirportName = fct_reorder(OriginAirportName, desc(mean_dep_delay_time))) %>%
  ggplot() +
  geom_col(mapping = aes(x = OriginAirportName, y = mean_dep_delay_time), fill = "midnightblue", alpha =
  theme(
    # Rotate X markers so we can read them
    axis.text.x = element_text(angle = 90)
  )
```



Could you try and guess why Chicago Airport has most departure delay time or why Long Beach has the least?

Do late departures tend to result in longer arrival delays than on-time departures?

Question 10. Starting with the `df_flights` data, first encode `DepDel15` column (A binary indicator that departure was delayed by more than 15 minutes) as categorical.

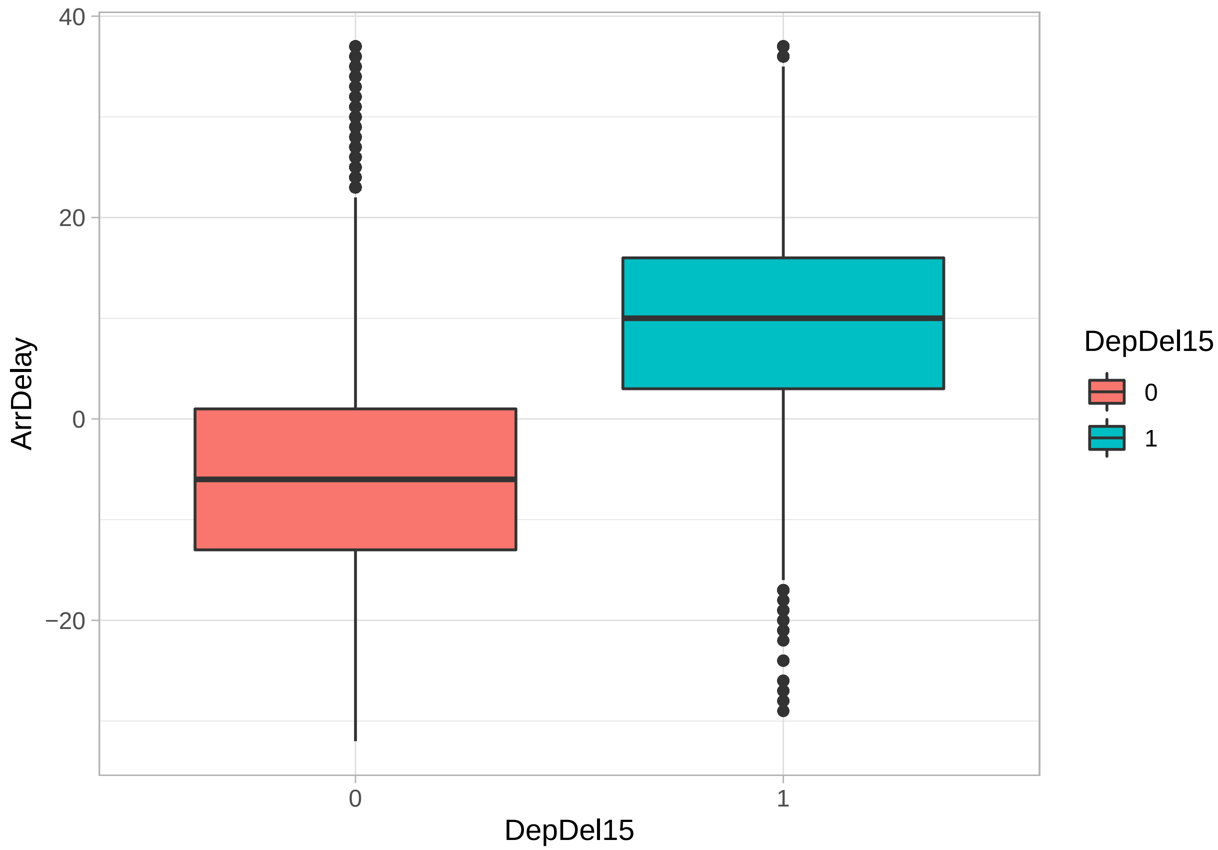
Use a **box plot** to investigate whether late departures (x-axis) tend to result in longer arrival delays (y-axis) than on-time departures. Map the fill aesthetic to the `DepDel15` variable.

> You can colour a box plot using either the `colour` aesthetic (like in the previous exercises), or, more usefully with the `fill` aesthetic.

```
BEGIN QUESTION
name: Question 10
manual: false
```

```
# Encode DepDel15 as a categorical variable
df_flights <- df_flights %>%
  mutate(DepDel15 = factor(DepDel15))

arr_delay_plot <- df_flights %>%
  ggplot() +
```

```
  geom_boxplot(mapping = aes(x = DepDel15, y = ArrDelay, fill = DepDel15))

arr_delay_plot
```



Does this surprise you?

```
. = " # BEGIN TEST CONFIG

success_message: That's a great start! You have successfully encoded **DepDel15** as a categorical vari

failure_message: Almost there. Ensure you modified **DepDel15** column to a factor/category variable.
" # END TEST CONFIG

## Test ##
test_that('DepDel15 is a factor variable', {

  expect_equal(class(df_flights$DepDel15), "factor")


})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

success_message: Great job! You now have yourself a beautiful and informative box plot chart. As you ca

failure_message: Let's give it another try. Ensure you have mapped the x aesthetic to **DepDel15**, y a
" # END TEST CONFIG

## Test ##
test_that('plot has expected aesthetic mappings', {

  expect_equal(class(arr_delay_plot$layers[[1]]$geom)[1], "GeomBoxplot")

  expect_equal(arr_delay_plot$labels$x, "DepDel15")

  expect_equal(arr_delay_plot$labels$y, "ArrDelay")

  expect_equal(arr_delay_plot$labels$fill, "DepDel15")



})
```

```
## Test passed
```

Which route (from origin airport to destination airport) has the most late arrivals?

Finally, let's investigate travel routes. We'll start by adding a column `Route` that indicates the Origin and Destination airports.

```
# Add a "Route" column
df_flights <- df_flights %>%
  mutate(Route = paste(OriginAirportName, DestAirportName, sep = ">"))
```

Great! Now we can use `group_by()`, `summarize()` and `arrange()` to find the routes with the most late arrivals

```
# Make grouped summaries to find the total delay associated with a particular route
df_flights %>%
  group_by(Route) %>%
  summarize(ArrDel15 = sum(ArrDel15)) %>%
  arrange(desc(ArrDel15))
```

```
## # A tibble: 2,479 x 2
##    Route                                                         ArrDel15
##    <chr>                                                            <dbl>
##  1 San Francisco International>Los Angeles International              90
##  2 Los Angeles International>San Francisco International              69
##  3 LaGuardia>Hartsfield-Jackson Atlanta International                 68
##  4 Los Angeles International>John F. Kennedy International            52
##  5 LaGuardia>Charlotte Douglas International                          51
##  6 Chicago O'Hare International>Hartsfield-Jackson Atlanta Internation~       44
##  7 LaGuardia>Chicago O'Hare International                             44
```

```
##  8 Los Angeles International>McCarran International                            43
##  9 John F. Kennedy International>San Francisco International                    42
## 10 McCarran International>Los Angeles International                            41
## # ... with 2,469 more rows
```

Which route has the highest average arrival delay time?

Over to you!

Question 11. Starting with the `df_flights` data, group the observations by `Route` then create a summary tibble with a column name `ArrDelay` which represents the mean arrival delay time. Arrange this in descending order.

Assign your results to a variable name `df_route_arrdelay`

```
BEGIN QUESTION
name: Question 11
manual: false
```

```r
# Create grouped summaries of the arrival delay time
df_route_arrdelay <- df_flights %>%
  group_by(Route) %>%
  summarise(ArrDelay = mean(ArrDelay)) %>%
  arrange(desc(ArrDelay))


# Print the first 5 rows
df_route_arrdelay %>%
  slice_head(n = 5)
```

```
## # A tibble: 5 x 2
##   Route                                                               ArrDelay
##   <chr>                                                                  <dbl>
## 1 Louis Armstrong New Orleans International>Ronald Reagan Washington N~    24.5
## 2 Cleveland-Hopkins International>Palm Beach International                18
## 3 John F. Kennedy International>Louisville International-Standiford Fi~    18
## 4 Cleveland-Hopkins International>Philadelphia International              12.8
## 5 Memphis International>Denver International                               9.76
```

```r
. = " # BEGIN TEST CONFIG

success_message: That's a great start! Your tibble dimensions are looking great!

failure_message: Almost there. Let's check the tibble dimensions again. The output tibble should have c
" # END TEST CONFIG

## Test ##
test_that('summary tibble has correct dimensions', {

  expect_output(glimpse(df_route_arrdelay), "Rows: 2,479\nColumns: 2", fixed = TRUE)

  expect_equal(sort(names(df_route_arrdelay)), c("ArrDelay", "Route"))
```

```
})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

success_message: Fantastic! You have successfully grouped_by, summarized and arranged the observations i


failure_message: Almost there. Ensure the tibble is arranged in descending order of their mean delay tim
" # END TEST CONFIG

## Test ##
test_that('summary tibble has correct values', {

  expect_equal(slice(df_route_arrdelay, 1)$ArrDelay, 24.5)

  expect_equal(slice(df_route_arrdelay, 2476)$Route, "Eppley Airfield>LaGuardia")

})
```

```
## Test passed
```

Congratulations on finishing the first challenge! We'll wrap it at that for now. Of course there are other ways to approach this challenge. So please feel free to experiment, google and share your solutions with friends.

See you in the next module where we get started with Machine Learning!

Happy Learning,

Eric, Gold Microsoft Learn Student Ambassador.

```
library(here)
```

```
## here() starts at C:/Users/medewan/Documents/GitHub/ml-basics-R
```

```
library(rmd2jupyter)
rmd2jupyter("01_Data_Exploration_Solution_Ottr.Rmd")
```

```
## file saved as 01_Data_Exploration_Solution_Ottr.ipynb
```