Final Exam

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

1.

Find the inverse of the following matrix and verify it using the `all.equal()` function.

$$\begin{pmatrix} 9 & 4 & 12 & 2 \\ 5 & 0 & 7 & 9 \\ 2 & 6 & 8 & 0 \\ 9 & 2 & 9 & 11 \end{pmatrix}$$

```r
# Define the matrix
A <- matrix(c(
    9, 5, 2, 9,
    4, 0, 6, 2,
    12, 7, 8, 9,
    2, 9, 0, 11
), nrow = 4, byrow = TRUE)

# Find the inverse
A_inv <- solve(A)

# Verify the inverse by checking if A * A_inv equals identity matrix
identity_check <- A %*% A_inv
identity_matrix <- diag(4)

# Use all.equal() to verify
verification <- all.equal(identity_check, identity_matrix)

A_inv
```

```
##             [,1]       [,2]        [,3]        [,4]
## [1,]  0.08571429 -0.1428571  0.08571429 -0.11428571
## [2,] -0.26000000 -0.3000000  0.29000000  0.03000000
## [3,] -0.12285714  0.1714286  0.02714286  0.04714286
## [4,]  0.19714286  0.2714286 -0.25285714  0.08714286
```

```r
verification
```

```
## [1] TRUE
```

2.

Execute the following lines which create two vectors of random integers which are chosen with replacement from the integers $0, 1, \ldots, 999$. Both vectors have length 250.

```r
xVec <- sample(0:999, 250, replace=T)
yVec <- sample(0:999, 250, replace=T)
```

(a) Create the vector $(y_2 - x_1, \cdots, y_n - x_{n-1})$.

(b) Pick out the values in `yVec` which are > 600.

(c) What are the index positions in `yVec` of the values which are > 600?

(d) Sort the numbers in the vector `xVec` in the order of increasing values in `yVec`.

(e) Pick out the elements in `yVec` at index positions $1, 4, 7, 10, 13, \cdots$.

```r
xVec <- sample(0:999, 250, replace = TRUE)
yVec <- sample(0:999, 250, replace = TRUE)

yVec[-1] - xVec[-length(xVec)]
```

```
##   [1]  482 -231   33  522  598  -37  122  337 -119 -460 -243 -111  191  461  112
##  [16]   16  341  281  494  249  -81  -27  259  288  398 -204  502  616 -498 -514
##  [31]  -35 -539 -583  275 -275 -273   59 -870  682  -61 -751 -183  684  209 -327
##  [46]  261 -274 -221  -66 -279  288 -317  457  580  -77 -249 -170 -369   90  -53
##  [61]  374  191 -144  871  -43   12 -128  587  -86   -1  422  324  143  -68 -235
##  [76]  -98 -252  139  784  310 -772  350  -98 -129 -373 -177  361  312 -591 -224
##  [91] -127  248   34   73 -198  782  596  396 -102 -104 -351  -53   21  390 -494
## [106] -309 -131  183   16   47 -908  -62  569   49  -96   33  -47  181 -231  730
## [121]   -6 -379  753  415 -218   66   86  -87 -525 -315 -476  807 -789  914  156
## [136]  614  794 -310  -19  267  200  750  321  137  815  684  457   86  456 -202
## [151]  200  499  191  -45 -298  763 -377 -525 -315 -396 -140   71  199  -95  217
## [166]  172 -116  310  -22  338  789  258  192 -190 -927   10  849  526  685  118
## [181]   51 -487  538 -694  200  146 -581  798  493 -241  -79 -399    2  727  756
## [196] -504  -58   85  211   40  -89 -354 -678 -170  458 -706 -518  167 -354  849
## [211] -421 -407 -543  359  174  106 -290  -98 -656  492  299   -6 -313 -554  383
## [226] -791 -552 -366 -373 -378   31  753  133  610  448 -179  327 -756  -41 -166
## [241]  360  254 -506  771 -229  380   24 -937  466
```

```r
yVec[yVec > 600]
```

```
##  [1] 817 868 755 815 846 620 736 783 687 934 916 840 622 738 818 683 925 636 832
## [20] 847 966 854 653 941 658 853 681 765 803 702 665 900 698 798 759 877 657 903
## [39] 724 731 776 861 897 826 961 721 865 999 751 754 934 812 779 881 922 896 909
## [58] 871 845 700 685 918 960 610 848 826 859 958 646 815 808 880 793 946 791 975
## [77] 778 949 942 780 697 976 905 661 605 883 991 733 960 753 851 923
```

```r
which(yVec > 600)
```

```
##  [1]   1   2   6  14  15  18  19  20  21  23  24  25  26  27  28  29  40  43  44
## [20]  54  55  56  64  65  69  70  73  79  80  81  87  88  96  97  98 105 108 109
## [39] 114 115 119 121 124 125 128 129 133 135 136 137 138 141 142 143 144 146 147
## [58] 148 150 151 153 155 157 163 171 172 178 180 184 187 189 190 194 195 196 199
```

```
## [77] 205 206 211 216 218 221 222 226 231 233 234 235 242 243 245 248
```

```r
xVec[order(yVec)]
```

```
##   [1] 477 424 145 539 712 970   0 573 207 948 781 107 484 137 842 699 624 155
##  [19] 437 157 243 368 240 962 555 389 385 791 788 819  85  89 191 130 234 703
##  [37] 278 392 485 496 108 968 450 595 850 372  80 819 486 881 390  58 784 963
##  [55] 516 217 487  23 431 932 737  31 278  56 323 815 324 328 669 428 368 623
##  [73] 961 186 258 520 545 711  10 890 381 212 197 987 200 824 684 306 374 536
##  [91] 658 326 423  79 471 543 778 256 804 647  83 666 250 148 442 375 793 529
## [109] 101 797 490 510 200  93  69  10 131 112 875 981 714  64 109  96 780 606
## [127] 899 144 223 524 442 919 357 207 603 803 896 279  65 908 626 878 273 444
## [145] 600  81  71 389 328 297 863 605 157 636 780 932 224 983 108 187 455 942
## [163] 148 831  70 720 939 931 539 444 872  35 373 124  16  98 803 635 682 529
## [181]  58 289 316 140 715 140 342 187  19 667 491 131 213 438 897 219 163 392
## [199] 387 579 385 898 386  67 568   7 196 224 902  48 386  37 639 366 253  13
## [217] 578 926 521  23 738 589 601 858 225 411 223 176 393 414 552 617 412 964
## [235] 164 657 599 326 617  35 924 441 623 499 808 931 152 606 123 595
```

```r
yVec[seq(1, length(yVec), by = 3)]
```

```
##  [1] 817 461 305 371  80 160 736 292 840 818 290 232 464 925 636 466  29 353 966
## [20] 307 490 653 335 853 681  62 765  31 494 900 100  90 798 503 319 244 903  11
## [39] 731 193 861 897 440 110 865 751 289 779 549 871 700 226 960 214 610 324 544
## [58] 826 594 859 559 646 815 880 117 791 975 382 778 363 942 438 319  56 387 661
## [77]  26 139 733 527 548 209 481 466
```

3.

For this problem we'll use the (built-in) dataset `state.x77`.

```r
data(state)
state.x77 <- as_tibble(state.x77, rownames  = 'State')
```

a. Select all the states having an income less than 4300, and calculate the average income of these states.
b. Sort the data by income and select the state with the highest income.
c. Add a variable to the data frame which categorizes the size of population: $<= 4500$ is `S`, > 4500 is `L`.
d. Find out the average income and illiteracy of the two groups of states, distinguishing by whether the states are small or large.

```r
data(state)
state.x77 <- as_tibble(state.x77, rownames = "State")

income_below_4300 <- state.x77 %>%
    filter(Income < 4300)

income_below_4300
```

```
## # A tibble: 20 x 9
##    State    Population Income Illiteracy `Life Exp` Murder `HS Grad` Frost  Area
##    <chr>         <dbl>  <dbl>      <dbl>      <dbl>  <dbl>     <dbl> <dbl> <dbl>
##  1 Alabama        3615   3624        2.1       69.0   15.1      41.3    20 50708
##  2 Arkans~        2110   3378        1.9       70.7   10.1      39.9    65 51945
##  3 Georgia        4931   4091        2         68.5   13.9      40.6    60 58073
##  4 Idaho           813   4119        0.6       71.9    5.3      59.5   126 82677
##  5 Kentuc~        3387   3712        1.6       70.1   10.6      38.5    95 39650
```

```
##  6 Louisi~       3806   3545         2.8        68.8   13.2       42.2    12  44930
##  7 Maine         1058   3694         0.7        70.4    2.7       54.7   161  30920
##  8 Missis~       2341   3098         2.4        68.1   12.5       41      50  47296
##  9 Missou~       4767   4254         0.8        70.7    9.3       48.8   108  68995
## 10 New Ha~        812   4281         0.7        71.2    3.3       57.6   174   9027
## 11 New Me~       1144   3601         2.2        70.3    9.7       55.2   120 121412
## 12 North ~       5441   3875         1.8        69.2   11.1       38.5    80  48798
## 13 Oklaho~       2715   3983         1.1        71.4    6.4       51.6    82  68782
## 14 South ~       2816   3635         2.3        68.0   11.6       37.8    65  30225
## 15 South ~        681   4167         0.5        72.1    1.7       53.3   172  75955
## 16 Tennes~       4173   3821         1.7        70.1   11         41.8    70  41328
## 17 Texas        12237   4188         2.2        70.9   12.2       47.4    35 262134
## 18 Utah          1203   4022         0.6        72.9    4.5       67.3   137  82096
## 19 Vermont        472   3907         0.6        71.6    5.5       57.1   168   9267
## 20 West V~       1799   3617         1.4        69.5    6.7       41.6   100  24070
```

```r
avg_income <- mean(income_below_4300$Income)
avg_income
```

```
## [1] 3830.6
```

```r
highest_income_state <- state.x77 %>%
    arrange(desc(Income)) %>%
    slice(1)
highest_income_state
```

```
## # A tibble: 1 x 9
##   State  Population Income Illiteracy `Life Exp` Murder `HS Grad` Frost   Area
##   <chr>       <dbl>  <dbl>      <dbl>      <dbl>  <dbl>     <dbl> <dbl>  <dbl>
## 1 Alaska        365   6315        1.5       69.3   11.3      66.7   152 566432
```

```r
state.x77 <- state.x77 %>%
    mutate(Population_Size = ifelse(Population <= 4500, "S", "L"))
state.x77
```

```
## # A tibble: 50 x 10
##    State    Population Income Illiteracy `Life Exp` Murder `HS Grad` Frost   Area
##    <chr>         <dbl>  <dbl>      <dbl>      <dbl>  <dbl>     <dbl> <dbl>  <dbl>
##  1 Alabama       3615   3624         2.1        69.0   15.1       41.3    20  50708
##  2 Alaska         365   6315         1.5        69.3   11.3       66.7   152 566432
##  3 Arizona       2212   4530         1.8        70.6    7.8       58.1    15 113417
##  4 Arkans~       2110   3378         1.9        70.7   10.1       39.9    65  51945
##  5 Califo~      21198   5114         1.1        71.7   10.3       62.6    20 156361
##  6 Colora~       2541   4884         0.7        72.1    6.8       63.9   166 103766
##  7 Connec~       3100   5348         1.1        72.5    3.1       56     139   4862
##  8 Delawa~        579   4809         0.9        70.1    6.2       54.6   103   1982
##  9 Florida       8277   4815         1.3        70.7   10.7       52.6    11  54090
## 10 Georgia       4931   4091         2          68.5   13.9       40.6    60  58073
## # i 40 more rows
## # i 1 more variable: Population_Size <chr>
```

```r
S_pop_states <- state.x77 %>%
    filter(Population_Size == "S")
L_pop_states <- state.x77 %>%
    filter(Population_Size == "L")

avg_income_S <- mean(S_pop_states$Income)
```

```r
avg_income_L <- mean(L_pop_states$Income)
avg_illiteracy_S <- mean(S_pop_states$Illiteracy)
avg_illiteracy_L <- mean(L_pop_states$Illiteracy)

data.frame(
    Population_Size = c("S", "L"),
    Avg_Income = c(avg_income_S, avg_income_L),
    Avg_Illiteracy = c(avg_illiteracy_S, avg_illiteracy_L)
)
```

```
##   Population_Size Avg_Income Avg_Illiteracy
## 1               S   4354.794       1.155882
## 2               L   4607.938       1.200000
```

4.

a. Write a function to simulate `n` observations of (X_1, X_2) which follow the uniform distribution over the square $[0,1] \times [0,1]$.

b. Write a function to calculate the proportion of the observations that the distance between (X_1, X_2) and the nearest edge is less than 0.25, and the proportion of them with the distance to the nearest vertex less than 0.25.

```r
simulate_uniform <- function(n) {
    data.frame(
        X1 = runif(n, 0, 1),
        X2 = runif(n, 0, 1)
    )
}
simulate_uniform(10) # For example, simulate 10 observations
```

```
##           X1         X2
## 1  0.5104605 0.48520525
## 2  0.1786997 0.40600410
## 3  0.8869470 0.22097863
## 4  0.4493209 0.65762326
## 5  0.2036143 0.96115245
## 6  0.1388348 0.78422080
## 7  0.1291463 0.13528432
## 8  0.7844916 0.19371239
## 9  0.3423287 0.01355722
## 10 0.6649215 0.09887688
```

```r
calculate_proportions <- function(data) {
    distances_to_edges <- pmin(data$X1, 1 - data$X1, data$X2, 1 - data$X2)
    distances_to_vertices <- pmin(
        sqrt(data$X1^2 + data$X2^2),
        sqrt((data$X1 - 1)^2 + data$X2^2),
        sqrt(data$X1^2 + (data$X2 - 1)^2),
        sqrt((data$X1 - 1)^2 + (data$X2 - 1)^2)
    )

    proportion_edges <- mean(distances_to_edges < 0.25)
    proportion_vertices <- mean(distances_to_vertices < 0.25)

    data.frame(
```

```r
        proportion_edges = proportion_edges,
        proportion_vertices = proportion_vertices
    )
}

# Example usage
data <- simulate_uniform(1000) # Simulate 1000 observations
proportions <- calculate_proportions(data)
proportions
```

```
##   proportion_edges proportion_vertices
## 1            0.759               0.202
```

5.

To estimate π with a Monte Carlo simulation, we draw the unit circle inside the unit square, the ratio of the area of the circle to the area of the square will be $\pi/4$. Then shot K arrows at the square, roughly $K * \pi/4$ should have fallen inside the circle. So if now you shoot N arrows at the square, and M fall inside the circle, you have the following relationship $M = N * \pi/4$. You can thus compute π like so: $\pi = 4 * M/N$. The more arrows N you throw at the square, the better approximation of π you'll have.

```r
n <- 10000

set.seed(1)
points <- tibble("x" = runif(n), "y" = runif(n))
```

Now, to know if a point is inside the unit circle, we need to check whether $x^2 + y^2 < 1$. Let's add a new column to the points tibble, called `inside` equal to `1` if the point is inside the unit circle and `0` if not:

```r
points <- points |>
    mutate(inside = map2_dbl(.x = x, .y = y, ~ifelse(.x**2 + .y**2 < 1, 1, 0))) |>
    rowid_to_column("N")
```

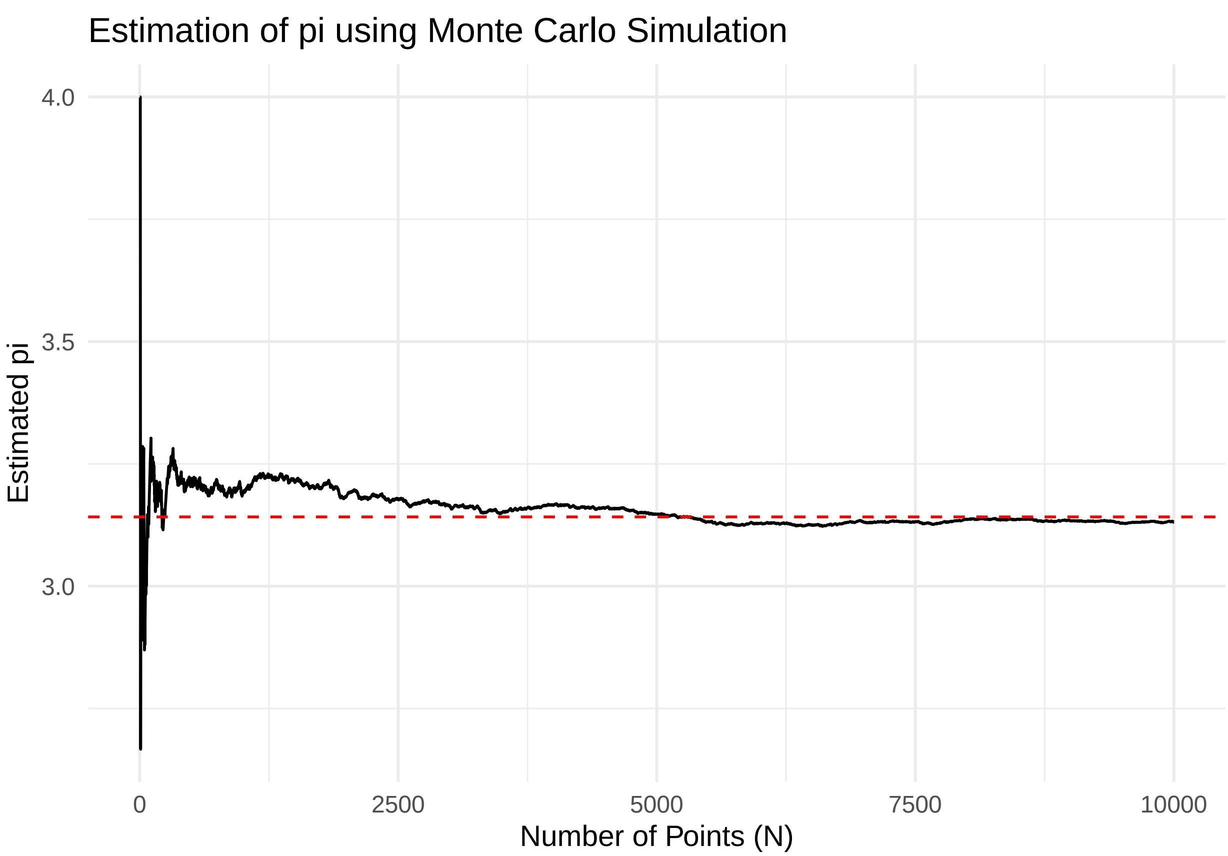
a. Compute the estimation of π at each row, by computing the cumulative sum of the 1's in the `inside` column and dividing that by the current value of `N` column:
b. Plot the estimates of π against `N`.

```r
n <- 10000
set.seed(1)
points <- tibble(
    x = runif(n, 0, 1),
    y = runif(n, 0, 1)
)

points <- points |>
    mutate(inside = map2_dbl(.x = x, .y = y, ~ ifelse(.x**2 + .y**2 < 1, 1, 0))) |>
    rowid_to_column("N")

points <- points |>
    mutate(pi_estimate = cumsum(inside) / N * 4)

head(points, n = 10)
```

```
## # A tibble: 10 x 5
##        N     x     y inside pi_estimate
##    <int> <dbl> <dbl>  <dbl>       <dbl>
```

```
## 1      1 0.266  0.0647       1         4
## 2      2 0.372  0.677        1         4
## 3      3 0.573  0.735        1         4
## 4      4 0.908  0.111        1         4
## 5      5 0.202  0.0467       1         4
## 6      6 0.898  0.131        1         4
## 7      7 0.945  0.881        0         3.43
## 8      8 0.661  0.840        0         3
## 9      9 0.629  0.868        0         2.67
## 10    10 0.0618 0.0334       1         2.8
```

```r
ggplot(points, aes(x = N, y = pi_estimate)) +
    geom_line() +
    labs(
        title = "Estimation of pi using Monte Carlo Simulation",
        x = "Number of Points (N)",
        y = "Estimated pi"
    ) +
    theme_minimal() +
    geom_hline(yintercept = pi, color = "red", linetype = "dashed")
```



6.

Mortality rates per 100,000 from male suicides for a number of age groups and a number of countries are given in the following data frame.

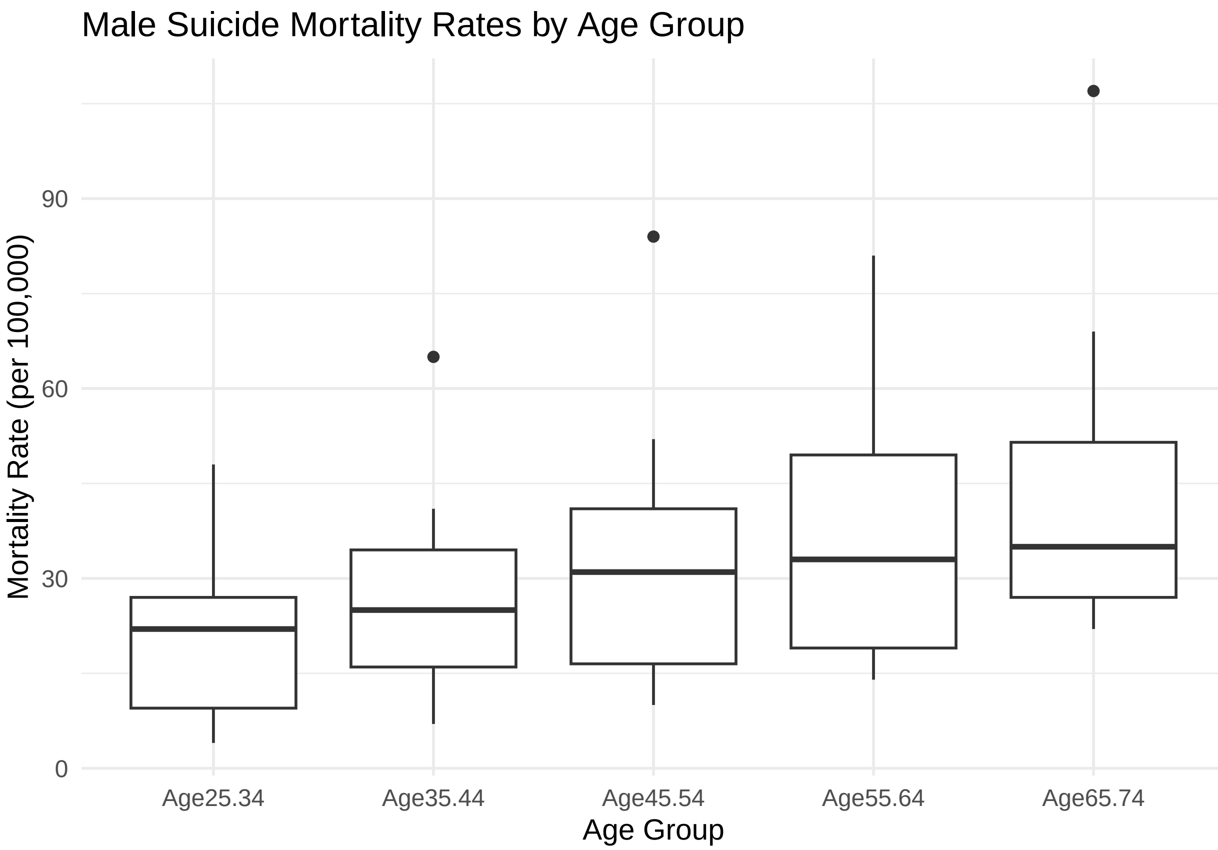
```r
suicrates <- tibble(Country = c('Canada', 'Israel', 'Japan', 'Austria', 'France', 'Germany',
'Hungary', 'Italy', 'Netherlands', 'Poland', 'Spain', 'Sweden', 'Switzerland', 'UK', 'USA'),
```

```
Age25.34 = c(22,  9, 22, 29, 16, 28, 48,  7,  8, 26,  4, 28, 22, 10, 20),
Age35.44 = c(27, 19, 19, 40, 25, 35, 65,  8, 11, 29,  7, 41, 34, 13, 22),
Age45.54 = c(31, 10, 21, 52, 36, 41, 84, 11, 18, 36, 10, 46, 41, 15, 28),
Age55.64 = c(34, 14, 31, 53, 47, 49, 81, 18, 20, 32, 16, 51, 50, 17, 33),
Age65.74 = c(24, 27, 49, 69, 56, 52, 107, 27, 28, 28, 22, 35, 51, 22, 37))
```

a. Transform `suicrates` into *long* form.

b. Construct side-by-side box plots for the data from different age groups, and comment on what the graphic tells us about the data.

```
suicrates <- tibble(
    Country = c(
        "Canada", "Israel", "Japan", "Austria", "France", "Germany",
        "Hungary", "Italy", "Netherlands", "Poland", "Spain", "Sweden", "Switzerland", "UK", "USA"
    ),
    Age25.34 = c(22, 9, 22, 29, 16, 28, 48, 7, 8, 26, 4, 28, 22, 10, 20),
    Age35.44 = c(27, 19, 19, 40, 25, 35, 65, 8, 11, 29, 7, 41, 34, 13, 22),
    Age45.54 = c(31, 10, 21, 52, 36, 41, 84, 11, 18, 36, 10, 46, 41, 15, 28),
    Age55.64 = c(34, 14, 31, 53, 47, 49, 81, 18, 20, 32, 16, 51, 50, 17, 33),
    Age65.74 = c(24, 27, 49, 69, 56, 52, 107, 27, 28, 28, 22, 35, 51, 22, 37)
)

suicrates_long <- suicrates %>%
    pivot_longer(
        cols = starts_with("Age"),
        names_to = "Age_Group",
        values_to = "Mortality_Rate"
    )

ggplot(suicrates_long, aes(x = Age_Group, y = Mortality_Rate)) +
    geom_boxplot() +
    labs(
        title = "Male Suicide Mortality Rates by Age Group",
        x = "Age Group",
        y = "Mortality Rate (per 100,000)"
    ) +
    theme_minimal()
```



Comment:

The box plots show that:

- **25-34 Age Group**: The suicide mortality rate is relatively low in this age group, with a median of approximately 20. The interquartile range is narrow, indicating that the data points are closely clustered.
- **35-44 Age Group**: The suicide mortality rate increases, with a median of around 30. The interquartile range is wider, indicating greater variability in the data.
- **45-54 Age Group**: The suicide mortality rate continues to rise, with a median approaching 40. The interquartile range is wider, indicating greater variability in the data.
- **55-64 Age Group**: The suicide mortality rate further increases, with a median close to 50. The interquartile range is wider, indicating greater variability in the data.
- **65-74 Age Group**: The suicide mortality rate is the highest, with a median close to 60. The interquartile range is wider, indicating greater variability in the data. Additionally, there is an outlier in this age group with a mortality rate approaching 80.

7.

Load the `LaborSupply` dataset from the `{Ecdat}` package and answer the following questions:

```r
#data(LaborSupply)
LaborSupply <- read_csv("LaborSupply.csv")
# create hour and wage variables
labor <- LaborSupply |>
  mutate(hour = exp(lnhr), wage = exp(lnwg), .before = kids) |>
  dplyr::select(-lnhr, -lnwg)
```

a. Compute the average annual hours worked and their standard deviations by year.

b. What age group worked the most hours in the year 1982?

c. Create a variable, `n_years` that equals the number of years an individual stays in the panel. Is the panel balanced?

d. Which are the individuals that do not have any kids during the whole period? Create a variable, `no_kids`, that flags these individuals (1 = no kids, 0 = kids)

e. Using the `no_kids` variable from before compute the average wage, standard deviation and number of observations in each group for the year 1980 (no kids group vs kids group).

```
LaborSupply <- read_csv("./LaborSupply.csv")
```

```
## Rows: 5320 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (7): lnhr, lnwg, kids, age, disab, id, year
##
## 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.
```

```
labor <- LaborSupply |>
    mutate(hour = exp(lnhr), wage = exp(lnwg), .before = kids) |>
    dplyr::select(-lnhr, -lnwg)

# Problem A
labor %>%
    group_by(year) %>%
    summarise(
        avg_hours = mean(hour, na.rm = TRUE),
        sd_hours = sd(hour, na.rm = TRUE),
        .groups = "drop"
    )
```

```
## # A tibble: 10 x 3
##     year avg_hours sd_hours
##    <dbl>     <dbl>    <dbl>
##  1  1979     2202.     502.
##  2  1980     2182.     454.
##  3  1981     2185.     460.
##  4  1982     2145.     442.
##  5  1983     2124.     550.
##  6  1984     2149.     492.
##  7  1985     2203.     515.
##  8  1986     2195.     482.
##  9  1987     2219.     529.
## 10  1988     2222.     478.
```

```
# Problem B
labor %>%
    filter(year == 1982) %>%
    group_by(age) %>%
    summarise(avg_hours = mean(hour, na.rm = TRUE), .groups = "drop") %>%
    arrange(desc(avg_hours)) %>%
    slice(1)
```

```
## # A tibble: 1 x 2
##     age avg_hours
```

```
##   <dbl>     <dbl>
## 1    46     2373.
```

```r
# Problem C
# Count number of years per individual
years_per_individual <- labor %>%
    group_by(id) %>%
    summarise(n_years = n_distinct(year), .groups = "drop")

n_years <- years_per_individual$n_years

labor <- labor %>%
    left_join(years_per_individual, by = "id")

is_balanced <- length(unique(years_per_individual$n_years)) == 1
is_balanced
```

```
## [1] TRUE
```

```r
# Problem D
no_kids <- labor %>%
    group_by(id) %>%
    summarise(no_kids = ifelse(all(kids == 0), 1, 0), .groups = "drop")

head(no_kids, n = 10)
```

```
## # A tibble: 10 x 2
##       id no_kids
##    <dbl>   <dbl>
##  1     1       0
##  2     2       0
##  3     3       1
##  4     4       0
##  5     5       0
##  6     6       0
##  7     7       0
##  8     8       0
##  9     9       0
## 10    10       1
```

```r
labor <- labor %>%
    left_join(no_kids, by = "id")

# Problem E
labor %>%
    filter(year == 1980) %>%
    group_by(no_kids) %>%
    summarise(
        avg_wage = mean(wage, na.rm = TRUE),
        sd_wage = sd(wage, na.rm = TRUE),
        n_obs = n(),
        .groups = "drop"
    )
```

```
## # A tibble: 2 x 4
##   no_kids avg_wage sd_wage n_obs
```

```
##      <dbl>    <dbl>   <dbl> <int>
## 1        0     14.5    6.69   489
## 2        1     15.9    6.71    43
```