Programming with the tidyverse

Task 1

Part a

We have our data in the appropriate folder. So, we will start by looking into using `read_csv()` to load in our data.

```r
?read_csv
```

We are not able to use this function specifically, because this file is delimited by `;` and not a comma. We are able to use the `read_csv2()` function for semicolon delimited data. So, we will use this to read in our data below.

```r
data <- read_csv2("./data/data.txt",
          col_names = TRUE)
```

```
i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.

Rows: 2 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ";"
dbl (3): x, y, z

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
data
```

```
# A tibble: 2 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
2     5     3     8
```

Part b

Next, we will load in the `6` delimited data with an empty row. This should look the same as the data above when finished loading it in. We will use `read_delim()`.

```
data1 <- read_delim("./data/data2.txt",
                    delim = "6",
                    col_names = TRUE,
                    col_types = "fdc")
data1
```

```
# A tibble: 3 x 3
  x         y z
  <fct> <dbl> <chr>
1 1         2 3
2 5         3 8
3 7         4 2
```

Task 2

Part a

We will first load in the `trailblazer` data and ensure that it has been loaded in properly.

```
trailblazer <- read_csv("./data/trailblazer.csv")
```

```
Rows: 9 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (1): Player
dbl (10): Game1_Home, Game2_Home, Game3_Away, Game4_Home, Game5_Home, Game6_...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(trailblazer)
```

```
Rows: 9
Columns: 11
$ Player      <chr> "Damian Lillard", "CJ McCollum", "Norman Powell", "Robert ~
$ Game1_Home  <dbl> 20, 24, 14, 8, 20, 5, 11, 2, 7
$ Game2_Home  <dbl> 19, 28, 16, 6, 9, 5, 18, 8, 11
$ Game3_Away  <dbl> 12, 20, NA, 0, 4, 8, 12, 5, 5
$ Game4_Home  <dbl> 20, 25, NA, 3, 17, 10, 17, 8, 9
$ Game5_Home  <dbl> 25, 14, 12, 9, 14, 9, 5, 3, 8
$ Game6_Away  <dbl> 14, 25, 14, 6, 13, 6, 19, 8, 8
$ Game7_Away  <dbl> 20, 20, 22, 0, 7, 0, 17, 7, 4
$ Game8_Away  <dbl> 26, 21, 23, 6, 6, 7, 15, 0, 0
$ Game9_Home  <dbl> 4, 27, 25, 19, 10, 0, 16, 2, 7
$ Game10_Home <dbl> 25, 7, 13, 12, 15, 6, 10, 4, 8
```

This looks to be read in correctly.

Part b

Now, we will pivot the data to have `Player`, `Game`, `Location`, and `Points`.

```r
library(tidyr)
trailblazer_longer <- trailblazer |>
  pivot_longer(cols = 2:11,
               names_to = c("Game", "Location"),
               names_sep = "_",
               values_to = "Points")
trailblazer_longer |> head(5)
```

```
# A tibble: 5 x 4
  Player         Game  Location Points
  <chr>          <chr> <chr>     <dbl>
1 Damian Lillard Game1 Home         20
2 Damian Lillard Game2 Home         19
3 Damian Lillard Game3 Away         12
4 Damian Lillard Game4 Home         20
5 Damian Lillard Game5 Home         25
```

This data set does have 90 rows and 4 columns. We printed out the first 5 as directed.

Part c

We will now investigate which players scored more, on average, when playing at home versus when playing away. We will first use `pivot_wider()` on our new data set to get columns `Player`, `Game`, `Home`, and `Away`.

```
trailblazer_longer |>
  pivot_wider(names_from = "Location",
              values_from = "Points"
              ) |>
  group_by(Player) |>
  mutate(mean_home = mean(Home, na.rm = TRUE),
         mean_away = mean(Away, na.rm = TRUE),
         diff = mean_home - mean_away) |>
  arrange(desc(diff))
```

```
# A tibble: 90 x 7
# Groups:   Player [9]
   Player       Game    Home  Away mean_home mean_away  diff
   <chr>        <chr>  <dbl> <dbl>     <dbl>     <dbl> <dbl>
 1 Jusuf Nurkic Game1     20    NA      14.2       7.5  6.67
 2 Jusuf Nurkic Game2      9    NA      14.2       7.5  6.67
 3 Jusuf Nurkic Game3     NA     4      14.2       7.5  6.67
 4 Jusuf Nurkic Game4     17    NA      14.2       7.5  6.67
 5 Jusuf Nurkic Game5     14    NA      14.2       7.5  6.67
 6 Jusuf Nurkic Game6     NA    13      14.2       7.5  6.67
 7 Jusuf Nurkic Game7     NA     7      14.2       7.5  6.67
 8 Jusuf Nurkic Game8     NA     6      14.2       7.5  6.67
 9 Jusuf Nurkic Game9     10    NA      14.2       7.5  6.67
10 Jusuf Nurkic Game10    15    NA      14.2       7.5  6.67
# i 80 more rows
```

In the first 10 games of the season, Jusuf Nurkic, Robert Covington, Nassir Little, Damian Lillard, and Cody Zeller all scored more points, on average, for home games than away games.

Task 3

Part a

We will now load in the penguins data set for use for the next tasks.

```
library(palmerpenguins)
```

Next, we will look at the provided code attempting to pivot the data into a wider format.

```
penguins |>
select(species, island, bill_length_mm) |>
pivot_wider(
names_from = island, values_from = bill_length_mm
)
```

```
Warning: Values from `bill_length_mm` are not uniquely identified; output will contain
list-cols.
* Use `values_fn = list` to suppress this warning.
* Use `values_fn = {summary_fun}` to summarise duplicates.
* Use the following dplyr code to identify duplicates.
  {data} |>
  dplyr::summarise(n = dplyr::n(), .by = c(species, island)) |>
  dplyr::filter(n > 1L)
```

```
# A tibble: 3 x 4
  species   Torgersen  Biscoe      Dream
  <fct>     <list>     <list>      <list>
1 Adelie    <dbl [52]> <dbl [44]>  <dbl [56]>
2 Gentoo    <NULL>     <dbl [124]> <NULL>
3 Chinstrap <NULL>     <NULL>      <dbl [68]>
```

This is happening due to the structure of the data. We are asking for the columns to come from the `island` variable, but there are multiple bill lengths for each species within that island. This means that the new entries are going to be lists with all of the specific values within it. For instance, there is a list of 52 doubles for the Adelie species on the Torgersen island (so, all of the 52 values got put into one entry as a list).m There are also null values, which most likely indicated that there were none of that species on that island measured for bill length.

Part b

We will now try to recreate the second table provided in the file using the `penguins` data set. From the insight above, we know that we are trying to represent the count of the bird of each species within each island. The `count` function from `dplyr` will help with this.

```
penguins |>
  group_by(species, island) |>
  summarise(count = n()) |>
  pivot_wider(names_from = island,
              values_from = count,
              values_fill = 0)
```

```
`summarise()` has grouped output by 'species'. You can override using the
`.groups` argument.
```

```
# A tibble: 3 x 4
# Groups:   species [3]
  species   Biscoe Dream Torgersen
  <fct>      <int> <int>     <int>
1 Adelie        44    56        52
2 Chinstrap      0    68         0
3 Gentoo       124     0         0
```

This output provides the table desired.

Task 4

Now, we know the `NA` values from the `penguins` data set. We will use one pipeline to add those values and print the first 10 observations of the tibble in ascending order to ensure that this worked as desired.

```
penguins |>
  mutate(bill_length_mm = case_when(
    is.na(bill_length_mm) & species == "Adelie" ~ 26,
    is.na(bill_length_mm) & species == "Gentoo" ~ 30,
    TRUE ~ bill_length_mm))|>
  arrange(bill_length_mm) |>
  slice_head(n = 10)
```

```
# A tibble: 10 x 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           26            NA                  NA          NA
 2 Gentoo  Biscoe              30            NA                  NA          NA
 3 Adelie  Dream               32.1          15.5               188        3050
 4 Adelie  Dream               33.1          16.1               178        2900
 5 Adelie  Torgersen           33.5          19                 190        3600
 6 Adelie  Dream               34            17.1               185        3400
 7 Adelie  Torgersen           34.1          18.1               193        3475
 8 Adelie  Torgersen           34.4          18.4               184        3325
 9 Adelie  Biscoe              34.5          18.1               187        2900
10 Adelie  Torgersen           34.6          21.1               198        4400
# i 2 more variables: sex <fct>, year <int>
```

This arrangement and printed output shows that the NA values were correctly changed.