Challenge - Data exploration

2023-02-28

Challenge: Explore real-world flight data

As a data scientist, a significant part of your role is to explore, analyze, and visualize data. In this challenge, you'll explore a real-world dataset that contains flight data from the US Department of Transportation.

Let's start by loading the packages you'll need for this exploration.

```
# Load the required packages
suppressPackageStartupMessages({
  library(tidyverse)
  library(summarytools)
  library(glue)
  library(patchwork)
  })

# Initialize github repo path
# containing test files used to check your answers
testsFolderPath <- "https://raw.githubusercontent.com/MicrosoftDocs/mslearn-machine-learning-with-r/mai
```

Now, you can import the data 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 DayOf~1 Carrier Origi~2 Origi~3 Origi~4 Origi~5 DestA~6
##   <dbl> <dbl>      <dbl>   <dbl> <chr>     <dbl> <chr>   <chr>   <chr>     <dbl>
## 1  2013     9         16       1 DL        15304 Tampa ~ Tampa   FL        12478
## 2  2013     9         23       1 WN        14122 Pittsb~ Pittsb~ PA        13232
## 3  2013     9          7       6 AS        14747 Seattl~ Seattle WA        11278
## 4  2013     7         22       1 OO        13930 Chicag~ Chicago IL        11042
## 5  2013     5         16       4 DL        13931 Norfol~ Norfolk VA        10397
## 6  2013     7         28       7 UA        12478 John F~ New Yo~ NY        14771
## 7  2013    10          6       7 WN        13796 Metrop~ Oakland CA        12191
## # ... with 10 more variables: DestAirportName <chr>, DestCity <chr>,
## #   DestState <chr>, CRSDepTime <dbl>, DepDelay <dbl>, DepDel15 <dbl>,
## #   CRSArrTime <dbl>, ArrDelay <dbl>, ArrDel15 <dbl>, Cancelled <dbl>, and
## #   abbreviated variable names 1: DayOfWeek, 2: OriginAirportID,
## #   3: OriginAirportName, 4: OriginCity, 5: OriginState, 6: DestAirportID
```

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 the departure was delayed (flights that left ahead of schedule have a negative value)
- DelDelay15: A binary indicator that the departure was delayed by more than 15 minutes (and is therefore considered "late")
- CRSArrTime: The scheduled arrival time
- ArrDelay: The number of minutes the arrival was delayed (flights that arrived ahead of schedule have a negative value)
- ArrDelay15: A binary indicator that the arrival was delayed by more than 15 minutes (and is therefore considered "late")
- Cancelled: A binary indicator that the flight was canceled

Your challenge is to explore the flight data to analyze factors that might cause delays in a flight's departure or arrival.

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 noticeable 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 departure airport to destination airport) has the most late arrivals?
  - Which route has the highest average arrival delay?

Sometimes, when there are a lot of columns in the data, it can be difficult at first to get a good grasp of it by using `slice_head`.

By using `glimpse`, you can view a transposed version of the data frame, where columns are displayed vertically and the data is displayed horizontally. This makes it possible to easily view every column in a data frame. At the same time, `glimpse` shows the dimension of the data frame 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 the data for missing values

After you've imported your data, it's always a good idea to clean it. The importance of this task is often underestimated, yet it's a fundamental step that's necessary for successful 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
```

It looks like there are some NA (missing values) `late departure` indicators in the **DepDel15** column. A departure is considered late if the delay is 15 minutes or more, so let's see the delays for the ones with an NA late indicator:

Step 1

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

Fill in the placeholder `....` with the right code.

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

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

Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%201.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## Excellent. You have successfully selected columns **DepDelay** and **DepDel15** and then filtered the
## Fantastic. Your tibble dimensions are also correct.
## All tests passed!
```

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

```
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 flights in DepDel15 (a binary indicator that the departure was delayed by more than 15 minutes) all have a corresponding delay of 0 in DepDelay(the number of minute the departure was delayed). Let's check by looking at the summary statistics for the DepDelay records:

```
# 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.

Step 2

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

Fill in the placeholder .... with the right code.

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

Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%202.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## Good job! No more missing values in column DepDel15.
## Fantastic. Your tibble dimensions are also correct.
## All tests passed!
```

Good job! There are no missing values now. Let's take this a little further.

Clean the 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 specified column.

```
# 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))
  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
```

Step 3

Starting with the `df_flights` data, keep only the **DepDelay** column. Assign this to a variable named `df_col`.

After you have this figured out, call the function `show_distribution` with the argument names and corresponding values `var_data = df_col` and `binwidth = 100`.

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

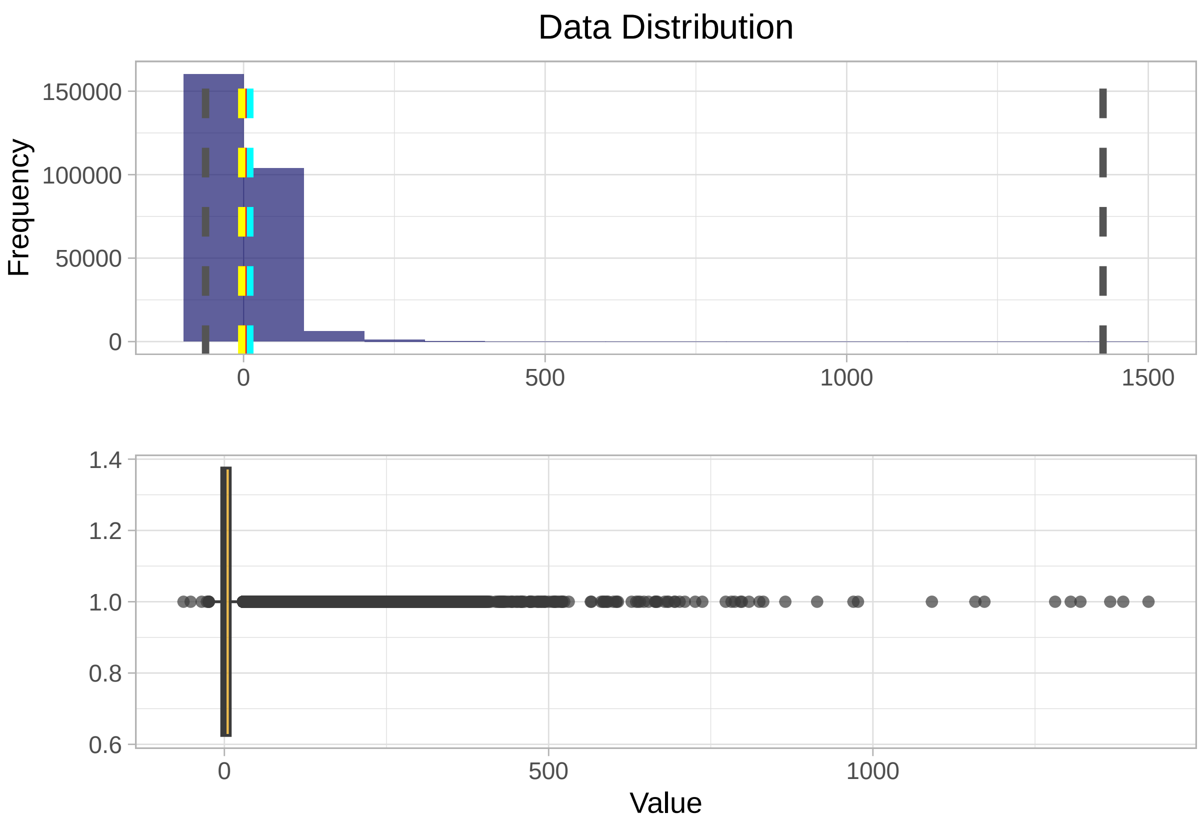
Fill in the placeholder `....` with the right code.

```r
# Select DepDelay column
df_col <- df_flights %>%
  select(DepDelay)

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

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

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



Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%203.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## Fantastic! You have successfully selected column **DepDelay**
## Your summary statistics are also looking great!
## All tests passed!
```

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

Step 4

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

After you have this figured out, call the function show_distribution with the argument names and corresponding values `var_data = df_col` and `binwidth = 100` (value of the width of each bin along the x-axis).

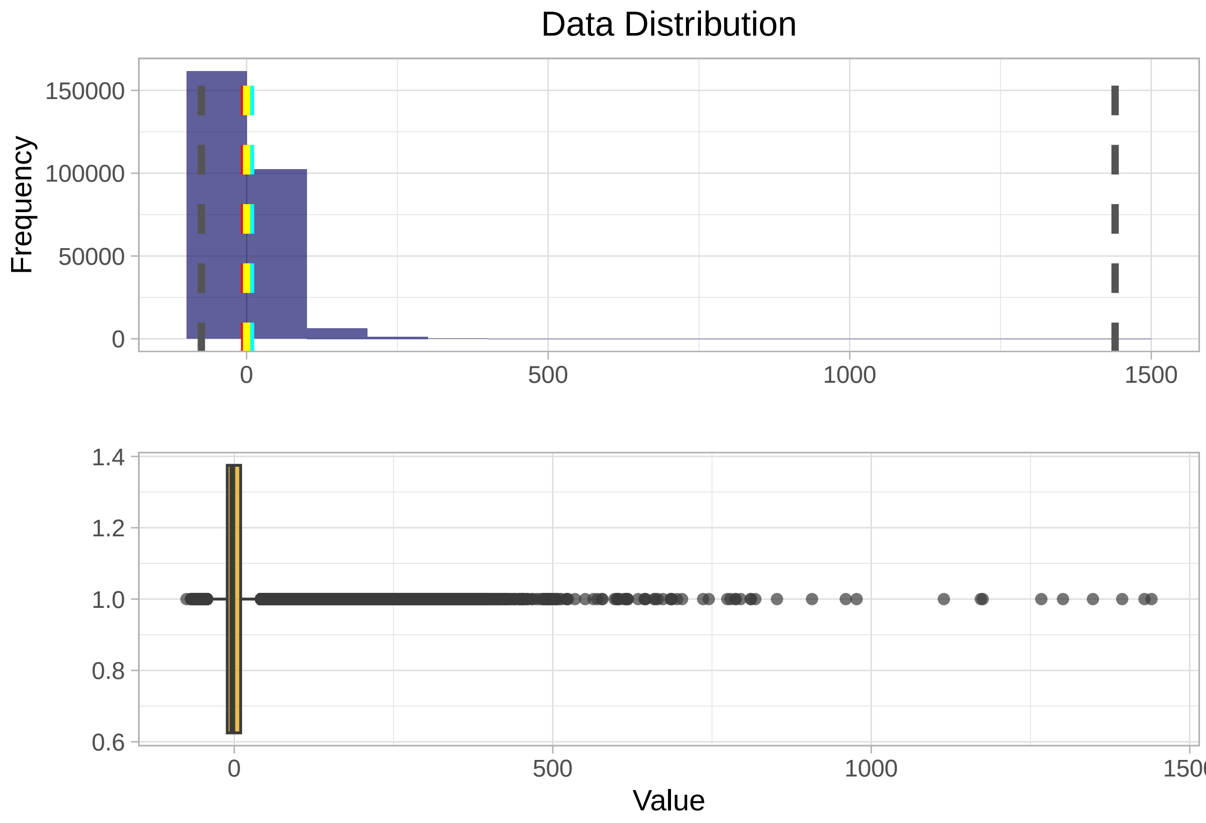
From the function output, what's the distribution of **ArrDelay**?

Fill in the placeholder `....` with the right code.

```
# 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]]
```



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

```r
# Trim outliers for ArrDelay based on 1st and 90th 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 you have quantiles corresponding to 1% and 90%, let's filter the `df_flights` data to include only rows whose arrival delay falls within this range.

Step 5

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

Fill in the placeholder `....` with the right code.

```
# Filter data to remove outliers
df_flights <- df_flights %>%
  filter(ArrDelay > arrdelay_01pcntile, ArrDelay < arrdelay_90pcntile)
```

Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%205.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## Well done! You have successfully filtered the data to include observations whose Arrival Delay falls
## All tests passed!
```

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

Step 6*

Starting with the `df_flights` data, obtain quantiles that correspond to 1% and 90%. Assign these values to the variables named `depdelay_01pcntile` and `depdelay_90pcntile`, respectively.

Fill in the placeholder `....` with the right code.

```
# 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
```

Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%206.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## That's it. You've got the correct values for the 1st and 90th percentiles.
## All tests passed!
```

Good job!

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

Step 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`.

Fill in the placeholder `....` with the right code.

```
# Filter data to remove outliers
df_flights <- df_flights %>%
  filter(DepDelay > depdelay_01pcntile, DepDelay < depdelay_90pcntile)
```

Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%207.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

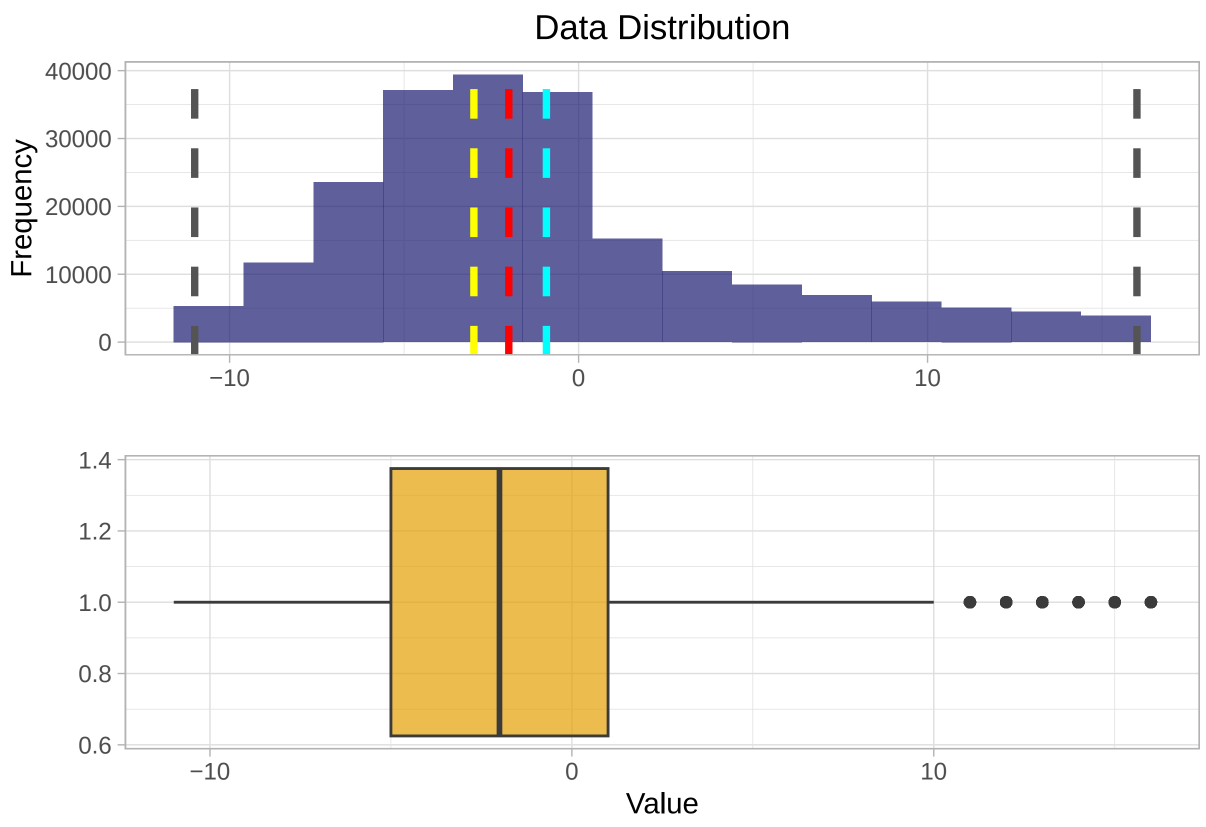
```
## Well done! You have successfully filtered the data to include observations whose Departure Delay fal
## All tests passed!
```

You rock!

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

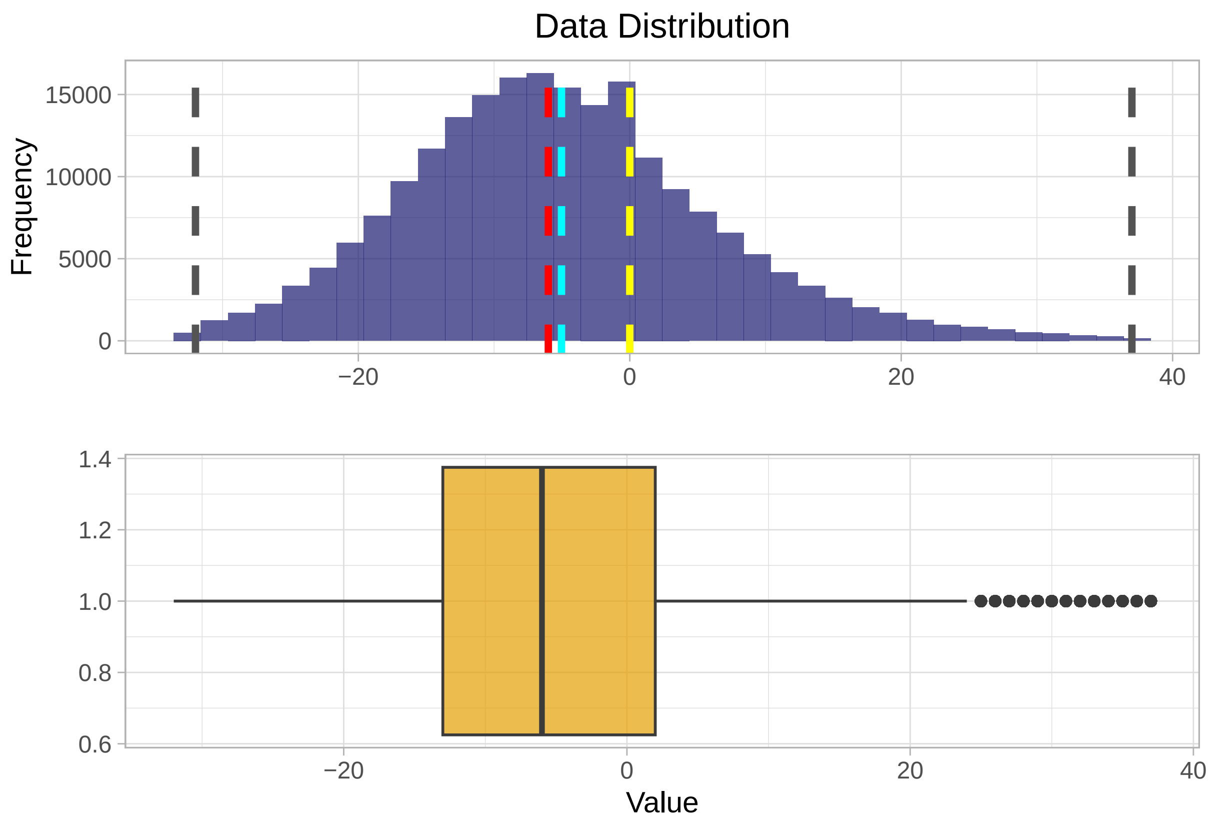
```
# Distribution of DepDelay
show_distribution(var_data = select(df_flights, DepDelay), binwidth = 2)
```

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



```
# 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, you 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?

Step 8

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

Fill in the placeholder `....` with the right code.

```
# Summarize the departure and arrival delays by finding the mean
df_delays <- df_flights %>%
 summarise(across(c(ArrDelay,DepDelay), mean))

df_delays
```

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

Test your answer:

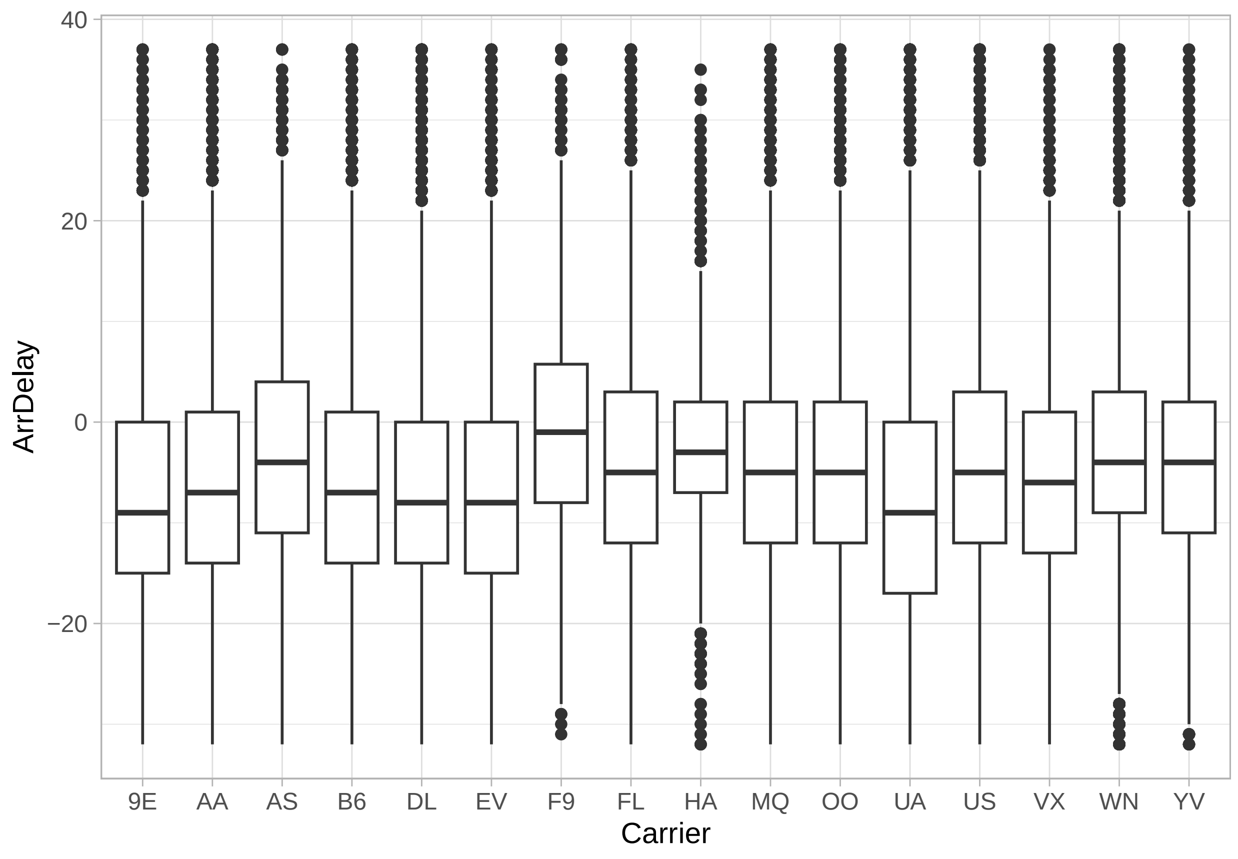
```r
testFilePath <- paste(testsFolderPath, "Question%208.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## Fantastic! You have successfully found the mean Delay time across **DepDelay** and **ArrDelay** colu
## All tests passed!
```

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

A box plot can be a good way to graphically depict 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))
```

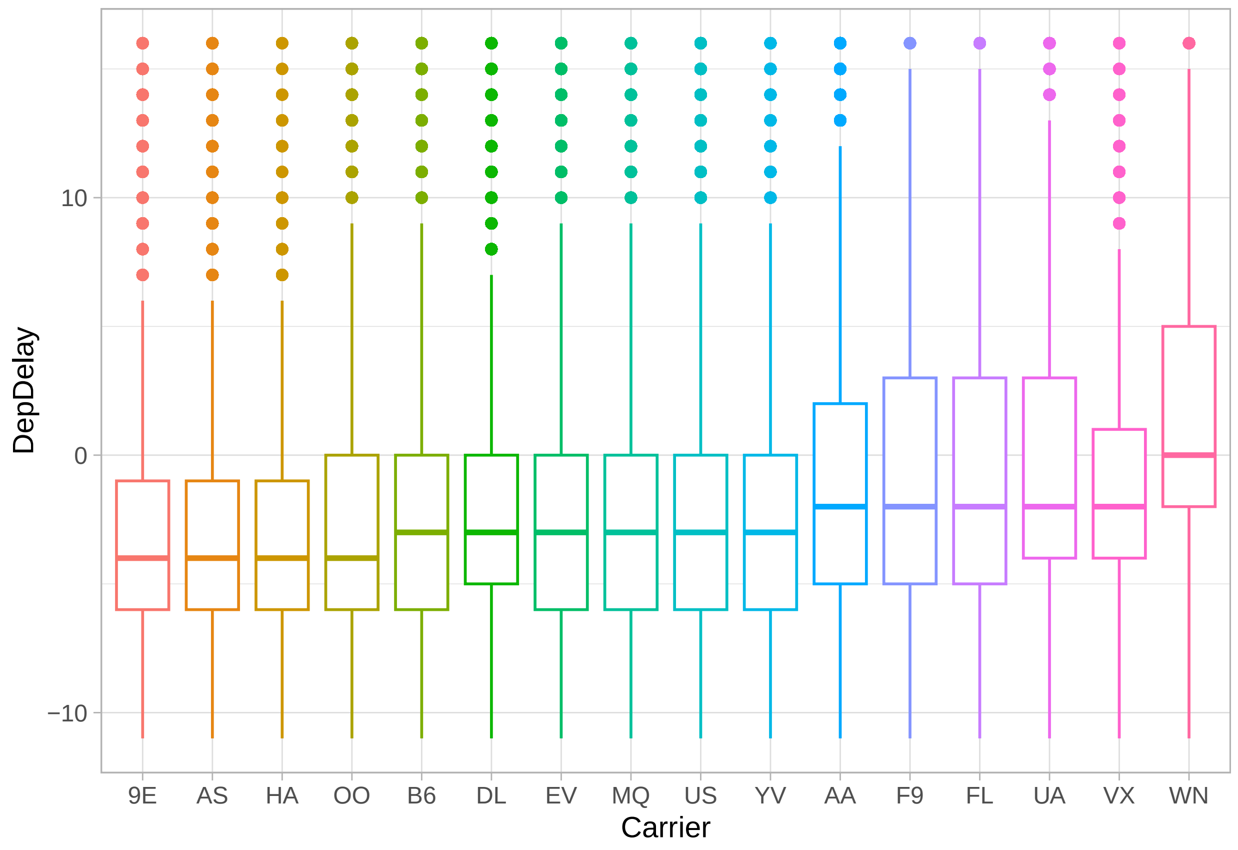


How do the carriers compare in terms of departure delay performance?

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

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

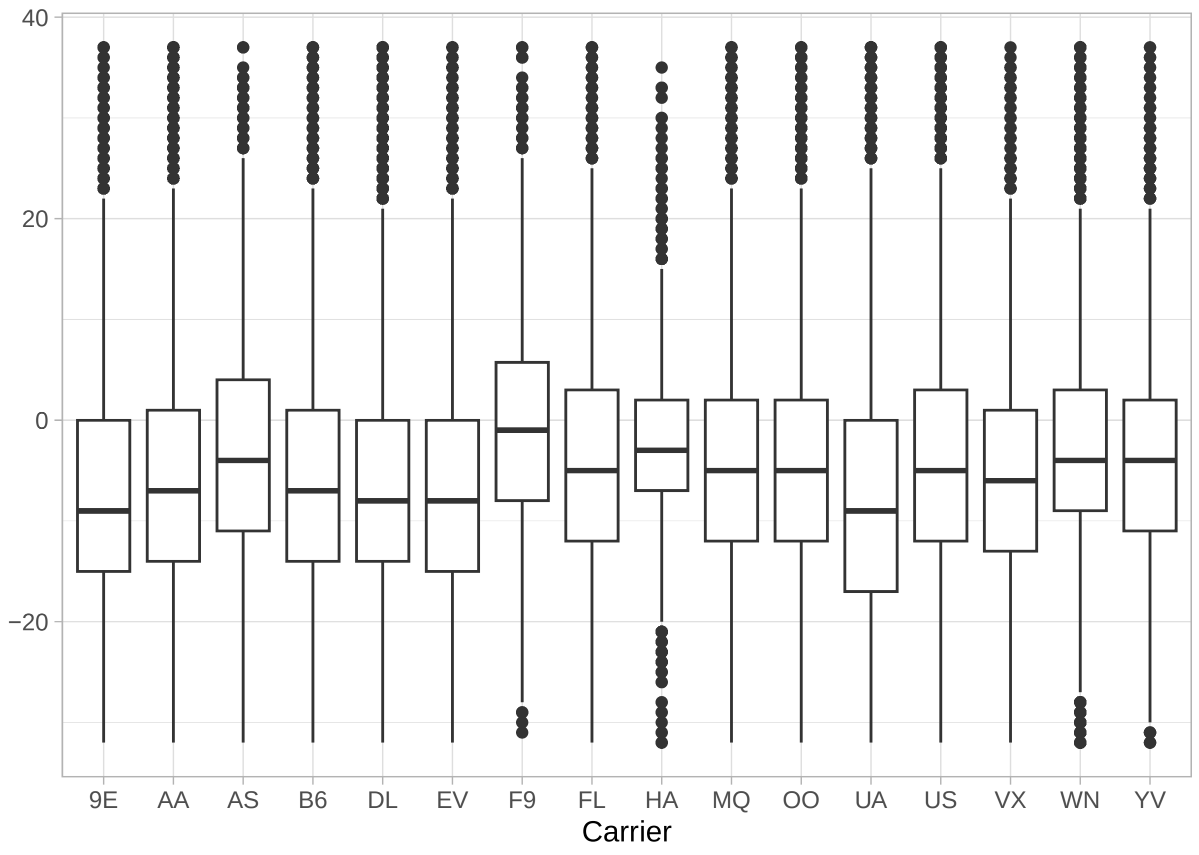
```r
df_flights %>%
  mutate(Carrier = fct_reorder(Carrier, DepDelay)) %>%
  ggplot() +
  geom_boxplot(mapping = aes(x = Carrier, y = DepDelay, color = Carrier),
  show.legend = FALSE)
```



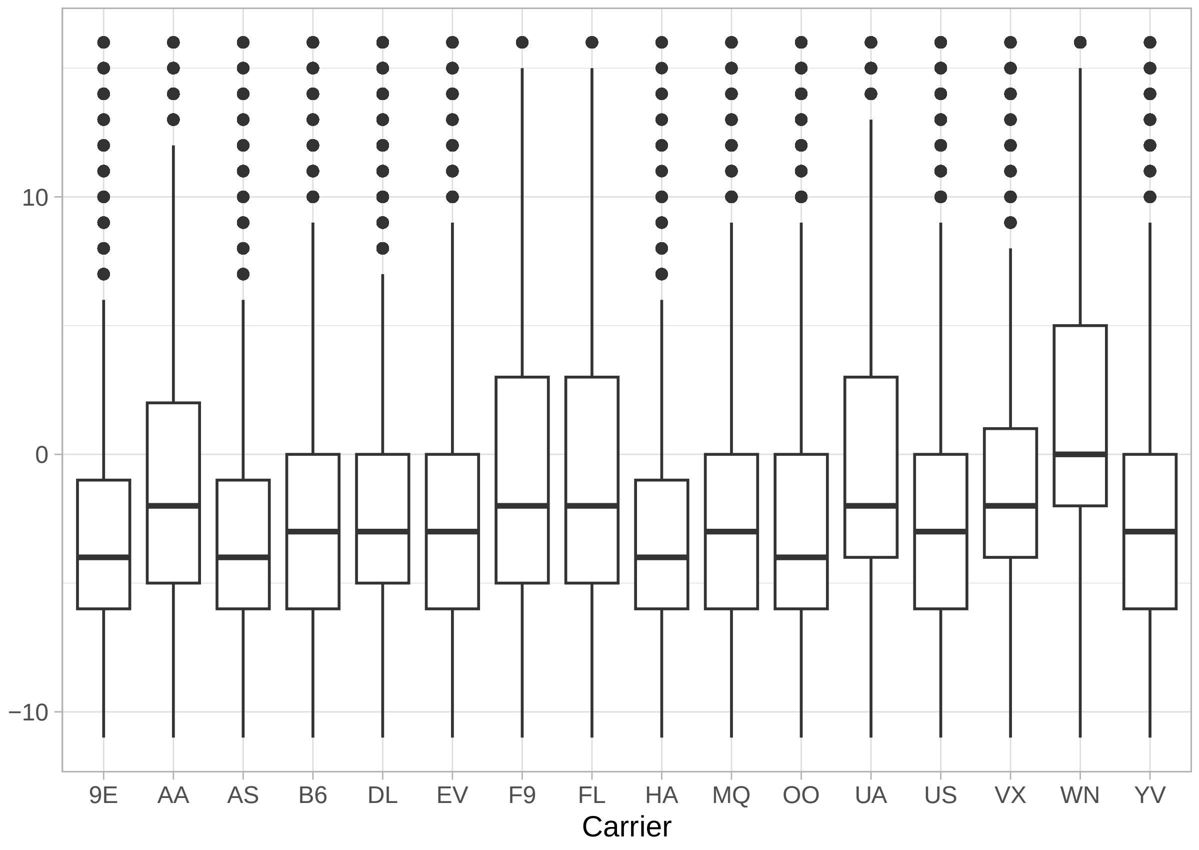
Alternatively, to create the preceding plots, you can use purr::map() to apply a function to each column. See ?map for more details.

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

```
## $ArrDelay
```



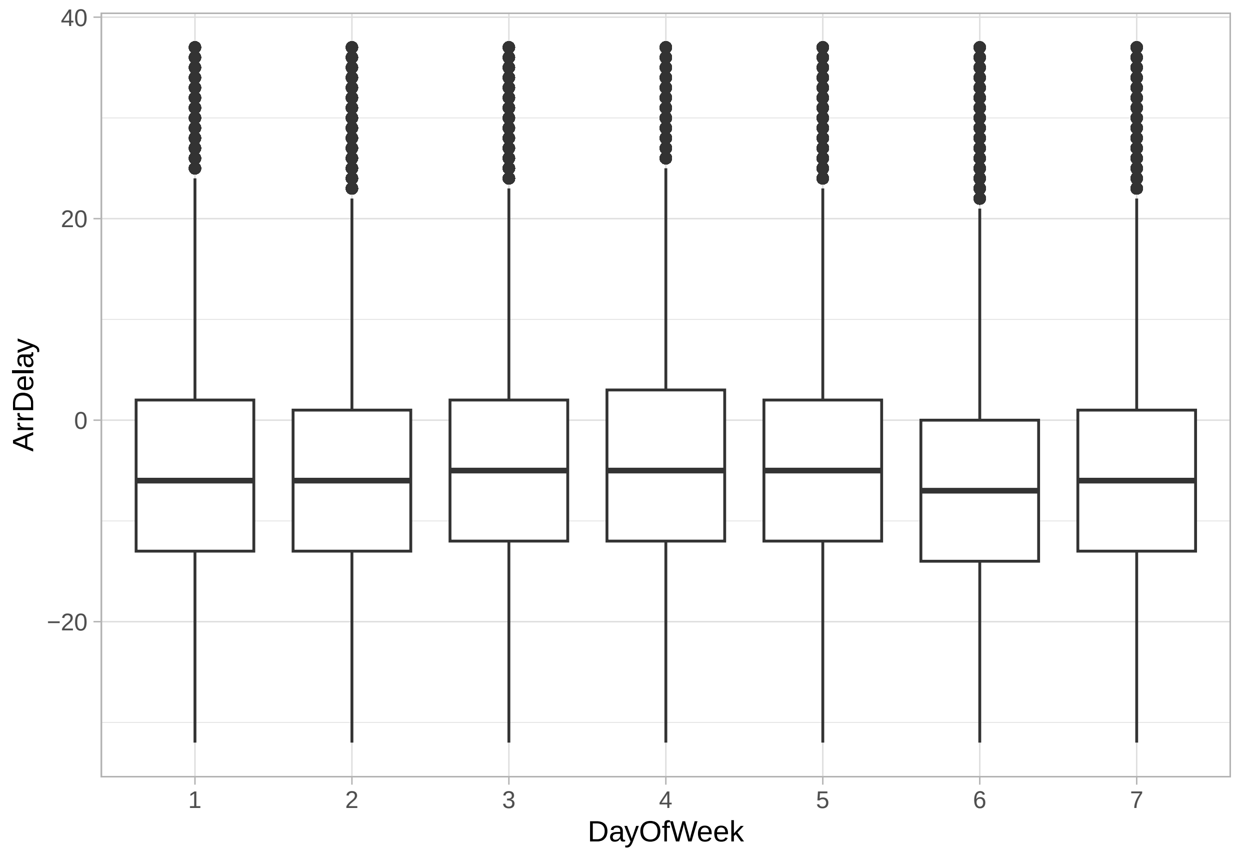
```
##
## $DepDelay
```



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 according to day of the week. To successfully accomplish this, you first have to encode days of the week as `categorical` variables (that is, `factors`).

```r
# 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.

Step 9

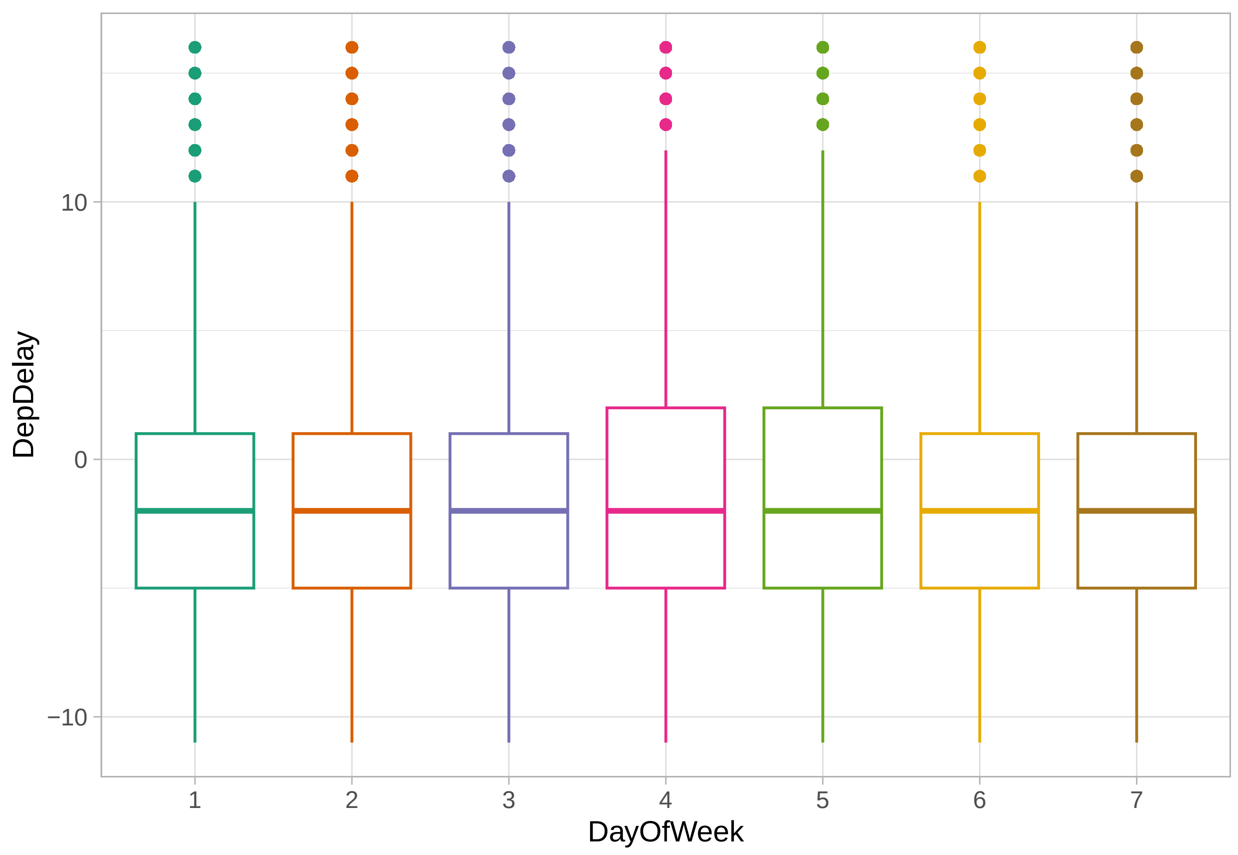
Let's investigate whether some days of the week (x-axis) are more prone to departure delays (y-axis) than others. Start by encoding the day of the week as a categorical variable.

Fill in the placeholder .... with the right code.

```r
# 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? Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%209.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## That's a great start! You have successfully encoded **DayOfWeek** as a categorical variable.
## Great job! You now have yourself a beautiful box plot chart. As you can see, there doesn't seem to be
## All tests passed!
```

Great progress!

Which departure airport has the highest average departure delay?

To answer this question, you have to group the data by `OriginAirportName`, summarize the observations by the mean of their departure delay `DepDelay`, and then arrange them in descending order of the mean departure delays.

First, 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))
```

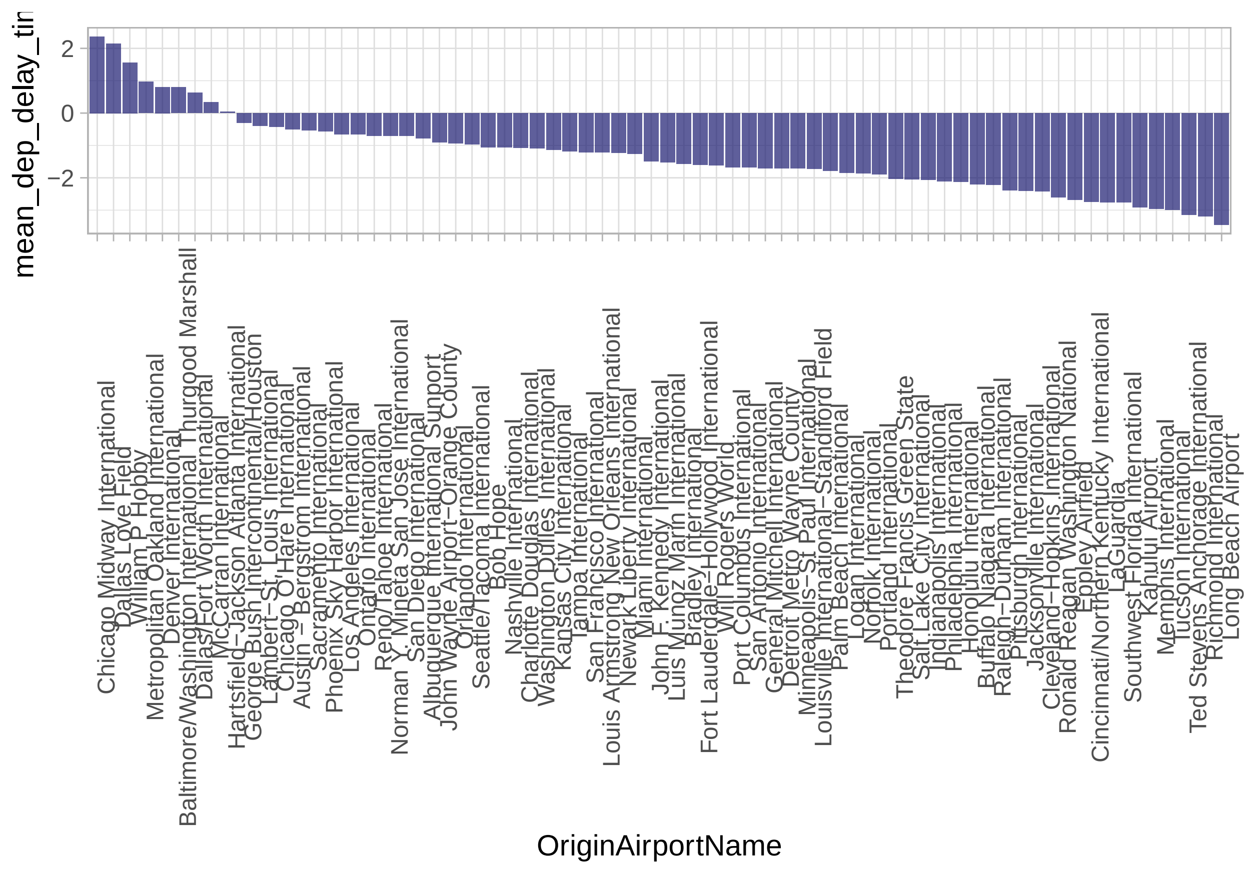
```r
# Print the first 7 rows
mean_departure_delays %>%
  slice_head(n = 7)
```

```
## # 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!

Now represent this visually by using bar plots.

```r
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 = 0.7) +
  theme(
    # Rotate X markers so we can read them
    axis.text.x = element_text(angle = 90)
  )
```



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

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

Step 10

Starting with the `df_flights` data, first encode the **DepDel15** column (a binary indicator that the 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.

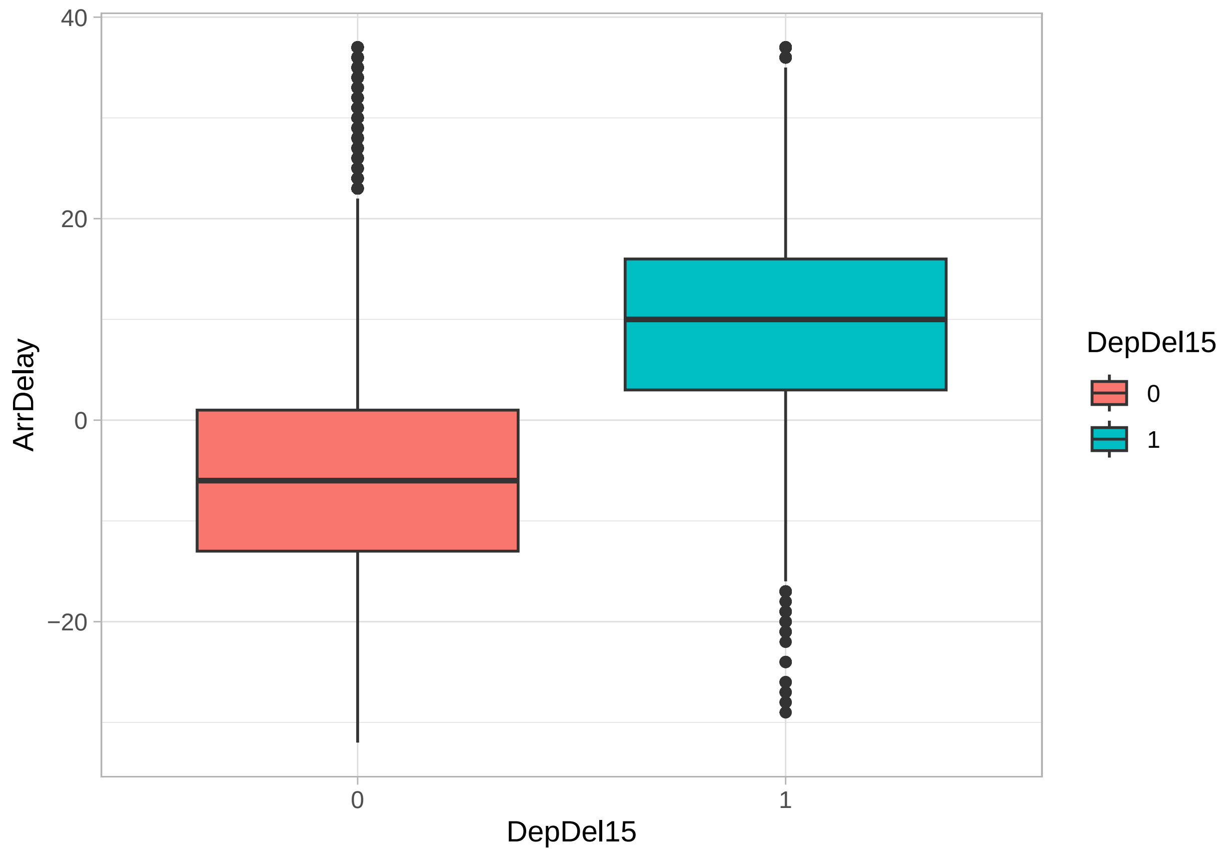
> **Tip**: You can color a box plot by using either the `colour` aesthetic (as in previous exercises) or, more usefully, the `fill` aesthetic.

Fill in the placeholder `....` with the right code.

```r
# 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? Test your answer:

```
testFilePath <- paste(testsFolderPath, "Question%2010.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## That's a great start! You have successfully encoded **DepDel15** as a categorical variable.
## Great job! You now have yourself a beautiful and informative box plot chart. As you can see, flights
## All tests passed!
```

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

Finally, let's investigate travel routes. Start by adding a **Route** column that indicates the departure and destination airports.

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

Great! Now you 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                                                           ArrDe~1
##    <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 International     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, and abbreviated variable name 1: ArrDel15
```

Which route has the highest average arrival delay time?

Over to you!

Step 11

Starting with the `df_flights` data, group the observations by `Route`, and 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 named `df_route_arrdelay`.

Fill in the placeholder `....` with the right code.

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


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

```
## # A tibble: 5 x 2
##   Route                                                     ArrDelay
##   <chr>                                                        <dbl>
## 1 Indianapolis International>Logan International                 -26
## 2 Jacksonville International>Chicago Midway International        -24.1
## 3 Denver International>Kahului Airport                           -22.7
## 4 Eppley Airfield>LaGuardia                                      -20.8
## 5 Lambert-St. Louis International>Cleveland-Hopkins International    -20
```

Test your answer:

```r
testFilePath <- paste(testsFolderPath, "Question%2011.R", sep="", collapse=NULL)
. <- ottr::check(testFilePath)
```

```
## Test Question 11 - 1 passed
## That's a great start! Your tibble dimensions are looking great!
##
## Test Question 11 - 2 failed:
## Almost there. Ensure the tibble is arranged in descending order of their mean delay time.
## Test failed
```

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

See you in the next module, where we'll get started with machine learning.