Test 3 Example Code

```
library(tidyverse)
library(mdsr)

# install.packages("nycflights13")
library(nycflights13)

## Install and load package 'sqldf':
#install.packages("sqldf")
library(sqldf)


data(flights)  # load the 'flights' table into the workspace
sqldf("select * from flights limit 0,10")


## Let's rename 'name' to 'airport_name' using AS:

sqldf("SELECT
name AS airport_name,
CONCAT('(', lat, ', ', lon, ')') AS coords
FROM airports
LIMIT 0, 6")

### Using the WHERE clause to pick only the rows that meet some condition (changed the
condition from the book example):

sqldf("SELECT
year, month, day, origin, dest,
flight, carrier
FROM flights
WHERE year = 2013 AND month = 6 AND day = 26
AND origin = 'LGA'
LIMIT 0, 6")

## Note the difference between the BETWEEN and IN operators used in logical
conditions:

sqldf("
SELECT
DISTINCT CONCAT(year, '-', month, '-', day)
AS theDate
FROM flights
WHERE year = 2013 AND month = 6 AND day BETWEEN 26 and 30
AND origin = 'LGA'
LIMIT 0, 6
")

sqldf("
SELECT
DISTINCT CONCAT(year, '-', month, '-', day)
AS theDate
FROM flights
WHERE year = 2013 AND month = 6 AND day IN (26, 30)
AND origin = 'LGA'
```

```
LIMIT 0, 6
")



## loading packages:
library(mdsr)
library(DBI)
db <- dbConnect_scidb("airlines")


## Use of GROUP BY to get counts by carrier:

sqldf("
SELECT
carrier,
COUNT(*) AS numFlights,
SUM(1) AS numFlightsAlso
FROM flights
WHERE year = 2013 AND month = 6 AND day = 26
AND origin = 'LGA'
GROUP BY carrier;
")


sqldf("
SELECT
carrier,
COUNT(*) AS numFlights,
MIN(dep_time)
FROM flights
WHERE year = 2013 AND month = 6 AND day = 26
 AND origin = 'LGA'
GROUP BY carrier;
")

## This shows the worst arrival delay, by airline, for 6/26/2013:
sqldf("
 SELECT
  carrier,
  COUNT(*) AS numFlights,
  MAX(arr_delay) AS WorstDelay
FROM flights
WHERE year = 2013 AND month = 6 AND day = 26
AND origin = 'LGA'
GROUP BY carrier
LIMIT 0, 6;
")


## Ordering the result with ORDER BY:
sqldf("
SELECT
dest, SUM(1) AS numFlights
FROM flights
```

```
WHERE year = 2013
AND origin = 'LGA'
GROUP BY dest
ORDER BY numFlights DESC
LIMIT 0, 10;
")


## Can also use ORDER BY without GROUP BY (this will print the results alphabetically
by airport_name):
## Note it also works even though 'airport_name' is an alias we created:

sqldf("SELECT
name AS airport_name,
CONCAT('(', lat, ', ', lon, ')') AS coords
FROM airports
ORDER BY airport_name
LIMIT 0, 6")


### One more example of  ORDER BY without GROUP BY (this lists results in reverse
alphabetical order by carrier):

sqldf("SELECT
year, month, day, origin, dest,
flight, carrier
FROM flights
WHERE year = 2013 AND month = 6 AND day = 26
AND origin = 'LGA'
ORDER BY carrier DESC
LIMIT 0, 16")

## Which destinations have the lowest average delay for 2013 flights from LaGuardia?

sqldf("
SELECT
dest, SUM(1) AS numFlights,
AVG(arr_delay) AS avg_arr_delay
FROM flights
WHERE year = 2013
AND origin = 'LGA'
GROUP BY dest
ORDER BY avg_arr_delay ASC
LIMIT 0, 6;
")

### Restricting the result set with HAVING:
## Among all destinations with at least 730 flights in 2013 (at least 2 flights per
day),
## which destinations have the lowest average delay for 2013 flights from LaGuardia?

sqldf("
SELECT
dest, SUM(1) AS numFlights,
AVG(arr_delay) AS avg_arr_delay
```

```
FROM flights
WHERE year = 2013
AND origin = 'LGA'
GROUP BY dest
HAVING numFlights > 365 * 2
ORDER BY avg_arr_delay ASC
LIMIT 0, 6;
")

# Note this HAVING clause works even though it's based on a derived column alias
# (only true for some SQL implementations)

## Examples showing the usage of LIMIT:

sqldf("
SELECT
dest, SUM(1) AS numFlights,
AVG(arr_delay) AS avg_arr_delay
FROM flights
WHERE year = 2013
AND origin = 'LGA'
GROUP BY dest
HAVING numFlights > 365 * 2
ORDER BY avg_arr_delay ASC
LIMIT 0, 6;
")


sqldf("
SELECT
dest, SUM(1) AS numFlights,
AVG(arr_delay) AS avg_arr_delay
FROM flights
WHERE year = 2013
AND origin = 'LGA'
GROUP BY dest
HAVING numFlights > 365 * 2
ORDER BY avg_arr_delay ASC
LIMIT 5, 10;
")


#### Examples of Joins in SQL:

# The 'airports' table has the FAA code and also the full airport name.
# If we join these tables together, we can get the full airport name for the
destinations, along with the information on the flights.

# Note the JOIN does an INNER JOIN.

sqldf("
SELECT
origin, dest,
airports.name AS dest_name,
flight, carrier
```

```
FROM flights
JOIN airports ON flights.dest = airports.faa
WHERE year = 2013 AND month = 6 AND day = 26
AND origin = 'LGA'
LIMIT 0, 6;
")


## Using table aliases:
## Making 'o' the alias for 'flights', and making 'a' the alias for 'airports':

sqldf("
SELECT
origin, dest,
a.name AS dest_name,
flight, carrier
FROM flights AS o
JOIN airports AS a ON o.dest = a.faa
WHERE year = 2013 AND month = 6 AND day = 26
AND origin = 'LGA'
LIMIT 0, 6;
")

### A LEFT JOIN:
## This will capture any destination airports in the 'flights' table whose airport
information is not available in the 'airports' table

sqldf("
SELECT
year, month, day, origin, dest,
a.name AS dest_name,
flight, carrier
FROM flights AS o
LEFT JOIN airports AS a ON o.dest = a.faa
WHERE year = 2013 AND month = 6 AND day = 26
AND a.name is null
LIMIT 0, 6;
")
# Note the destinations returned that don't have a match in the 'airports' table are
all in Puerto Rico...



## Some simple examples related the Chapter 16 concepts:

# Reading in a .csv file from an external location and creating a data frame
'BP_full':

BP_full <- readr::read_csv(file="https://people.stat.sc.edu/hitchcock/Table6_8.csv")

# Creating a connection to a database driver:

con <- dbConnect(RSQLite::SQLite(), ":memory:")

# Here, ":memory:" is a special path that creates an in-memory database.
```

```
# Other database drivers require more details about the host, user, etc.

# Writing the data frame to a table in the database:
dbWriteTable(con, "presBP", BP_full)

# reading (and printing) the table:
dbReadTable(con, "presBP")

# Creating another table that is an exact copy of the 'presBP' table
# before we start updating 'presBP'

dbExecute(con, "CREATE TABLE presBPcopy AS SELECT * FROM presBP")
dbReadTable(con, "presBPcopy")

### Now doing some updates and alterations on the original 'presBP' table:

# Inserting a new row into the table:

dbSendQuery(conn = con,
            "INSERT INTO presBP
         VALUES ('DJT', 'before', 120, 60, '1/15/2016')")

dbReadTable(con, "presBP")



# Doing individual updates on the table:

dbSendQuery(conn = con,
            "UPDATE presBP
             SET sbp= 122
             WHERE subject='DJT' ")

dbReadTable(con, "presBP")

dbSendQuery(conn = con,
            "UPDATE presBP
             SET sbp= 123, dbp=62
             WHERE subject='DJT' ")

dbReadTable(con, "presBP")

# Adding a column to the table and filling in values:

#dbGetQuery(con, "ALTER TABLE presBP ADD COLUMN party TEXT") # Gives warning, better
to use 'dbExecute':

dbExecute(con, "ALTER TABLE presBP ADD COLUMN party TEXT")

dbGetQuery(con, "SELECT * FROM presBP")

dbSendQuery(conn = con,
            "UPDATE presBP
             SET party='Rep'
             WHERE subject='DJT' OR subject='GWB' ")
```

```
dbSendQuery(conn = con,
            "UPDATE presBP
             SET party='Dem'
             WHERE subject!='DJT' AND subject!='GWB' ")

dbGetQuery(con, "SELECT * FROM presBP")

# Example R code, Chapter 19, Part 1

## loading packages
library(tidyverse)
library(mdsr)

# URL of text file of Macbeth from Gutenberg Project:
macbeth_url <- "http://www.gutenberg.org/cache/epub/1129/pg1129.txt"

# getting the text from the URL into an R object
Macbeth_raw <- RCurl::getURL(macbeth_url)


## loading the 'Macbeth_raw' data frame, which is actually part of the 'mdsr' package:
data(Macbeth_raw)

length(Macbeth_raw)

nchar(Macbeth_raw)

# 'Macbeth_raw' is actually just one VERY long character string

## str_split will split the long character string into a vector of many character
strings
## We split at the end-of-line characters \r and \n

# str_split returns a list: we only want the first element
macbeth <- Macbeth_raw %>%
  str_split("\r\n") %>%
  pluck(1)  # plucks the first element from the list

length(macbeth)

head(macbeth)  # the first few strings are the publisher's notes...


## picking some consecutive strings from inside the document:
macbeth[300:310]


## Finding the lines where the character MACBETH speaks
## by looking for the subset of strings with "  MACBETH" in them.
## The 'str_subset' function does this:

macbeth_lines <- macbeth %>%
  str_subset("  MACBETH")
length(macbeth_lines)
```

```
head(macbeth_lines)


## Finding the lines where the character MACDUFF speaks
## by looking for the subset of strings with "  MACDUFF" in them.

macbeth %>%
  str_subset("  MACDUFF") %>%
  length()


## str_subset returns the subset of the elements containing the specified string:
macbeth %>%
  str_subset("  MACBETH") %>%
  length()

## str_detect returns a vector (having the same length as the whole large object)
containing
## TRUEs and FALSEs elementwise, depending on whether each element contains the
specified string:
macbeth %>%
  str_detect("  MACBETH") %>%
  length()

# We see the first 6 lines of 'macbeth' do NOT contain the string "  MACBETH":
macbeth %>%
  str_detect("  MACBETH") %>%
  head()

# To find the indices of the elements where "  MACBETH" *does* appear, use str_which:
macbeth %>%
  str_which("  MACBETH")


## 'str_extract' from the 'stringr' package returns the actual matching piece from
each selected element
## having that specified pattern:
pattern <- "  MACBETH"
macbeth %>%
  str_subset(pattern) %>%
  str_extract(pattern) %>%
  head()


## The '.' metacharacter matches any character, so searching for "MAC." will return
strings that start with 'MAC'.
## This includes MACBETH and MACDUFF, but also some unrelated words ...
macbeth %>%
  str_subset("MAC.") %>%
  head(12)

## To actually search for a period (.) character, you must precede the period with two
backslashes:
macbeth %>%
  str_subset("MACBETH\\.") %>%
```

```
  head()


## The [B-Z] stands for any character between B and Z:
macbeth %>%
  str_subset("MAC[B-Z]") %>%
  head()

## Using [D-Z] will exclude MACBETH from the search results:
macbeth %>%
  str_subset("MAC[D-Z]") %>%
  head()


## The (B|D) represents EITHER the B or D characters, so this will find MACBETH or
MACDUFF
macbeth %>%
  str_subset("MAC(B|D)") %>%
  head()



## The ^ searches only for the specified string when it appears at the beginning of
the line of text.
## Note the difference here:

macbeth %>%
  str_subset("^  MAC[B-Z]") %>%
  head()

macbeth %>%
  str_subset(" MAC[B-Z]") %>%
  head()

## The $ searches only for the specified string when it appears at the end of the line
of text.
## Note the difference here:

macbeth %>%
  str_subset("MACBETH$") %>%
  head()

macbeth %>%
  str_subset("MACBETH") %>%
  head()


## The ? searches for instances where the previous element in the pattern (here, a
space) is repeated 0 times or 1 time.
## The * searches for instances where the previous element in the pattern (here, a
space) is repeated 0 or more times.
## The + searches for instances where the previous element in the pattern (here, a
space) is repeated 1 or more times.

macbeth %>%
```

```
  str_subset("^ ?MAC[B-Z]") %>%
  head()
macbeth %>%
  str_subset("^ *MAC[B-Z]") %>%
  head()
macbeth %>%
  str_subset("^ +MAC[B-Z]") %>%
  head()


## Note that all these searches are case-sensitive!
## See the difference in these three results:

macbeth %>%
  str_subset("MACBETH") %>%
  head()

macbeth %>%
  str_subset("Macbeth") %>%
  head()

macbeth %>%
  str_subset("macbeth") %>%
  head()

#############################################################################
#
#  TEXT ANALYSIS OF ARTICLE BY Prof. Hitchcock
#
#############################################################################

# Importing the article from a text file on the web:

DBH_url <- "https://people.stat.sc.edu/hitchcock/HistoryStatisticsCourseTASrev.txt"

DBH_raw <- RCurl::getURL(DBH_url)

length(DBH_raw)

nchar(DBH_raw)


# str_split returns a list: we only want the first element
dbh <- DBH_raw %>%
  str_split("\r\n") %>%
  pluck(1)  # plucks the first element from the list

length(dbh)

head(dbh,25)

## This uses some tools we've mentioned to create a tibble with information about on
which line each section of the article begins...
sections <- tibble(
  line = str_which(dbh, "^[1-9] +"),
```

```
  line_text = str_subset(dbh, "^[1-9] +"),
  labels = str_extract(line_text, "^[1-9] +")
)

# install.packages("tidytext")
library(tidytext)


d <- tibble(txt = dbh)
d
d %>%
  unnest_tokens(output = word, input = txt)

d_clean <- d %>%
  unnest_tokens(output = word, input = txt) %>%
  anti_join(get_stopwords(), by = "word")

# Default word cloud:

wordcloud(d_clean$word)

# Formatted word cloud:

  wordcloud(d_clean$word,
    max.words = 40,
    scale = c(5, 1),
    colors = topo.colors(n = 30),
    random.color = TRUE
  )

## Bigrams and N-grams:

## 4-grams:

## Finding all the 4-grams in the article:
dbh_4grams <- d %>%
  unnest_tokens(output = dbh_4gram, input = txt, token = "ngrams",
    n = 4) %>%
  select(dbh_4gram)  # only selecting two of the columns
dbh_4grams

## How many times each 4-gram appears, sorted from most to least:
dbh_4grams %>%
  count(dbh_4gram, sort = TRUE)


## Most common words in the whole article:

d_clean %>%
  count(word) %>%
  arrange(desc(n)) %>%
  head()


### Don’t need to worry about Chapter 17 R code for the free-response coding problem!
```