Lab Sheet 4: Basics of data analysis in R and a Simple Sentiment Analysis

Lab Sheet 4a: Basics of data analysis in R

Introduction

In this section, we will explore basic data transformation techniques using the `dplyr` package with the `nycflights13` dataset, which contains information about all flights that departed from NYC in 2013.

Load Necessary Packages

```
# Load required packages
library(nycflights13)  # Dataset containing flight data
library(tidyverse)     # A collection of R packages for data manipulation and visualization
```

Data Transformation with dplyr

1. Filtering Data We can use the `filter()` function to subset our data based on certain conditions.

```
# Filter flights that had a departure delay greater than 120 minutes
flights |>
  filter(dep_delay > 120)
```

```
## # A tibble: 9,723 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      848           1835       853     1001
##  2  2013     1     1      957            733       144     1056
##  3  2013     1     1     1114            900       134     1447
##  4  2013     1     1     1540           1338       122     2020
##  5  2013     1     1     1815           1325       290     2120
##  6  2013     1     1     1842           1422       260     1958
##  7  2013     1     1     1856           1645       131     2212
##  8  2013     1     1     1934           1725       129     2126
##  9  2013     1     1     1938           1703       155     2109
## 10  2013     1     1     1942           1705       157     2124
## # i 9,713 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
# Flights that departed on January 1
flights |>
  filter(month == 1 & day == 1)
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
```

```
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # i 832 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
# Flights that departed in November or December
flights |>
  filter(month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013    11     1        5           2359         6      352
##  2  2013    11     1       35           2250       105      123
##  3  2013    11     1      455            500        -5      641
##  4  2013    11     1      539            545        -6      856
##  5  2013    11     1      542            545        -3      831
##  6  2013    11     1      549            600       -11      912
##  7  2013    11     1      550            600       -10      705
##  8  2013    11     1      554            600        -6      659
##  9  2013    11     1      554            600        -6      826
## 10  2013    11     1      554            600        -6      749
## # i 55,393 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
# Another way to filter for November or December
flights |>
  filter(month %in% c(11, 12))
```

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

```
## # i 55,393 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

2. Arranging Data The `arrange()` function helps us sort the data.

```
# Arrange flights by year, month, day, and departure time
flights |>
  arrange(year, month, day, dep_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
# Arrange flights by departure delay in descending order
flights |>
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     9      641            900      1301     1242
##  2  2013     6    15     1432           1935      1137     1607
##  3  2013     1    10     1121           1635      1126     1239
##  4  2013     9    20     1139           1845      1014     1457
##  5  2013     7    22      845           1600      1005     1044
##  6  2013     4    10     1100           1900       960     1342
##  7  2013     3    17     2321            810       911      135
##  8  2013     6    27      959           1900       899     1236
##  9  2013     7    22     2257            759       898      121
## 10  2013    12     5      756           1700       896     1058
## # i 336,766 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

3. Distinct Values To find unique values in our dataset, we can use `distinct()`.

```r
# Remove duplicate rows from the dataset
flights |>
  distinct()
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```r
# Keep only unique combinations of origin and destination
flights |>
  distinct(origin, dest, .keep_all = TRUE)
```

```
## # A tibble: 224 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # i 214 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```r
# Count unique combinations of origin and destination
flights |>
  count(origin, dest, sort = TRUE)
```

```
## # A tibble: 224 x 3
##    origin dest      n
##    <chr>  <chr> <int>
##  1 JFK    LAX   11262
##  2 LGA    ATL   10263
```

```
##   3 LGA    ORD    8857
##   4 JFK    SFO    8204
##   5 LGA    CLT    6168
##   6 EWR    ORD    6100
##   7 JFK    BOS    5898
##   8 LGA    MIA    5781
##   9 JFK    MCO    5464
##  10 EWR    BOS    5327
## # i 214 more rows
```

4. Creating New Variables with `mutate()` The `mutate()` function allows us to create new variables.

```
# Calculate the gain in minutes and speed of flights
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60  # Speed in miles per hour
  )
```

```
## # A tibble: 336,776 x 21
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 14 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, gain <dbl>, speed <dbl>
```

5. Using Pipes The pipe operator (`|>`) lets us chain commands together for clearer and more concise code.

```
# Filter for flights to IAH, calculate speed, and arrange by speed
flights |>
  filter(dest == "IAH") |>
  mutate(speed = distance / air_time * 60) |>
  select(year:day, dep_time, carrier, flight, speed) |>
  arrange(desc(speed))
```

```
## # A tibble: 7,198 x 7
##     year month   day dep_time carrier flight speed
##    <int> <int> <int>    <int> <chr>    <int> <dbl>
##  1  2013     7     9      707 UA         226  522.
##  2  2013     8    27     1850 UA        1128  521.
##  3  2013     8    28      902 UA        1711  519.
##  4  2013     8    28     2122 UA        1022  519.
##  5  2013     6    11     1628 UA        1178  515.
```

```
##  6  2013     8    27     1017 UA          333  515.
##  7  2013     8    27     1205 UA         1421  515.
##  8  2013     8    27     1758 UA          302  515.
##  9  2013     9    27      521 UA          252  515.
## 10  2013     8    28      625 UA          559  515.
## # i 7,188 more rows
```

6. Grouping Data Grouping data allows us to perform calculations on subsets of the data.

```
# Group by month and calculate average departure delay
flights |>
  group_by(month) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE)  # Handle missing values
  )
```

```
## # A tibble: 12 x 2
##    month avg_delay
##    <int>     <dbl>
##  1     1     10.0
##  2     2     10.8
##  3     3     13.2
##  4     4     13.9
##  5     5     13.0
##  6     6     20.8
##  7     7     21.7
##  8     8     12.6
##  9     9      6.72
## 10    10      6.24
## 11    11      5.44
## 12    12     16.6
```

Conclusion

In this section, we've covered essential data transformation techniques using `dplyr`. These skills are foundational for data analysis in R.

Lab Sheet 4b: A Simple Sentiment Analysis

Introduction

In this section, we will perform sentiment analysis on a news article using the `tidytext` and `rvest` packages. We will scrape the article, process the text, and visualize the results.

Load Necessary Packages

```
# Load required packages for sentiment analysis
library(tidytext)   # Text mining
library(rvest)      # Web scraping
library(textdata)   # Sentiment datasets
library(wordcloud)  # Word cloud visualization
library(RColorBrewer) # Color palettes for visualizations
library(wordcloud2) # Interactive word clouds
```

Scrape and Extract Text from a Website

We will use the `rvest` package to scrape a news article.

```
# Define the URL of the article
url <- 'https://indianexpress.com/article/technology/science/nobel-prize-physics-john-hopfield-geoffrey-

# Read the HTML content from the webpage
news <- read_html(url)
```

Extract Text

We will extract all the paragraphs from the article and store them in a tibble.

```
# Scrape and extract all paragraphs (<p>) and store them in a tibble
text <- tibble(
  news %>%
    html_elements('p') %>%  # Select all paragraph elements (<p>) from the HTML content
    html_text()  # Extract text from the paragraph elements
) %>%
  rename('text' = 1)  # Rename the column to "text"
```

Get Sentiments

We will load the NRC sentiment dataset, which categorizes words into different emotions.

```
# Load sentiment data
sentiments <- get_sentiments('nrc')
```

Tokenize the Text

We will split the text into individual words for analysis.

```
# Separate all texts into individual words
tokens <- text %>%
  unnest_tokens(input = text, output = word) %>%
  filter(!grepl('[0-9]', word)) # Remove numbers
```

Remove Stop-Words and Count Frequency

We will remove common stop-words and count the frequency of each word.

```
# Remove stop-words and count the frequency of each word
word_freq <- tokens %>%
  anti_join(stop_words) %>%
  count(word, sort=TRUE)
```

```
## Joining with `by = join_by(word)`
```

Visualize Word Frequencies with Word Clouds

We can visualize the most frequent words using word clouds.

```
# Set seed for reproducibility
set.seed(1234)

# Basic word cloud
wordcloud(words = word_freq$word,
          freq = word_freq$n,
```

```
          min.freq = 1,
          max.words=150,
          random.order=FALSE,
          rot.per=0.40,
          colors=brewer.pal(9, "Dark2")
)
```


```
# Interactive word cloud
wordcloud2(
  data = word_freq,
  color = "random-dark",  # Color scheme
  backgroundColor = "white",  # Background color
  fontFamily = "Arial",  # Font family
  minRotation = -pi/4,  # Minimum rotation angle
  maxRotation = pi/4   # Maximum rotation angle
)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

Count Sentiment Scores

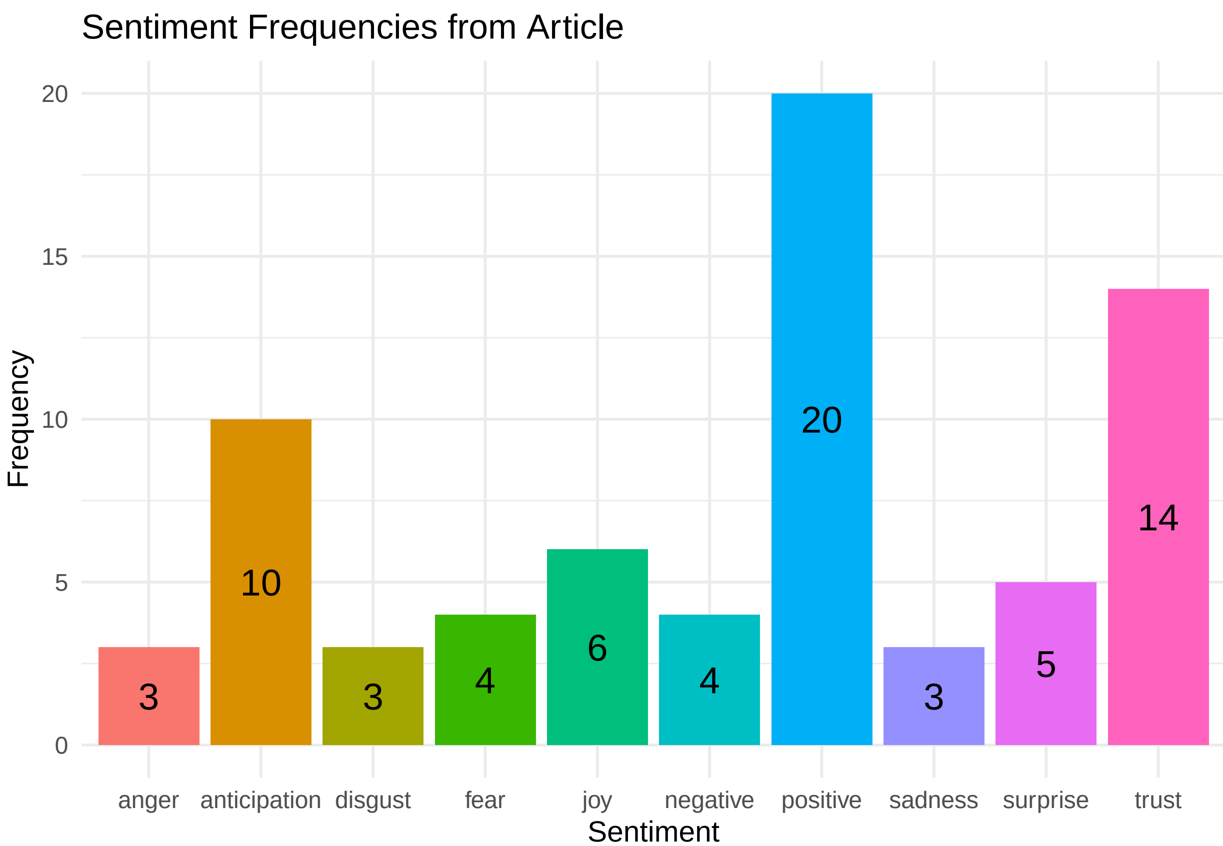
Next, we will associate each word with its sentiment score.

```
# Count the frequency of sentiments associated with each word
freq_count <- tokens %>%
  inner_join(sentiments, by='word', multiple = "all") %>%
  count(sentiment, sort = TRUE)
```

Plot the Sentiment Frequencies

Finally, we will visualize the sentiment frequencies using a bar plot.

```r
# Create a bar plot of sentiment frequencies
gg <- freq_count %>%
  ggplot(aes(x= sentiment, y= n, fill= sentiment)) +
  geom_col(show.legend = FALSE) +
  geom_text(aes(label=n), size=5, position = position_stack(vjust = 0.5)) +  # Center labels
  labs(x= 'Sentiment', y= 'Frequency') +
  ggtitle('Sentiment Frequencies from Article') +
  theme_minimal()  # Clean theme
gg
```



Conclusion

In this section, we performed sentiment analysis on a news article, demonstrating how to scrape text data, analyze it for sentiment, and visualize the results.

Additional Notes

- Ensure you have all the necessary packages installed before running the scripts.
- Explore additional datasets and articles for further practice with sentiment analysis.