011852253 Assignment 3

SHARATH KUMAR KARNATI

2024-09-21

#QUESTION 1:

(50 pts total) For this question you will be using either the dplyr package from R or the Pandas library in Python to manipulate and clean up a dataset called NBA_Stats_23_24.csv (available in the Modules section on Canvas under the folder Datasets for Assignments). This data was pulled from https://www.nba.com/stats website.

The dataset contains information about the Men's National Basketball Association games in 2023 - 2024. It has 735 rows and 30 variables. Here is a description of the variables:

Variable Description Rk Rank Player Player's name Pos Position Age Player's age Tm Team G Games played GS Games started MP Minutes played per game FG Field goals per game FGA Field goal attempts per game FG% Field goal percentage 3P 3-point field goals per game 3PA 3-point field goal attempts per game 3P% 3-point field goal percentage 2P 2-point field goals per game 2PA 2-point field goal attempts per game 2P% 2-point field goal percentage eFG% Effective field goal percentage FT Free throws per game FTA Free throw attempts per game FT% Free throw percentage ORB Offensive rebounds per game DRB Defensive rebounds per game TRB Total rebounds per game AST Assists per game STL Steals per game BLK Blocks per game TOV Turnovers per game PF Personal fouls per game PTS Points per game

Load the data into R or Python, and check for missing values (NaN). All the tasks in this assignment can be hand coded, but the goal is to use the functions built into dplyr or Pandas to complete the tasks. Suggested functions for Python are shown in blue while suggested R functions are shown in red. Note: if you are using Python, be sure to load the data as a Pandas DataFrame. Below are the tasks to perform. Before you begin, print the first few values of the columns with a header containing the string "FG". (head(), head())

```
# Load necessary library
suppressMessages(library(dplyr))

# Load the dataset
nba_data <- read.csv('/Users/sharathkarnati/Desktop/DS/SK assignment3/NBA_Stats_23_24.csv',header = TRU

# Select columns containing "FG" in the header and print the first few rows
nba_data %>%
  select(contains("FG")) %>%
  head(10)
```

```
##     FG  FGA   FG.  eFG.
## 1  3.2  6.3 0.501 0.529
## 2  3.1  6.8 0.459 0.497
## 3  3.2  6.1 0.525 0.547
## 4  7.5 14.3 0.521 0.529
## 5  2.3  5.6 0.411 0.483
## 6  2.1  4.9 0.426 0.520
## 7  2.7  6.8 0.391 0.432
## 8  4.0  9.3 0.435 0.528
```

```
## 9  2.9  6.6 0.439 0.560
## 10 4.5  9.1 0.499 0.649
```

#QUESTION 1.a:

a) (5 pts) Count the number of players with Free Throws per game greater than 0.5 and Assists per game greater than 0.7. (filter(), query())

```
# Filter and count players with Free Throws per game > 0.5 and Assists per game > 0.7
nba_data %>%
  filter(FT > 0.5, AST > 0.7) %>%
  summarise(count = n())
```

```
##   count
## 1   405
```

#QUESTION 1.b b) (10 pts) Print the Player, Team, Field goals per game, Turnovers per game, and Points per game of the players with the 10 highest points, in descending order of points. (select(), arrange(), loc(), sort_values()). Which player has the seventh highest points?

```
# Load necessary library
library(dplyr)

# Select relevant columns and filter the top 10 players with the highest points
top_10_players <- nba_data %>%
  select(Player, Tm, FG, TOV, PTS) %>%
  arrange(desc(PTS)) %>%
  head(10)

# Print the result
print(top_10_players)
```

```
##                    Player  Tm   FG TOV  PTS
## 1             Joel Embiid PHI 11.5 3.8 34.7
## 2             Luka Don?i? DAL 11.5 4.0 33.9
## 3    Giannis Antetokounmpo MIL 11.5 3.4 30.4
## 4  Shai Gilgeous-Alexander OKC 10.6 2.2 30.1
## 5           Jalen Brunson NYK 10.3 2.4 28.7
## 6            Devin Booker PHO  9.4 2.6 27.1
## 7            Kevin Durant PHO 10.0 3.3 27.1
## 8            Jayson Tatum BOS  9.1 2.5 26.9
## 9            De'Aaron Fox SAC  9.7 2.6 26.6
## 10       Donovan Mitchell CLE  9.1 2.8 26.6
```

```
# Identify the player with the 7th highest points
seventh_highest_player <- top_10_players %>%
  slice(7) %>%
  select(Player)

print(seventh_highest_player)
```

```
##         Player
## 1 Kevin Durant
```

Answer: **KEVIN DURANT** has the seventh highest points from the data given.

#QUESTION 1.c c) (10 pts) Add two new columns to the dataframe: FGP (in percentage) is the ratio of FG to FGA, FTP (in percentage) is the ratio of FT to FTA. Note that the unit should be expressed in

percentage (ranging from 0 to 100) and rounded to 2 decimal places (e.g., for Jamal Cain, FGP is 43.33) (mutate(), assign()). What is the FGP and FTP for Josh Giddey?

```
# Load necessary library
library(dplyr)

# Add two new columns: FGP (Field Goal Percentage) and FTP (Free Throw Percentage)
nba_data <- nba_data %>%
  mutate(
    FGP = round((FG / FGA) * 100, 2),  # Calculate FGP as FG/FGA * 100 and round to 2 decimal places
    FTP = round((FT / FTA) * 100, 2)   # Calculate FTP as FT/FTA * 100 and round to 2 decimal places
  )
nba_data %>%
head(5)
```

```
##   Rk           Player  Pos Age  Tm  G GS   MP  FG  FGA   FG. X3P X3PA  X3P. X2P
## 1  1 Precious Achiuwa PF-C  24 TOT 74 18 21.9 3.2  6.3 0.501 0.4  1.3 0.268 2.8
## 2  1 Precious Achiuwa    C  24 TOR 25  0 17.5 3.1  6.8 0.459 0.5  1.9 0.277 2.6
## 3  1 Precious Achiuwa   PF  24 NYK 49 18 24.2 3.2  6.1 0.525 0.3  1.0 0.260 2.9
## 4  2      Bam Adebayo    C  26 MIA 71 71 34.0 7.5 14.3 0.521 0.2  0.6 0.357 7.3
## 5  3     Ochai Agbaji   SG  23 TOT 78 28 21.0 2.3  5.6 0.411 0.8  2.7 0.294 1.5
##   X2PA  X2P.  eFG.  FT FTA   FT. ORB DRB  TRB AST STL BLK TOV  PF  PTS   FGP
## 1  5.0 0.562 0.529 0.9 1.5 0.616 2.6 4.0  6.6 1.3 0.6 0.9 1.1 1.9  7.6 50.79
## 2  4.9 0.528 0.497 1.0 1.7 0.571 2.0 3.4  5.4 1.8 0.6 0.5 1.2 1.6  7.7 45.59
## 3  5.1 0.578 0.547 0.9 1.4 0.643 2.9 4.3  7.2 1.1 0.6 1.1 1.1 2.1  7.6 52.46
## 4 13.7 0.528 0.529 4.1 5.5 0.755 2.2 8.1 10.4 3.9 1.1 0.9 2.3 2.2 19.3 52.45
## 5  2.8 0.523 0.483 0.5 0.7 0.661 0.9 1.8  2.8 1.1 0.6 0.6 0.8 1.5  5.8 41.07
##     FTP
## 1 60.00
## 2 58.82
## 3 64.29
## 4 74.55
## 5 71.43
```

```
# Print the FGP and FTP for Josh Giddey
josh_giddey_stats <- nba_data %>%
  filter(Player == "Josh Giddey") %>%
  select(Player, FGP, FTP)

print(josh_giddey_stats)
```

```
##        Player   FGP   FTP
## 1 Josh Giddey 47.17 81.25
```

Answer: **JOSH GIDDEY** has FGP of **47.17%** and FTP of **81.25%**

#QUESTION 1.d

d) (10 pts) Display the average, min and max Offensive rebounds per game for each team, in descending order of the team average. (group_by(), summarise(), groupby(), agg()). You can exclude NAs for this calculation. Which team has the max Offensive rebounds per game?

```
# Load necessary library
library(dplyr)

# Group by team and calculate the average, min, and max for Offensive rebounds (ORB) per game
team_orb_stats <- nba_data %>%
  filter(!is.na(ORB)) %>%  # Exclude NAs in ORB
```

```
  group_by(Tm) %>%
  summarise(
    avg_ORB = mean(ORB, na.rm = TRUE),  # Calculate average ORB
    min_ORB = min(ORB, na.rm = TRUE),   # Calculate min ORB
    max_ORB = max(ORB, na.rm = TRUE)    # Calculate max ORB
  ) %>%
  arrange(desc(avg_ORB))  # Order by average ORB in descending order

# Print the result
print(team_orb_stats)
```

```
## # A tibble: 31 x 4
##    Tm    avg_ORB min_ORB max_ORB
##    <chr>   <dbl>   <dbl>   <dbl>
##  1 UTA     1.1       0.2     2.6
##  2 POR     1.09      0       3.2
##  3 MEM     1.07      0       2.8
##  4 ATL     1.02      0.1     4.6
##  5 CHI     1         0       3.4
##  6 HOU     1         0.1     2.9
##  7 TOR     0.98      0       2.9
##  8 BRK     0.933     0       2.7
##  9 GSW     0.906     0       2
## 10 BOS     0.889     0       1.9
## # i 21 more rows
```

```
# Find the team with the maximum offensive rebounds per game
team_max_orb <- team_orb_stats %>%
  filter(max_ORB == max(team_orb_stats$max_ORB))

print(team_max_orb)
```

```
## # A tibble: 2 x 4
##   Tm    avg_ORB min_ORB max_ORB
##   <chr>   <dbl>   <dbl>   <dbl>
## 1 ATL     1.02      0.1     4.6
## 2 NYK     0.827     0       4.6
```

Answer: **ATL and NYK** has the maximum offensive rebounds per game.

#QUESTION 1.e e) (15 pts) In question 1c, you added a new column called FTP. Impute the missing (or NaN) FTP values as the FGP (also added in 1c) multiplied by the average FTP for that team. Make a second copy of your dataframe, but this time impute missing (or NaN) FTP values with just the average FTP for that team. What assumptions do these data filling methods make? Which is the best way to impute the data, or do you see a better way, and why? You may impute or remove other variables as you find appropriate. Briefly explain your decisions. (group_by(), mutate(), groupby(),assign())

```
# Load necessary library
library(dplyr)

# Make a copy of the original dataframe for Method 1
nba_data_method1 <- nba_data

# Imputation Method 1: Impute missing FTP as FGP * team's average FTP
nba_data_method1 <- nba_data_method1 %>%
  group_by(Tm) %>%
```

```
  mutate(
    avg_FTP = mean(FTP, na.rm = TRUE),  # Calculate team's average FTP
    FTP = ifelse(is.na(FTP), FGP * avg_FTP / 100, FTP)  # Impute missing FTP values
  ) %>%
  ungroup()  # Ungroup to avoid grouping affecting further operations

nba_data_method1 %>%
  head(10)
```

```
## # A tibble: 10 x 33
##       Rk Player      Pos     Age Tm        G    GS    MP    FG   FGA   FG.   X3P
##    <int> <chr>       <chr> <int> <chr> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1 Precious A~ PF-C     24 TOT      74    18  21.9   3.2   6.3 0.501   0.4
##  2     1 Precious A~ C        24 TOR      25     0  17.5   3.1   6.8 0.459   0.5
##  3     1 Precious A~ PF       24 NYK      49    18  24.2   3.2   6.1 0.525   0.3
##  4     2 Bam Adebayo C        26 MIA      71    71  34     7.5  14.3 0.521   0.2
##  5     3 Ochai Agba~ SG       23 TOT      78    28  21     2.3   5.6 0.411   0.8
##  6     3 Ochai Agba~ SG       23 UTA      51    10  19.7   2.1   4.9 0.426   0.9
##  7     3 Ochai Agba~ SG       23 TOR      27    18  23.6   2.7   6.8 0.391   0.6
##  8     4 Santi Alda~ PF       23 MEM      61    35  26.5   4     9.3 0.435   1.7
##  9     5 Nickeil Al~ SG       25 MIN      82    20  23.4   2.9   6.6 0.439   1.6
## 10     6 Grayson Al~ SG       28 PHO      75    74  33.5   4.5   9.1 0.499   2.7
## # i 21 more variables: X3PA <dbl>, X3P. <dbl>, X2P <dbl>, X2PA <dbl>,
## #   X2P. <dbl>, eFG. <dbl>, FT <dbl>, FTA <dbl>, FT. <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, FGP <dbl>, FTP <dbl>, avg_FTP <dbl>
```

```
# Make a copy of the original dataframe for Method 2
nba_data_method2 <- nba_data

# Imputation Method 2: Impute missing FTP with just the team's average FTP
nba_data_method2 <- nba_data_method2 %>%
  group_by(Tm) %>%
  mutate(
    avg_FTP = mean(FTP, na.rm = TRUE),  # Calculate team's average FTP
    FTP = ifelse(is.na(FTP), avg_FTP, FTP)  # Impute missing FTP with the average FTP
  ) %>%
  ungroup()  # Ungroup to avoid grouping affecting further operations

# Print to check the imputed values for both methods
print(nba_data_method1)
```

```
## # A tibble: 735 x 33
##       Rk Player      Pos     Age Tm        G    GS    MP    FG   FGA   FG.   X3P
##    <int> <chr>       <chr> <int> <chr> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1 Precious A~ PF-C     24 TOT      74    18  21.9   3.2   6.3 0.501   0.4
##  2     1 Precious A~ C        24 TOR      25     0  17.5   3.1   6.8 0.459   0.5
##  3     1 Precious A~ PF       24 NYK      49    18  24.2   3.2   6.1 0.525   0.3
##  4     2 Bam Adebayo C        26 MIA      71    71  34     7.5  14.3 0.521   0.2
##  5     3 Ochai Agba~ SG       23 TOT      78    28  21     2.3   5.6 0.411   0.8
##  6     3 Ochai Agba~ SG       23 UTA      51    10  19.7   2.1   4.9 0.426   0.9
##  7     3 Ochai Agba~ SG       23 TOR      27    18  23.6   2.7   6.8 0.391   0.6
##  8     4 Santi Alda~ PF       23 MEM      61    35  26.5   4     9.3 0.435   1.7
##  9     5 Nickeil Al~ SG       25 MIN      82    20  23.4   2.9   6.6 0.439   1.6
## 10     6 Grayson Al~ SG       28 PHO      75    74  33.5   4.5   9.1 0.499   2.7
```

```
## # i 725 more rows
## # i 21 more variables: X3PA <dbl>, X3P. <dbl>, X2P <dbl>, X2PA <dbl>,
## #   X2P. <dbl>, eFG. <dbl>, FT <dbl>, FTA <dbl>, FT. <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, FGP <dbl>, FTP <dbl>, avg_FTP <dbl>
```

```
print(nba_data_method2)
```

```
## # A tibble: 735 x 33
##       Rk Player      Pos     Age Tm        G    GS    MP    FG   FGA   FG.   X3P
##    <int> <chr>       <chr> <int> <chr> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1 Precious A~ PF-C     24 TOT      74    18  21.9   3.2   6.3 0.501   0.4
##  2     1 Precious A~ C        24 TOR      25     0  17.5   3.1   6.8 0.459   0.5
##  3     1 Precious A~ PF       24 NYK      49    18  24.2   3.2   6.1 0.525   0.3
##  4     2 Bam Adebayo C        26 MIA      71    71  34     7.5  14.3 0.521   0.2
##  5     3 Ochai Agba~ SG       23 TOT      78    28  21     2.3   5.6 0.411   0.8
##  6     3 Ochai Agba~ SG       23 UTA      51    10  19.7   2.1   4.9 0.426   0.9
##  7     3 Ochai Agba~ SG       23 TOR      27    18  23.6   2.7   6.8 0.391   0.6
##  8     4 Santi Alda~ PF       23 MEM      61    35  26.5   4     9.3 0.435   1.7
##  9     5 Nickeil Al~ SG       25 MIN      82    20  23.4   2.9   6.6 0.439   1.6
## 10     6 Grayson Al~ SG       28 PHO      75    74  33.5   4.5   9.1 0.499   2.7
## # i 725 more rows
## # i 21 more variables: X3PA <dbl>, X3P. <dbl>, X2P <dbl>, X2PA <dbl>,
## #   X2P. <dbl>, eFG. <dbl>, FT <dbl>, FTA <dbl>, FT. <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, FGP <dbl>, FTP <dbl>, avg_FTP <dbl>
```

or, by imputing the unnecessary data from table : the code looks like :

```
# Load necessary library
library(dplyr)

# Original dataset (assuming `nba_data` is already loaded)

# Imputation Method 1: FGP * avg_FTP for the team
nba_data_method1 <- nba_data %>%
  filter(!is.na(FGP)) %>%   # Ensure FGP is not NA for multiplication
  group_by(Tm) %>%
  mutate(
    avg_FTP = mean(FTP, na.rm = TRUE),  # Calculate team average FTP
    FTP = ifelse(is.na(FTP), round(FGP * avg_FTP / 100, 2), FTP)  # Impute missing FTP
  ) %>%
  ungroup() %>%
  select(-avg_FTP)  # Remove the temporary avg_FTP column

# Imputation Method 2: Impute missing FTP with the team's average FTP
nba_data_method2 <- nba_data %>%
  group_by(Tm) %>%
  mutate(
    avg_FTP = mean(FTP, na.rm = TRUE),  # Calculate team average FTP
    FTP = ifelse(is.na(FTP), round(avg_FTP, 2), FTP)  # Impute missing FTP with team average FTP
  ) %>%
  ungroup() %>%
  select(-avg_FTP)  # Remove the temporary avg_FTP column

# Remove unnecessary rows or data that can't be meaningfully imputed (if FTP or FGP are NA)
```

```r
# You can adjust this part based on specific needs.
nba_data_clean_method1 <- nba_data_method1 %>%
  filter(!is.na(FTP), !is.na(FGP))  # Ensure no NA remains in FTP or FGP after imputation

nba_data_clean_method2 <- nba_data_method2 %>%
  filter(!is.na(FTP), !is.na(FGP))  # Ensure no NA remains in FTP or FGP after imputation

# Print the resulting datasets
cat("Method 1 - Imputed FTP as FGP * Team's avg FTP:\n")
```

```
## Method 1 - Imputed FTP as FGP * Team's avg FTP:
```

```r
print(nba_data_clean_method1)
```

```
## # A tibble: 727 x 32
##       Rk Player      Pos     Age Tm        G    GS    MP    FG   FGA   FG.   X3P
##    <int> <chr>       <chr> <int> <chr> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1 Precious A~ PF-C     24 TOT      74    18  21.9   3.2   6.3 0.501   0.4
##  2     1 Precious A~ C        24 TOR      25     0  17.5   3.1   6.8 0.459   0.5
##  3     1 Precious A~ PF       24 NYK      49    18  24.2   3.2   6.1 0.525   0.3
##  4     2 Bam Adebayo C        26 MIA      71    71  34     7.5  14.3 0.521   0.2
##  5     3 Ochai Agba~ SG       23 TOT      78    28  21     2.3   5.6 0.411   0.8
##  6     3 Ochai Agba~ SG       23 UTA      51    10  19.7   2.1   4.9 0.426   0.9
##  7     3 Ochai Agba~ SG       23 TOR      27    18  23.6   2.7   6.8 0.391   0.6
##  8     4 Santi Alda~ PF       23 MEM      61    35  26.5   4     9.3 0.435   1.7
##  9     5 Nickeil Al~ SG       25 MIN      82    20  23.4   2.9   6.6 0.439   1.6
## 10     6 Grayson Al~ SG       28 PHO      75    74  33.5   4.5   9.1 0.499   2.7
## # i 717 more rows
## # i 20 more variables: X3PA <dbl>, X3P. <dbl>, X2P <dbl>, X2PA <dbl>,
## #   X2P. <dbl>, eFG. <dbl>, FT <dbl>, FTA <dbl>, FT. <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, FGP <dbl>, FTP <dbl>
```

```r
cat("\nMethod 2 - Imputed FTP as Team's avg FTP:\n")
```

```
##
## Method 2 - Imputed FTP as Team's avg FTP:
```

```r
print(nba_data_clean_method2)
```

```
## # A tibble: 727 x 32
##       Rk Player      Pos     Age Tm        G    GS    MP    FG   FGA   FG.   X3P
##    <int> <chr>       <chr> <int> <chr> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1 Precious A~ PF-C     24 TOT      74    18  21.9   3.2   6.3 0.501   0.4
##  2     1 Precious A~ C        24 TOR      25     0  17.5   3.1   6.8 0.459   0.5
##  3     1 Precious A~ PF       24 NYK      49    18  24.2   3.2   6.1 0.525   0.3
##  4     2 Bam Adebayo C        26 MIA      71    71  34     7.5  14.3 0.521   0.2
##  5     3 Ochai Agba~ SG       23 TOT      78    28  21     2.3   5.6 0.411   0.8
##  6     3 Ochai Agba~ SG       23 UTA      51    10  19.7   2.1   4.9 0.426   0.9
##  7     3 Ochai Agba~ SG       23 TOR      27    18  23.6   2.7   6.8 0.391   0.6
##  8     4 Santi Alda~ PF       23 MEM      61    35  26.5   4     9.3 0.435   1.7
##  9     5 Nickeil Al~ SG       25 MIN      82    20  23.4   2.9   6.6 0.439   1.6
## 10     6 Grayson Al~ SG       28 PHO      75    74  33.5   4.5   9.1 0.499   2.7
## # i 717 more rows
## # i 20 more variables: X3PA <dbl>, X3P. <dbl>, X2P <dbl>, X2PA <dbl>,
## #   X2P. <dbl>, eFG. <dbl>, FT <dbl>, FTA <dbl>, FT. <dbl>, ORB <dbl>,
```

```
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, FGP <dbl>, FTP <dbl>
```

#ANSWER:

Assumptions of Each Method:

- **Missing values of FTP = FGP * average of FTP**: This method assumes a relationship between FTP and FGP, meaning that the missing FTP values can be estimated based on the player's FGP and the team's average FTP. It implies that a player's free throw percentage is somehow linked to their field goal performance, which might not always be true.
- **Missing values of FTP = average of FTP**: This method replaces missing FTP values with the team's average FTP, assuming that all players on the team have similar FTPs. It doesn't consider any connection between FTP and FGP, making it a more straightforward approach without relying on additional variable relationships.

#Better Approach:

A simpler and more reliable way to impute missing values is by using the second approach—replacing the missing values with the team's average FTP. This method minimizes assumptions about how different variables are related, and it ensures that the overall team statistics remain stable. By using the team's average FTP, we avoid making drastic changes to the mean, which could occur if we imputed based on a potentially unrelated statistic like FGP. This approach maintains the balance and integrity of the dataset without overcomplicating the imputation process.

#QUESTION 2:

(50 pts total) For this question, you will first need to read section 5.3.1 in the R for Data Science book (https://r4ds.hadley.nz/data-tidy#sec-billboard). Grab the dataset "billboard" from the tidyr package (tidyr::billboard), and tidy it as shown in the case study before answering the following questions. The dataset is also available on the Modules page under Datasets for 3 Assignments on Canvas. Note: if you are using Pandas you can perform these same operations by just replacing the pivot_longer() function with melt() and the pivot_wider() function with pivot().

```r
# Load necessary libraries
library(dplyr)
library(tidyr)
library(readr)

# Set the path to your downloaded file (replace with your file path)
file_path <- "/Users/sharathkarnati/Desktop/DS/SK assignment3/billboard.csv"

# Read the CSV file into R
billboard <- read.csv(file_path)

# Inspect the first few rows to verify the dataset
head(billboard)
```

```
##         artist                    track date.entered wk1 wk2 wk3 wk4 wk5 wk6 wk7
## 1        2 Pac Baby Don't Cry (Keep...   2000-02-26  87  82  72  77  87  94  99
## 2      2Ge+her The Hardest Part Of ...   2000-09-02  91  87  92  NA  NA  NA  NA
## 3 3 Doors Down              Kryptonite   2000-04-08  81  70  68  67  66  57  54
## 4 3 Doors Down                   Loser   2000-10-21  76  76  72  69  67  65  55
## 5     504 Boyz           Wobble Wobble   2000-04-15  57  34  25  17  17  31  36
## 6         98^0 Give Me Just One Nig...   2000-08-19  51  39  34  26  26  19   2
##   wk8 wk9 wk10 wk11 wk12 wk13 wk14 wk15 wk16 wk17 wk18 wk19 wk20 wk21 wk22 wk23
```

```
## 1  NA  NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 2  NA  NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 3  53  51   51   51   51   47   44   38   28   22   18   18   14   12    7    6
## 4  59  62   61   61   59   61   66   72   76   75   67   73   70   NA   NA   NA
## 5  49  53   57   64   70   75   76   78   85   92   96   NA   NA   NA   NA   NA
## 6   2   3    6    7   22   29   36   47   67   66   84   93   94   NA   NA   NA
##   wk24 wk25 wk26 wk27 wk28 wk29 wk30 wk31 wk32 wk33 wk34 wk35 wk36 wk37 wk38
## 1   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 2   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 3    6    6    5    5    4    4    4    4    3    3    3    4    5    5    9
## 4   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 5   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 6   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
##   wk39 wk40 wk41 wk42 wk43 wk44 wk45 wk46 wk47 wk48 wk49 wk50 wk51 wk52 wk53
## 1   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 2   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 3    9   15   14   13   14   16   17   21   22   24   28   33   42   42   49
## 4   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 5   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 6   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
##   wk54 wk55 wk56 wk57 wk58 wk59 wk60 wk61 wk62 wk63 wk64 wk65 wk66 wk67 wk68
## 1   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 2   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 3   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 4   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 5   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
## 6   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
##   wk69 wk70 wk71 wk72 wk73 wk74 wk75 wk76
## 1   NA   NA   NA   NA   NA   NA   NA   NA
## 2   NA   NA   NA   NA   NA   NA   NA   NA
## 3   NA   NA   NA   NA   NA   NA   NA   NA
## 4   NA   NA   NA   NA   NA   NA   NA   NA
## 5   NA   NA   NA   NA   NA   NA   NA   NA
## 6   NA   NA   NA   NA   NA   NA   NA   NA
```

```r
# Proceed with the tidying process as described before
billboard_tidy <- billboard %>%
  pivot_longer(cols = starts_with("wk"), names_to = "week", values_to = "rank") %>%
  mutate(week = parse_number(week))  # Extract numeric part of 'week'

# View the tidied dataset
head(billboard_tidy)
```

```
## # A tibble: 6 x 5
##   artist track                   date.entered  week  rank
##   <chr>  <chr>                   <chr>        <dbl> <int>
## 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26       1    87
## 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26       2    82
## 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26       3    72
## 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26       4    77
## 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26       5    87
## 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26       6    94
```

a) (5 pts) Explain why this line # > mutate(week = parse_number(week)) is necessary to properly tidy the data. What happens if you skip this line?

#ANSWER: The line mutate(week = parse_number(week)) is important because it transforms the "week" column from a character string to a numeric value, making it easier to analyze and perform operations on.

#Explanation: Purpose: The parse_number() function extracts numbers from the "week" column (e.g., "wk1" becomes 1, "wk2" becomes 2). Why This Matters: This change enables you to perform calculations, sort the data correctly, and use it in visualizations that require numeric input. #What Happens if You Skip It: Incorrect Sorting: Without converting, sorting would be alphabetical (e.g., "wk10" might appear before "wk2"), which is wrong for week numbers. Analysis Errors: Any mathematical operations (like filtering by week) would not work properly, potentially leading to incorrect results. In short, this line is crucial to ensure that the "week" data is treated correctly as numeric for meaningful analysis.

b) (5 pts) How many entries are removed from the dataset when you set values_drop_na to true in the pivot_longer command (in this dataset)?

```
# Load necessary libraries
library(dplyr)
library(tidyr)

# Set the path to your downloaded file (replace with your file path)
file_path <- "/Users/sharathkarnati/Desktop/DS/SK assignment3/billboard.csv"

# Read the CSV file
billboard <- read.csv(file_path)

# Tidy the dataset without dropping NAs
billboard_tidy <- billboard %>%
  pivot_longer(cols = starts_with("wk"), names_to = "week", values_to = "rank")

# Count the total number of rows without dropping NAs
n_total <- nrow(billboard_tidy)

# Tidy the dataset and drop NAs using values_drop_na = TRUE
billboard_tidy_dropped_na <- billboard %>%
  pivot_longer(cols = starts_with("wk"), names_to = "week", values_to = "rank", values_drop_na = TRUE)

# Count the total number of rows after dropping NAs
n_dropped <- nrow(billboard_tidy_dropped_na)

# Calculate how many rows were removed
entries_removed <- n_total - n_dropped

# Print the result
entries_removed
```

```
## [1] 18785
```

c) (5 pts) Explain the difference between an explicit and implicit missing value, in general. Can you find any implicit missing values in this dataset? If so, where?

```
library(dplyr)
library(tidyr)

# Set the path to your downloaded file (replace with your file path)
file_path <- "/Users/sharathkarnati/Desktop/DS/SK assignment3/billboard.csv"

# Read the CSV file
billboard <- read.csv(file_path)
```

```r
# Tidy the dataset without dropping NAs
billboard_tidy <- billboard %>%
  pivot_longer(cols = starts_with("wk"), names_to = "week", values_to = "rank")

# Convert 'week' to numeric
billboard_tidy <- billboard_tidy %>%
  mutate(week = parse_number(week))

# Identify implicit missing values
# Create a complete dataset with all possible weeks for each song
complete_data <- billboard_tidy %>%
  group_by(artist, track) %>%
  complete(week = full_seq(1:max(week), 1))

# Find implicit missing values by identifying rows with NA ranks
implicit_missing <- complete_data %>%
  filter(is.na(rank))

# Display implicit missing values
implicit_missing
```

```
## # A tibble: 18,785 x 5
## # Groups:   artist, track [317]
##    artist track                    week date.entered  rank
##    <chr>  <chr>                   <dbl> <chr>        <int>
##  1 2 Pac  Baby Don't Cry (Keep...     8 2000-02-26      NA
##  2 2 Pac  Baby Don't Cry (Keep...     9 2000-02-26      NA
##  3 2 Pac  Baby Don't Cry (Keep...    10 2000-02-26      NA
##  4 2 Pac  Baby Don't Cry (Keep...    11 2000-02-26      NA
##  5 2 Pac  Baby Don't Cry (Keep...    12 2000-02-26      NA
##  6 2 Pac  Baby Don't Cry (Keep...    13 2000-02-26      NA
##  7 2 Pac  Baby Don't Cry (Keep...    14 2000-02-26      NA
##  8 2 Pac  Baby Don't Cry (Keep...    15 2000-02-26      NA
##  9 2 Pac  Baby Don't Cry (Keep...    16 2000-02-26      NA
## 10 2 Pac  Baby Don't Cry (Keep...    17 2000-02-26      NA
## # i 18,775 more rows
```

#ANSWER:

- **Explicit Missing Values:** These are clearly marked as missing in the dataset, often shown as `NA`, `null`, or another placeholder. They indicate that a value is absent but recognized. For example, an `NA` entry in a dataset explicitly acknowledges that data is missing.
- **Implicit Missing Values:** These occur when data is expected but not present, and there's no explicit marker indicating their absence. They are inferred from the structure of the dataset. For example, if certain weeks or rows are missing from the dataset without any indication, these are implicit missing values.

In the "billboard" dataset, implicit missing values could occur if a song does not have rank data for some weeks, meaning no rows were entered for those weeks, even though the song continued to chart.

#CONCLUSION:

In the billboard dataset, there are missing values in the rank column when a song doesn't appear on the charts for certain weeks. These gaps aren't marked as NA; instead, they're just implied by the lack of data. For instance, if a song is on the chart for only 5 weeks, it means there are implicitly missing entries for the

other weeks (6 to 76).

d) (5 pts) Looking at the features (artist, track, date.entered, week, rank) in the tidied data, are they all appropriately typed? Are there any features you think would be better suited as a different type? Why or why not?

```
str(billboard_tidy)
```

```
## tibble [24,092 x 5] (S3: tbl_df/tbl/data.frame)
##  $ artist      : chr [1:24092] "2 Pac" "2 Pac" "2 Pac" "2 Pac" ...
##  $ track       : chr [1:24092] "Baby Don't Cry (Keep..." "Baby Don't Cry (Keep..." "Baby Don't Cry (
##  $ date.entered: chr [1:24092] "2000-02-26" "2000-02-26" "2000-02-26" "2000-02-26" ...
##  $ week        : num [1:24092] 1 2 3 4 5 6 7 8 9 10 ...
##  $ rank        : int [1:24092] 87 82 72 77 87 94 99 NA NA NA ...
```

ANSWER:

Let's review the features in the tidied **billboard** dataset:

- **artist**: This is a character vector (`chr`), which is appropriate since it contains the names of the artists.
- **track**: This is also a character vector (`chr`), suitable for storing the names of the tracks.
- **date.entered**: This is a `Date` vector, which is appropriate as it represents the date the track entered the chart.
- **week**: This is an integer vector (`int`), which is correct because it represents the week number.
- **date**: This is a `Date` vector, suitable as it represents the specific date for each week.
- **rank**: This is a numeric vector (`num`), which is appropriate since it stores the ranking of the track.

Suggested Changes:

- **rank**: Since rankings are whole numbers, converting the rank column from numeric (`num`) to an integer (`int`) might improve clarity, as it doesn't need decimal precision.
- **week**: Similarly, if there are any calculations involving the week and rank (e.g., sorting or indexing), having both **week** and **rank** as integers may optimize performance and avoid confusion.

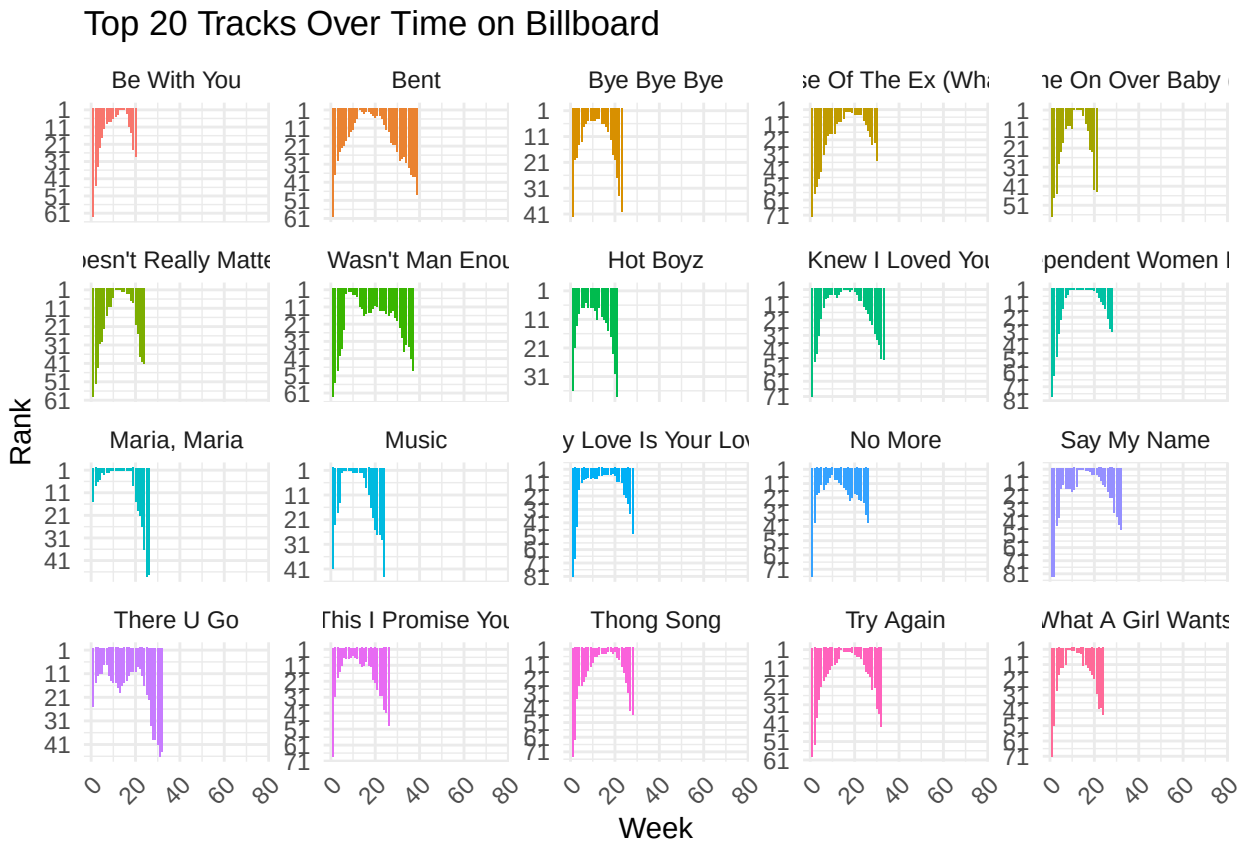
In summary, while the types are mostly appropriate, converting the **rank** to an integer and ensuring **week** remains an integer might be helpful for performance and clarity during calculations.

e) (5 pts) Generate an informative visualization, which shows something about the data. Give a brief description of what it shows, and why you thought it would be interesting to investigate.

```
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Get the top 20 tracks based on their average rank
top_20_tracks <- billboard_tidy %>%
  group_by(artist, track) %>%
  summarise(avg_rank = mean(rank, na.rm = TRUE), .groups = 'drop') %>%
  top_n(-20, avg_rank) # Select top 20 tracks with lowest average rank

# Filter the original data for the top 20 tracks
top_20_tracks_data <- billboard_tidy %>%
  filter(paste(artist, track) %in% paste(top_20_tracks$artist, top_20_tracks$track))

# Create a faceted bar plot to show the rank progression for each track
ggplot(top_20_tracks_data, aes(x = week, y = rank, fill = track)) +
  geom_bar(stat = "identity", position = "dodge") +
```

```
  scale_y_reverse(breaks = seq(1, 100, by = 10)) + # Reverse Y-axis for ranking
  labs(title = "Top 20 Tracks Over Time on Billboard",
       x = "Week",
       y = "Rank") +
  theme_minimal() +
  facet_wrap(~ track, scales = "free_y") + # Facet by track to show each track's progression
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")
```

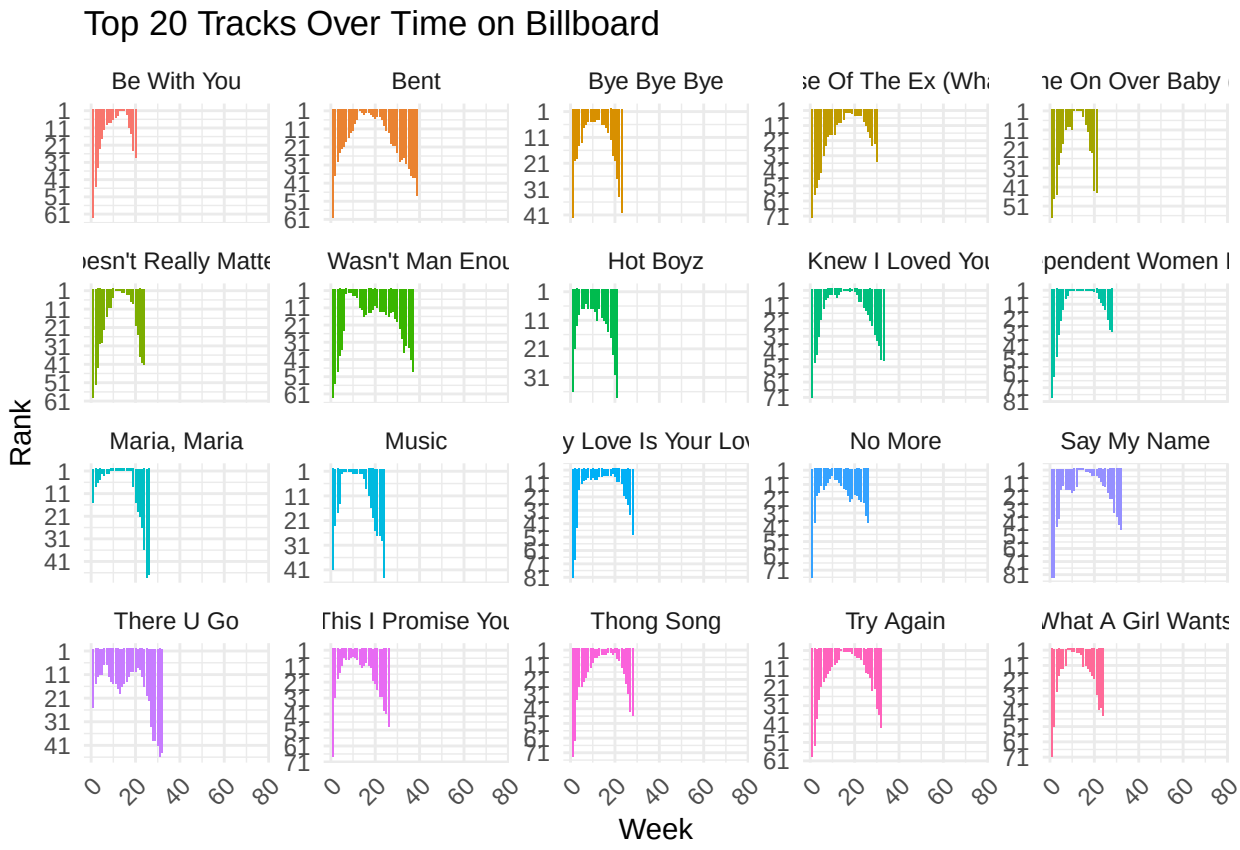
```
## Warning: Removed 966 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```




```
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Get the top 20 tracks based on their average rank
top_20_tracks <- billboard_tidy %>%
  group_by(artist, track) %>%
  summarise(avg_rank = mean(rank, na.rm = TRUE), .groups = 'drop') %>%
  top_n(-20, avg_rank) # Select top 20 tracks with lowest average rank

# Filter the original data for the top 20 tracks
top_20_tracks_data <- billboard_tidy %>%
  filter(paste(artist, track) %in% paste(top_20_tracks$artist, top_20_tracks$track))

# Create a faceted bar plot to show the rank progression for each track
ggplot(top_20_tracks_data, aes(x = week, y = rank, fill = track)) +
```

```
  geom_bar(stat = "identity", position = "dodge") +
  scale_y_reverse(breaks = seq(1, 100, by = 10)) + # Reverse Y-axis for ranking
  labs(title = "Top 20 Tracks Over Time on Billboard",
       x = "Week",
       y = "Rank") +
  theme_minimal() +
  facet_wrap(~ track, scales = "free_y") + # Facet by track to show each track's progression
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")
```

```
## Warning: Removed 966 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```



#ANSWER:

Informative Visualization Description

Visualization: The faceted bar plot displays the rank progression over time for the **top 20 songs** on the Billboard charts, with each song having its own sub-plot. Each bar in the plot represents the rank of the song for a specific week, and the Y-axis is reversed so that higher ranks (closer to 1) appear at the top of each plot.

What it Shows: - The plot helps us see how each song's rank fluctuates over time, with bars providing a visual sense of how long a track stayed at a certain rank, and whether it rose or fell in the rankings. - The **faceted structure** lets us compare the top 20 songs individually, giving insights into each song's performance trajectory without overwhelming the viewer with overlapping lines.

Why It's Interesting: - **Consistency vs. Volatility**: You can easily identify songs that maintained high ranks consistently and compare them to those that fluctuated a lot or dropped off the charts quickly. - **Comparing Performance**: Some songs may have started low but improved over time, while others had strong debuts but quickly dropped in rank. This variation in rank progression is visually clear in the faceted

format. - **Longevity**: Songs with many bars near the top suggest long-term success on the chart, while those with bars concentrated near the bottom indicate brief appearances or poor performance.

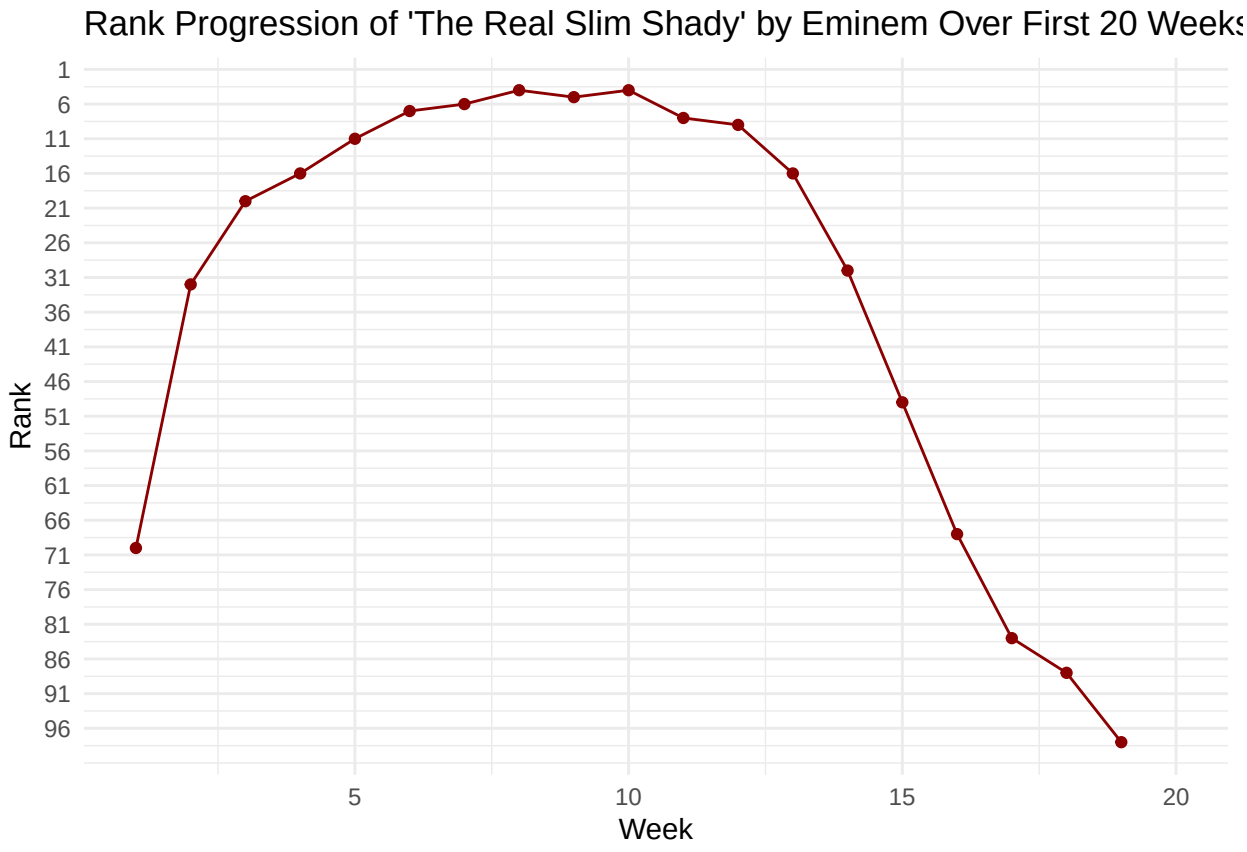
This investigation highlights both the **consistency** and **volatility** in song performance, which could provide insights into trends in music popularity or the dynamics of chart success over time.

f) (5 pts) Generate a line plot showing the rank progression of a specific song over time. You can choose a song you like best from the dataset. (Hint: higher ranks are better so reverse your axis appropriately). Briefly describe what the plot shows.

```
# Filter for the song "The Real Slim Shady" and limit to the first 20 weeks
the_real_slim_shady_data <- billboard_tidy %>%
  filter(track == "The Real Slim Shady", week <= 20)

# Create the plot
ggplot(the_real_slim_shady_data, aes(x = week, y = rank)) +
  geom_line(color = "darkred") +  # Using dark red for the line
  geom_point(color = "darkred") + # Points in dark red
  scale_y_reverse(breaks = seq(1, 100, by = 5)) + # Reverse Y-axis for ranking
  labs(title = "Rank Progression of 'The Real Slim Shady' by Eminem Over First 20 Weeks",
       x = "Week",
       y = "Rank") +
  theme_minimal()
```

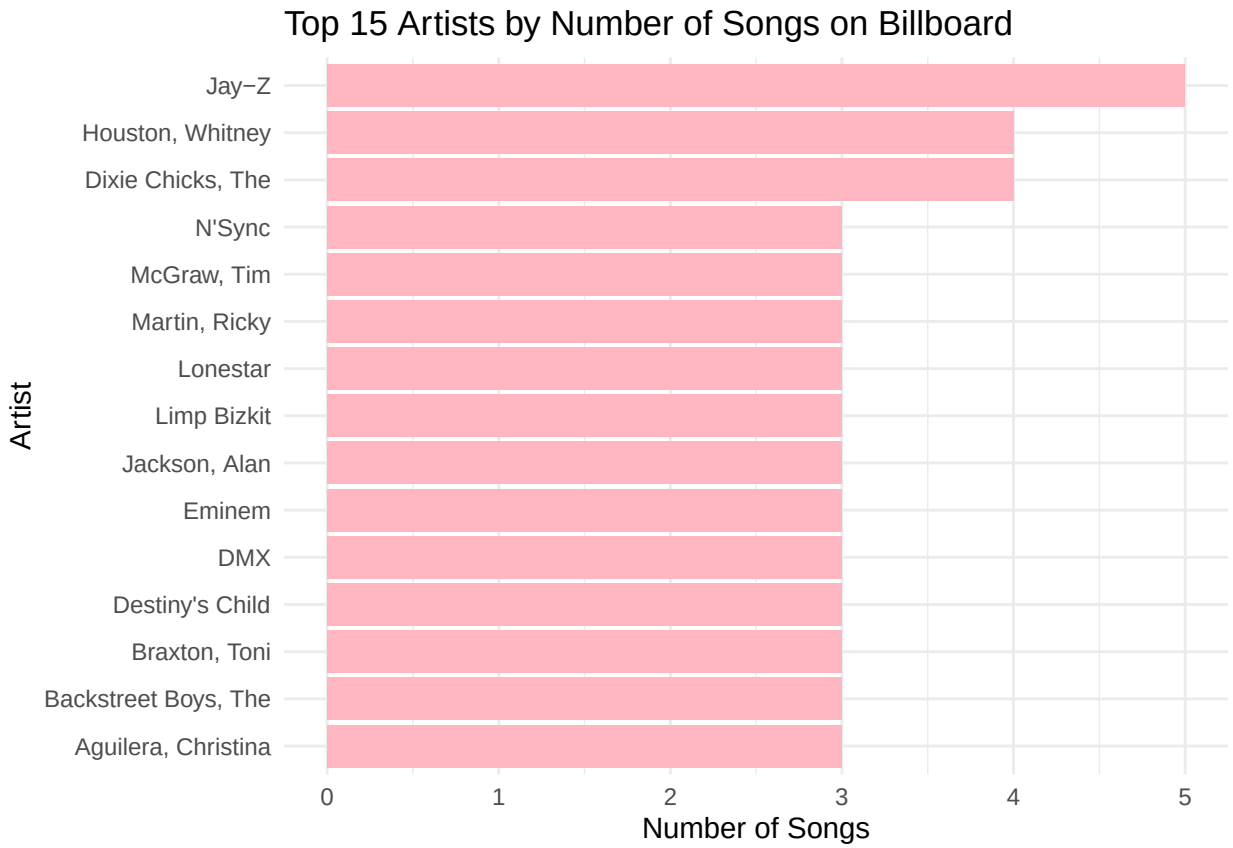
```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



g) (8 pts) Produce a barplot to show the count of songs per artist in the dataset. Limit the plot to the top 15 artists by number of songs. What are your thoughts about this top 15 list? Were you surprised by the presence of any particular artist?

```
# Load necessary libraries
library(ggplot2)
library(dplyr)

# Count the number of songs per artist
artist_song_count <- billboard_tidy %>%
  group_by(artist) %>%
  summarise(song_count = n_distinct(track)) %>%
  arrange(desc(song_count)) %>%
  head(15)  # Limit to top 15 artists

# Create the barplot for the top 15 artists by song count
ggplot(artist_song_count, aes(x = reorder(artist, song_count), y = song_count)) +
  geom_bar(stat = "identity", fill = "lightpink") +
  coord_flip() +  # Flip coordinates for easier readability
  labs(title = "Top 15 Artists by Number of Songs on Billboard",
       x = "Artist",
       y = "Number of Songs") +
  theme_minimal()
```



h) (12 pts) Suppose you have the following dataset called RevQtr (You can download this dataset from the Modules page, under Datasets for Assignments, on Canvas): Group Year Qtr.1 Qtr.2 Qtr_3 Qtr.4 1 2022 61 24 81 70 1 2023 30 92 96 84 1 2024 84 97 33 12 2 2022 31 62 11 97 2 2023 39 47 11 73 2 2024 69 30 42 85 3 2022 67 31 98 58 3 2023 68 51 69 89 3 2024 24 71 71 56 4 2022 71 60 64 73 4 2023 12 60

16 30 4 2024 82 48 27 13 The table consists of 6 columns. The first shows the Group code, the second shows the year and the last four columns provide the revenue for each quarter of the year. Re-structure this table and show the code you would write to tidy the dataset (using gather()/pivot_longer() and separate()/pivot_wider() or melt() and pivot()) such that the columns are organized as: 4 Group, Year, Interval_Type, Interval_ID and Revenue. Note: Here the entire Interval_Type column will contain value 'Qtr' since the dataset provides revenue for every quarter. The Interval_ID will contain the quarter number. Below is an instance of a row of the re-structured table: Group Year Interval_Type Interval_ID Revenue 1 2022 Qtr 1 61 How many rows does the new dataset have?

```
rev_qtr = read.csv("/Users/sharathkarnati/Desktop/DS/SK assignment3/RevQtr.csv", sep=",", header = TRUE)

rev_qtr %>%
  pivot_longer(
    c('Qtr.1', 'Qtr.2','Qtr_3','Qtr.4'),
    names_to = 'Quater',
    values_to = 'Revenue') %>%
  mutate(Quater = stringr::str_replace(Quater, "Qtr_3", "Qtr.3")) %>%
  separate(Quater, into = c("Interval_Type","Interval_ID"))
```

```
## # A tibble: 48 x 5
##    Group  Year Interval_Type Interval_ID Revenue
##    <int> <int> <chr>         <chr>         <int>
##  1     1  2022 Qtr           1                61
##  2     1  2022 Qtr           2                24
##  3     1  2022 Qtr           3                81
##  4     1  2022 Qtr           4                70
##  5     1  2023 Qtr           1                30
##  6     1  2023 Qtr           2                92
##  7     1  2023 Qtr           3                96
##  8     1  2023 Qtr           4                84
##  9     1  2024 Qtr           1                84
## 10     1  2024 Qtr           2                97
## # i 38 more rows
```

#Answer: We can observe that from the new dataset, we have 48 rows.