PROGRAMMING FOR DATA ANALYSIS

CT127-3-2-PFDA

Individual Assignment

Assignment Title	:	Employee Dataset
Student Name	:	William Wijaya
TP Number	:	TP059360
Intake Code	:	APU2F2109CS(CYB)
Lecturer Name	:	Minnu Helen Joseph
Hand Out Date	:	4th October 2021
Hand In Date	:	11st November 2021

Table of Contents

Table of Figure

Introduction and Assumptions

The given dataset contains a list of employee records with the attributes of their work data, namely gender, hired date, termination date, cause of termination, job position, job location, etc. By analysing this dataset, we may learn the behaviour of the company in terms of hiring and termination criterion.

Assumption:

- There was no termination of employees before 2006.
- Every employee only has one job throughout their career in the company.

```
empID <- emp_table$EmployeeID %>% unique
missing <- length(seq(min(empID), max(empID))) - length(empID)
unknown_period <- difftime(
  min(emp_table$recorddate_key),
  min(emp_table$orighiredate_key)
  ) %>%
  time_length("years") %>%
  floor

average_term <- missing / unknown_period
```

Figure 1 Calculate the average of terminated employee before data record is being done

```
> average_term
[1] 45.9375
```

Figure 2 Output of the average terminated employee before data were recorded

Assuming that the company starts naming the employee ID from 1318 and goes sequentially, we can see there was only 46 employees terminated every year before the data was first recorded. Thus, the assumption is made that there is no termination of employees before that since the number is relatively small.

Data Import and Tidying

```
library(stringr)
library(plyr)
library(dplyr)
library(tidyr)
library(tibble)
library(lubridate)
library(ggplot2)
library(plotrix)

rm(list=ls())
setwd("C:/Users/LEGION.LAPTOP-HVMBBO3R/OneDrive - Asia Pacific University
/Documents/My Documents/APU/Semester 3/Programming-for-Data-Analysis/Assignment
/my/")
emp_data <- read.csv("employee_attrition.csv", header = TRUE)
emp_table <- tibble(emp_data) %>%
    mutate(
        recorddate_key = as.Date(recorddate_key, format = "%m/%d/%Y"),
        birthdate_key = as.Date(birthdate_key, format = "%m/%d/%Y"),
        orighiredate_key = as.Date(orighiredate_key, format = "%m/%d/%Y"),
        terminationdate_key = as.Date(terminationdate_key, format = "%m/%d/%Y"),
        city_name = factor(city_name),
        department_name = factor(department_name),
        job_title = factor(job_title),
        gender_short = factor(gender_short),
        gender_full = factor(gender_full),
        termreason_desc = factor(termreason_desc),
        termtype_desc = factor(termtype_desc),
        STATUS_YEAR = as.Date(as.character(STATUS_YEAR), format = "%Y"),
        STATUS = factor(STATUS),
        BUSINESS_UNIT = factor(BUSINESS_UNIT)
        )

str(emp_table)
```

Figure 3 Data Tidying

```
> str(emp_table)
tibble [49,653 x 18] (S3: tbl_df/tbl/data.frame)
 $ EmployeeID         : int [1:49653] 1318 1318 1318 1318 1318 1318 1318 1318 1318 131
8 ...
 $ recorddate_key     : Date[1:49653], format: "2006-12-31" "2007-12-31" ...
 $ birthdate_key      : Date[1:49653], format: "1954-01-03" "1954-01-03" ...
 $ orighiredate_key   : Date[1:49653], format: "1989-08-28" "1989-08-28" ...
 $ terminationdate_key: Date[1:49653], format: "1900-01-01" "1900-01-01" ...
 $ age                : int [1:49653] 52 53 54 55 56 57 58 59 60 61 ...
 $ length_of_service  : int [1:49653] 17 18 19 20 21 22 23 24 25 26 ...
 $ city_name          : Factor w/ 40 levels "Abbotsford","Aldergrove",..: 35 35 35 35
 35 35 35 35 35 35 ...
 $ department_name    : Factor w/ 21 levels "Accounting","Accounts Payable",..: 10 10
 10 10 10 10 10 10 10 10 ...
 $ job_title          : Factor w/ 47 levels "Accounting Clerk",..: 9 9 9 9 9 9 9 9 9 9
 ...
 $ store_name         : int [1:49653] 35 35 35 35 35 35 35 35 35 35 ...
 $ gender_short       : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ gender_full        : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ termreason_desc    : Factor w/ 4 levels "Layoff","Not Applicable",..: 2 2 2 2 2 2 2
 2 2 2 ...
 $ termtype_desc      : Factor w/ 3 levels "Involuntary",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ STATUS_YEAR        : Date[1:49653], format: "2006-11-03" "2007-11-03" ...
 $ STATUS             : Factor w/ 2 levels "ACTIVE","TERMINATED": 1 1 1 1 1 1 1 1 1 1
 ...
 $ BUSINESS_UNIT      : Factor w/ 2 levels "HEADOFFICE","STORES": 1 1 1 1 1 1 1 1 1 1
 ...
```

Figure 4 Structure of the data set after tidying process

Before starting to analyse the data, we would need to load several packages to improve our analysis in terms of flexibility and efficiency. The first library is *stringr* which will help us in strings manipulation and operations. Next is *plyr*, which will be used to for rounding purposes and since it has overlapping function name with *dplyr* which we will use on our analysis, we will load *plyr* before *dplyr* to prevent masking. The *dplyr* package provide us with data exploration and manipulation tools. Next package, *tidyr,* contains function to help us tidy up data to ease out our analysis process. Then, *tibble* aid us in viewing data frame easily, *lubridate* as date operation library, and lastly *ggplot2* and *plotrix* as data visualization tools.

To have a clean workspace every time we execute our script, we will clean all previous environment variables. Next is data importing and cleaning. The data is read from csv with *header* equals true to use the existing column name and then converted to more convenient data frame using *tibble*. All date data will be converted to the corresponding data type and categorical column will be refactored.

```
theme_update(plot.title = element_text(hjust=0.5),
             axis.title.x = element_text(face="bold", color="black", size=10),
             axis.title.y = element_text(face="bold", color="black", size=10),
             legend.title = element_text(face="bold", size=12),
             axis.text.x = element_text(angle=90, size=8))
```

Figure 5 Configuring the default theme for ggplot graph

Next, the default style of the plot is established using *theme_update* to make our plotting code more readable.

Question 1: How is the company growth throughout the year?

Analysis 1-1: Find the number of employees hired, terminated, and total per year.

First, we will extract the year from the hire date. Since there are multiple observations for a single employee and every employee has one hired year data, we will group them up by year and find the number of distinct employees. The same goes to termination data but with the year 1900 filtered out since it is a default value for employees which are still active.

After computing both recruitment and termination, we merged them up together, clean up *NA* values with 0, and lastly loop through the data frame to compute the number of employees by adding up *n*-th year existing employees with the changes value (hired – termination).

```
compute_hire_amt <- function(dt) {
    hire_amt <- (dt %>%
                     mutate(year = year(orighiredate_key)) %>%
                     group_by(year) %>%
                     summarise(n = n_distinct(EmployeeID))
                     )
    return(hire_amt)
}

compute_term_amt <- function(dt) {
    term_amt <- (dt %>%
                     mutate(year = year(terminationdate_key)) %>%
                     filter(year != 1900) %>%
                     group_by(year) %>%
                     summarise(n = n_distinct(EmployeeID))
                     )
    return(term_amt)
}
```

Figure 6 Function to extract the number of hired employees per year

```r
compute_no_employee <- function(dt) {
    hire_amt <- compute_hire_amt(dt)
    term_amt <- compute_term_amt(dt)
    merged <- merge(hire_amt, term_amt, by = "year", all = TRUE)
    colnames(merged) <- c("year", "hire", "term")
    merged$hire <- ifelse(is.na(merged$hire), 0, merged$hire)
    merged$term <- ifelse(is.na(merged$term), 0, merged$term)
    merged$final <- merged$hire - merged$term
    num_emp <- data.frame(merged$year, merged$hire, merged$term)
    total <- NULL
    for (val in merged$final) {
        if (is.null(total)) {
            total <- val
        } else {
            tmp <- total[length(total)] + val
            total <- c(total, tmp)
        }
    }
    num_emp <- cbind(num_emp, total)
    return(num_emp)
}
```

Figure 7 Function to compute the total number of employees per year

```r
num_emp <- compute_no_employee(emp_table)
colnames(num_emp) <- c("year", "hired", "termed", "total")
```

Figure 8 Code to compute the total number of employees per year and renaming columns

```
> num_emp
   year hired termed total
1  1989    86      0    86
2  1990   214      0   300
3  1991   200      0   500
4  1992   248      0   748
5  1993   298      0  1046
6  1994   298      0  1344
7  1995   312      0  1656
8  1996   273      0  1929
9  1997   312      0  2241
10 1998   341      0  2582
11 1999   315      0  2897
12 2000   305      0  3202
13 2001   212      0  3414
14 2002   238      0  3652
15 2003   218      0  3870
16 2004   214      0  4084
17 2005   256      0  4340
18 2006   239    134  4445
19 2007   237    162  4520
20 2008   246    164  4602
21 2009   249    142  4709
22 2010   254    123  4840
23 2011   242    110  4972
24 2012   259    130  5101
25 2013   218    105  5214
26 2014     0    253  4961
27 2015     0    162  4799
```

Figure 9 Output for total number or employees per year

Since our merged table is not in the appropriate format where the variables act as columns name rather than grouped in a single column, we will use *pivot_longer* to create a new column named *action* which will contain the factor *hired*, *termed*, and *total*, and another column to store the corresponding value. Next, we will construct the data in line plot, with *year* data as the *x*-axis and *amount* as *y*-axis. To list out all the year number in *x*-axis, we use *scale_x_continuous* to create the sequence. As for the *y*-axis, we use *scale_y_continuous* and *round_any* to get the max value round up to nearest multiple of 25.

```r
line_plot <- num_emp %>%
    pivot_longer(
        cols = c("hired", "termed"),
        names_to = "action",
        values_to = "amount"
        ) %>%
    ggplot(aes(x=year, y=amount)) +
    geom_line(aes(color=action), size = 2) +
    scale_x_continuous(
        name = "year",
        breaks = seq(min(num_emp$year), max(num_emp$year))
    ) +
    scale_y_continuous(
        name="amount",
        breaks = seq(
            0,
            round_any(
                max(c(num_emp[,"hired"], num_emp[,"termed"])), 25, ceiling
                ),
            25
        )
    ) +
    labs(title = "\nHired vs Termed\n")
```

Figure 10 Code for plotting hired and termed number of employese per year using line plot

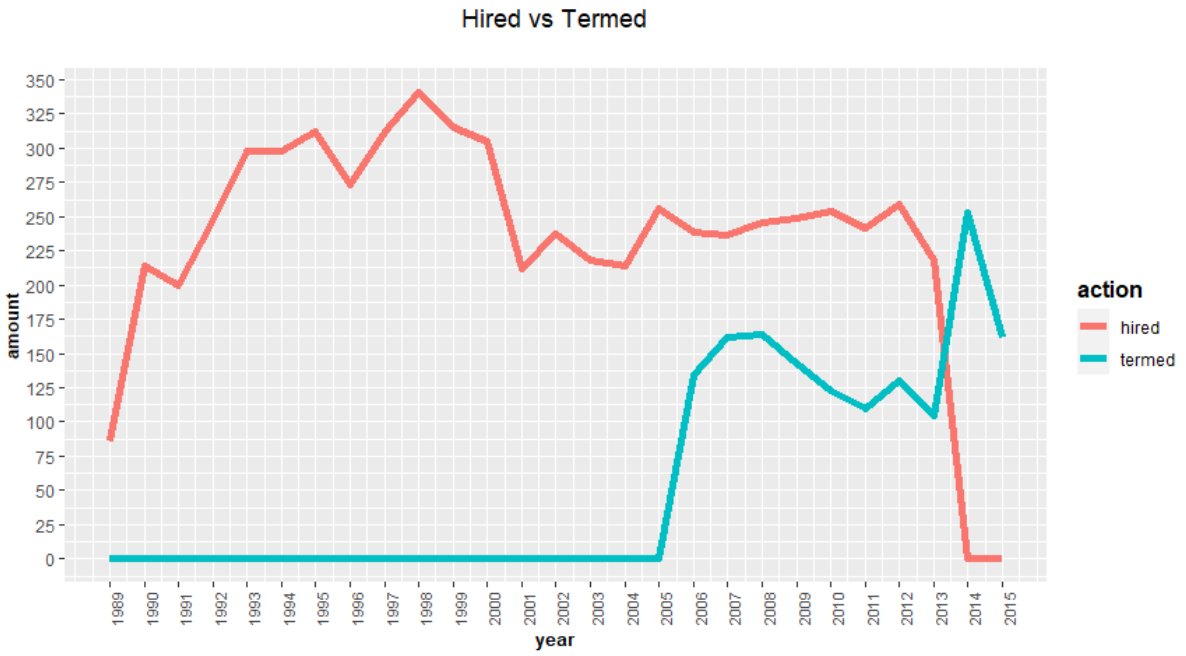


Figure 11 Line plot indicating the number of hired and termed employee per year

Next is to plot the total employees in a bar chart using *ggplot* and *geom_bar*.

```
bar_plot <- ggplot(num_emp, aes(x=year, y=total)) +
    geom_bar(stat="Identity", fill="light blue") +
    geom_text(aes(label=total), size=3) +
    scale_x_continuous(
        breaks = seq(min(num_emp$year), max(num_emp$year))
    ) +
    labs(x = "\nYear\n", y = "\nAmount\n", title = "\nTotal Employees\n")
```

Figure 12 Code for plotting the total number of employees per year using bar plot

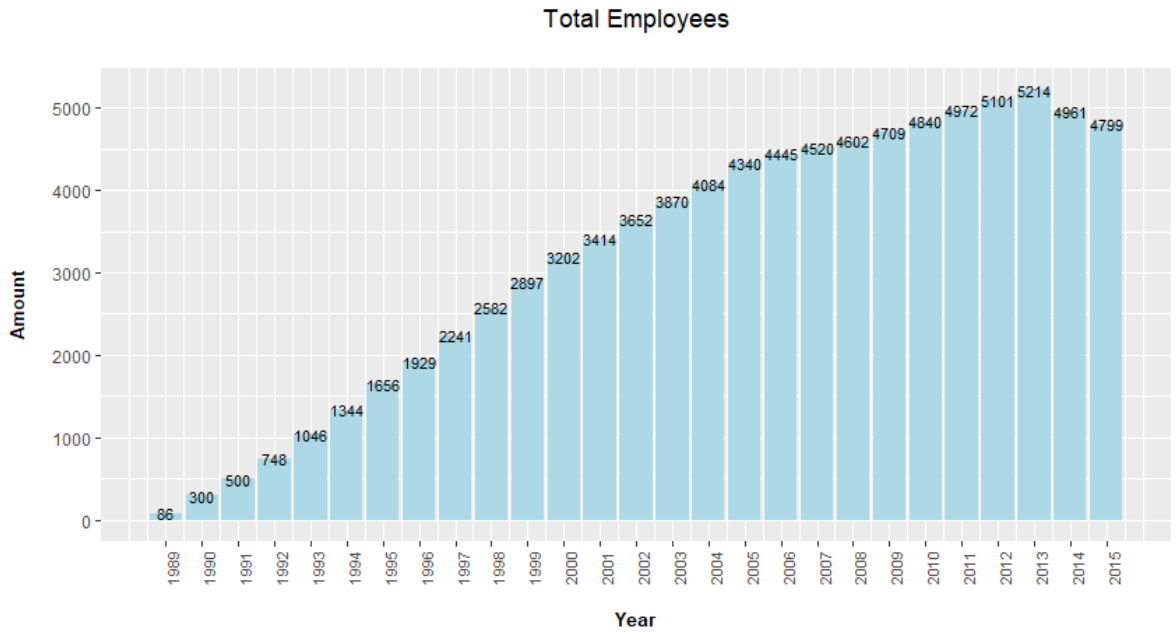


Figure 13 Bar plot for total number of employees per year

Analysis 1-2: Find the number of recruitments and terminations based on business unit per year.

To make our code less repetitive, we will create two functions which will help us in getting terminated and hired employees with grouped column supplied as argument. Using this function, we will get the data grouped by year and business unit. Once again, we will merge both data frames and use *pivot_longer* to tidy up our data.

```r
term_by_var <- function(emp_table, colVec) {
    return(
        emp_table %>%
        mutate(year = year(terminationdate_key)) %>%
        # filter(year != 1900) %>%
        group_by_at(colVec)
    )
}

hire_by_var <- function(emp_table, colVec) {
    return(
        emp_table %>%
        mutate(year = year(orighiredate_key)) %>%
        group_by_at(colVec)
    )
}

# Number of employees hired yearly by BUSINESS UNIT
hire_by_year_BU <- hire_by_var(emp_table, c("year", "BUSINESS_UNIT")) %>%
    summarise(n = n_distinct(EmployeeID))

# Number of employees terminated yearly by BUSINESS UNIT
term_by_year_BU <- term_by_var(emp_table, c("year", "BUSINESS_UNIT")) %>%
    filter(year != 1900) %>%
    summarise(n = n_distinct(EmployeeID))

temp <- merge(hire_by_year_BU, term_by_year_BU, by = c("year", "BUSINESS_UNIT"),
              all = TRUE)
colnames(temp) <- c("year", "BUSINESS_UNIT", "hired", "termed")
temp$hired <- ifelse(is.na(temp$hired), 0, temp$hired)
temp$termed <- ifelse(is.na(temp$termed), 0, temp$termed)
```

Figure 14 Function and code to extract the number of hired and terminated employees yearly by Business Unit

```r
long_format_temp <- temp %>%
    pivot_longer(
        cols = c("hired", "termed"),
        names_to = "action",
        values_to = "amount"
    )

total_amt_per_year <- tapply(
    long_format_temp$amount,
    long_format_temp$year,
    FUN = sum
)
```

Figure 15 Code to tidy up a data frame to long format

```
> hire_by_year_BU
# A tibble: 27 x 3
# Groups:   year [25]
    year BUSINESS_UNIT     n
   <dbl> <fct>         <int>
 1  1989 HEADOFFICE       41
 2  1989 STORES           45
 3  1990 HEADOFFICE       39
 4  1990 STORES          175
 5  1991 STORES          200
 6  1992 STORES          248
 7  1993 STORES          298
 8  1994 STORES          298
 9  1995 STORES          312
10  1996 STORES          273
# ... with 17 more rows
> term_by_year_BU
# A tibble: 15 x 3
# Groups:   year [10]
    year BUSINESS_UNIT     n
   <dbl> <fct>         <int>
 1  2006 STORES          134
 2  2007 HEADOFFICE        1
 3  2007 STORES          161
 4  2008 STORES          164
 5  2009 HEADOFFICE       23
 6  2009 STORES          119
 7  2010 HEADOFFICE        9
 8  2010 STORES          114
 9  2011 STORES          110
10  2012 STORES          130
11  2013 STORES          105
12  2014 HEADOFFICE       24
13  2014 STORES          229
14  2015 HEADOFFICE       12
```

Figure 16 Output for total hired and terminated employee yearly by Business Unit

```
> sample_n(temp, 6)
  year BUSINESS_UNIT hired termed
1 1996        STORES   273      0
2 1998        STORES   341      0
3 1997        STORES   312      0
4 1990    HEADOFFICE    39      0
5 2006        STORES   239    134
6 1989        STORES    45      0
> long_format_temp
# A tibble: 68 x 4
    year BUSINESS_UNIT action amount
   <dbl> <fct>         <chr>   <dbl>
 1  1989 HEADOFFICE    hired      41
 2  1989 HEADOFFICE    termed      0
 3  1989 STORES        hired      45
 4  1989 STORES        termed      0
 5  1990 HEADOFFICE    hired      39
 6  1990 HEADOFFICE    termed      0
 7  1990 STORES        hired     175
 8  1990 STORES        termed      0
 9  1991 STORES        hired     200
10  1991 STORES        termed      0
# ... with 58 more rows
> total_amt_per_year
1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004
  86  214  200  248  298  298  312  273  312  341  315  305  212  238  218  214
2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
 256  373  399  410  391  377  352  389  323  253  162
```

Figure 17 Output showing the merged data frame in wide format and long format

Next is to plot our data as bar chart. To be able to fill the bar with two grouped columns, we will mutate the data frame by creating a new column with both value from each column concatenated. Since for a single year, there are multiple observation, the formed bar chart is stacked.

```r
analysis_1_2 <- long_format_temp %>%
    mutate(
        BU_action = factor(
            str_replace(interaction(BUSINESS_UNIT, action), "\\.", "\n")
        )
    ) %>%
    ggplot(aes(x=year, y=amount, fill=BU_action)) +
    geom_bar(stat = "Identity") +
    scale_x_continuous(
        breaks = seq(min(long_format_temp$year), max(long_format_temp$year))
    ) +
    scale_y_continuous(
        breaks = seq(0, round_any(max(total_amt_per_year), 50, ceiling), 50)
    ) +
    scale_fill_discrete(
        name = "Action on BU"
    ) +
    labs(
        x="Year",
        title = "\nHired vs Termed by Business Unit yearly\n",
    ) +
    theme(legend.text = element_text(size=8))
```

Figure 18 Code for plotting yearly hired and terminated employees by Business Unit using stacked bar chart

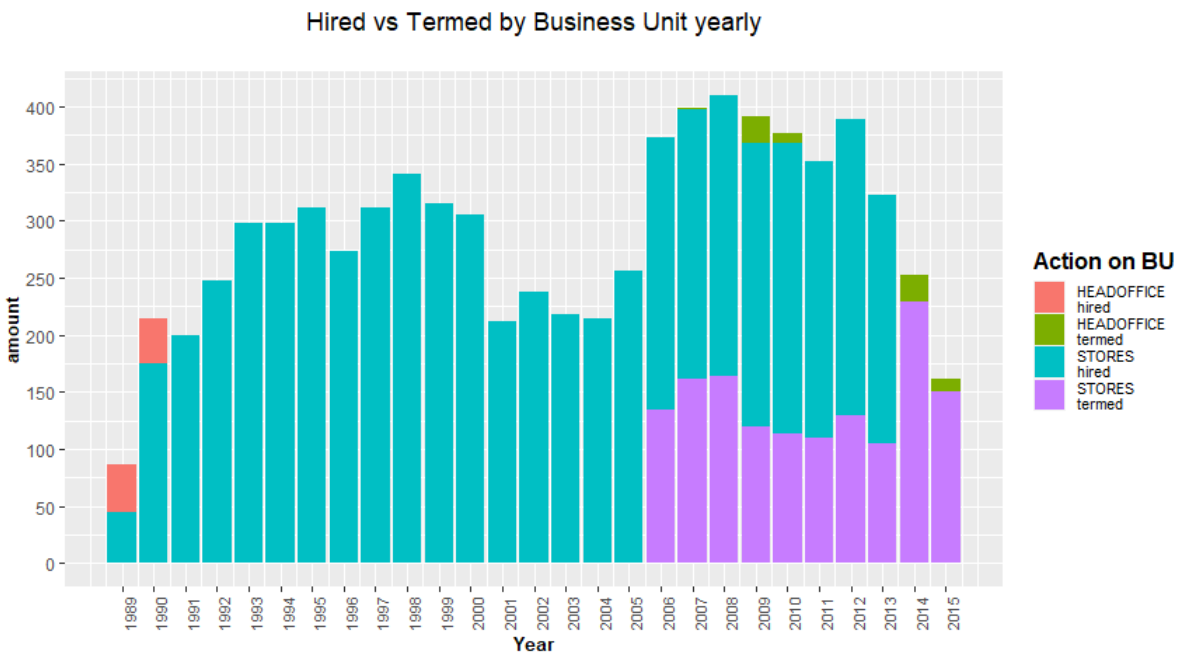


Figure 19 Stacked bar chart of hired and terminated employees by Business Unit yearly

Analysis 1-3: Find the number of recruitments and termination based on gender.

Similar to previous bar chart where each year there will be two genders, instead of stacking them up together, we will display it side by side using parameter *position = "dodge"* in *geom_bar* function.

```
hire_by_year_gender <- hire_by_var(emp_table, c("year", "gender_short")) %>%
    summarise(n = n_distinct(EmployeeID))
colnames(hire_by_year_gender) <- c("year", "gender", "n")

analysis_1_3_a <- hire_by_year_gender %>%
    ggplot(aes(x=year, y=n, fill=gender)) +
    geom_bar(stat = "Identity", position = "dodge", width = 0.7) +
    scale_x_continuous(breaks = seq(min(hire_by_year_gender$year),
                                    max(hire_by_year_gender$year))) +
    scale_y_continuous(
        breaks = seq(0, round_any(max(hire_by_year_gender$n), 25, ceiling), 25)
    ) +
    scale_fill_discrete(name = "Gender") +
    labs(x="Year", title = "\nNumbers of hired employees by gender yearly")
```

Figure 20 Code for plotting the number of hired employees yearly based on gender using side-by-side bar chart

```
> hire_by_year_gender
# A tibble: 50 x 3
# Groups:   year [25]
    year gender     n
   <dbl> <fct>  <int>
 1  1989 F         40
 2  1989 M         46
 3  1990 F        118
 4  1990 M         96
 5  1991 F        105
 6  1991 M         95
 7  1992 F        142
 8  1992 M        106
 9  1993 F        188
10  1993 M        110
# ... with 40 more rows
```

Figure 21 Output of the data frame with the data of hired gender each year

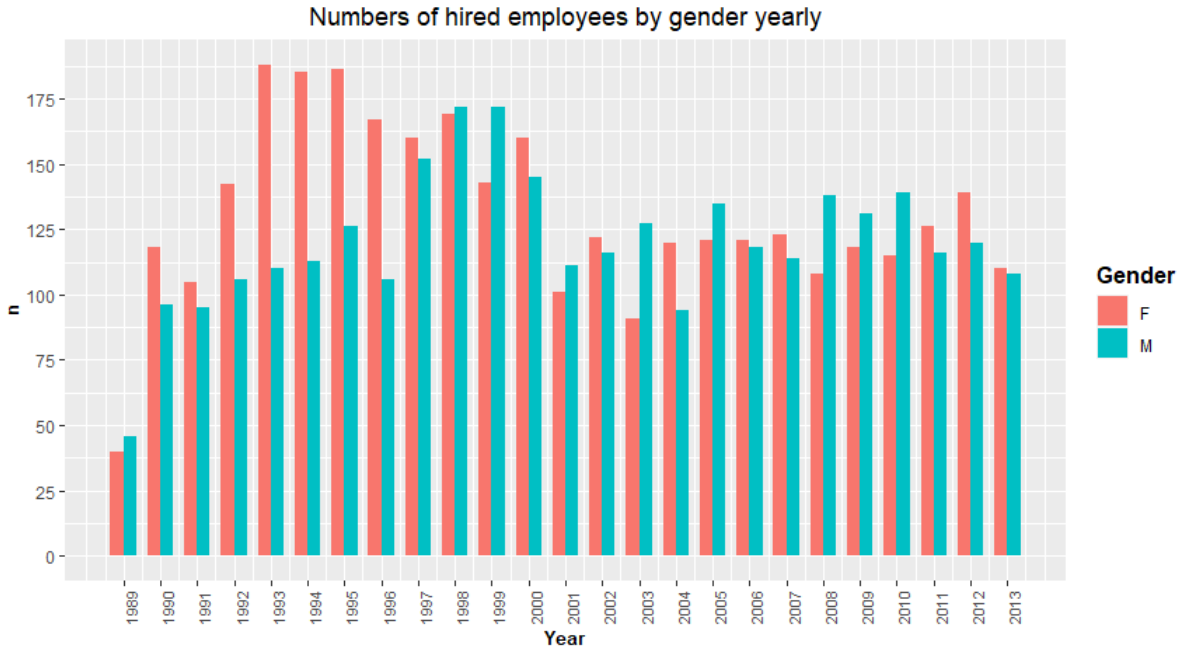


Figure 22 Side-by-side bar chart showing the number of hired employees based on gender yearly

```
term_by_year_gender <- term_by_var(emp_table, c("year", "gender_short")) %>%
    filter(year != 1900) %>%
    summarise(n = n_distinct(EmployeeID))
colnames(term_by_year_gender) <- c("year", "gender", "n")

analysis_1_3_b <- term_by_year_gender %>%
    ggplot(aes(x=year, y=n, fill=gender)) +
    geom_bar(stat = "Identity", position = "dodge", width = 0.7) +
    scale_x_continuous(breaks = seq(min(term_by_year_gender$year),
                                    max(term_by_year_gender$year))) +
    scale_y_continuous(
        breaks = seq(0, round_any(max(term_by_year_gender$n), 25, ceiling), 25)
    ) +
    scale_fill_discrete(name = "Gender") +
    labs(x="Year", title = "\nNumbers of termed employees by gender yearly")
```

Figure 23 Code for plotting the number of terminated employees yearly based on gender using side-by-side bar chart

```
> sample_frac(term_by_year_gender, 0.5)
# A tibble: 10 x 3
# Groups:   year [10]
    year gender     n
   <dbl> <fct>  <int>
 1  2006 M         74
 2  2007 M         81
 3  2008 F         84
 4  2009 F         78
 5  2010 M         59
 6  2011 M         35
 7  2012 F         98
 8  2013 F         95
 9  2014 F        152
10  2015 F        128
```

Figure 24 Output for data frame with data of terminated employees yearly by gender

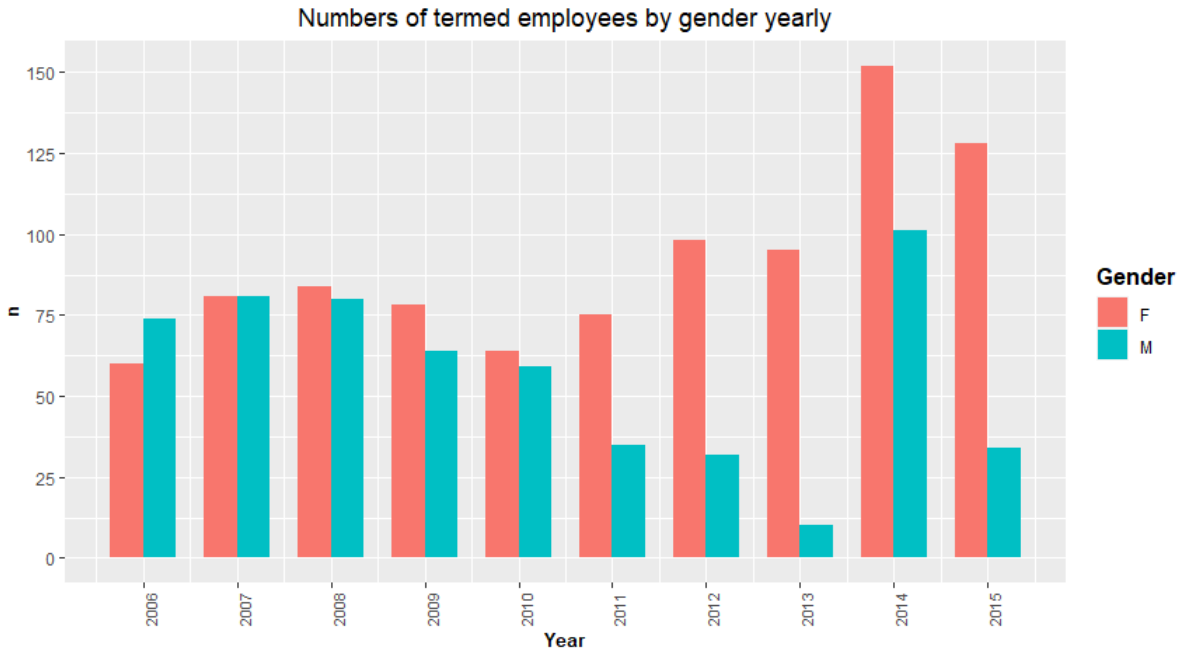


Figure 25 Side-by-side bar chart showing the number of terminated employees yearly based on gender

To view the data as overall, we will use similar code logic as previous but instead of grouping by year, we will group by gender and action and compute the total for each group. Next, we ungroup it and create a new column which will act as factor for our chart colour. The formed chart is a pie chart with the help of *coord_polar* function.

```r
by_year_gender <- merge(hire_by_year_gender, term_by_year_gender,
                        by = c("year", "gender"), all = TRUE)
colnames(by_year_gender) <- c("year", "gender", "hired", "termed")
by_year_gender$hired <- ifelse(is.na(by_year_gender$hired), 0,
                               by_year_gender$hired)
by_year_gender$termed <- ifelse(is.na(by_year_gender$termed), 0,
                               by_year_gender$termed)

by_year_gender_long <- by_year_gender %>%
    pivot_longer(cols = c("hired", "termed"),
                 names_to = "action", values_to = "amount")

overall_gender <- by_year_gender_long %>%
    group_by(gender, action) %>%
    summarise(total = sum(amount)) %>%
    ungroup() %>%
    mutate(gender_action = factor(paste(gender, action))) %>%
    select(gender_action, total) %>%
    ggplot(aes(x="", y=total, fill=gender_action)) +
    geom_bar(stat = "Identity", color="white", width=0.1) +
    geom_text(aes(label=total), position = position_stack(vjust=0.5)) +
    coord_polar("y") +
    labs(title = "Gender hired and termed overall",
         x = "", y = "") +
    scale_fill_discrete(name = "Gender and action") +
    theme_void()
```

Figure 26 Code for merging hired and terminated employees overall and plotting it using pie chart

```
> head(by_year_gender); head(by_year_gender_long)
  year gender hired termed
1 1989      F    40      0
2 1989      M    46      0
3 1990      F   118      0
4 1990      M    96      0
5 1991      F   105      0
6 1991      M    95      0
# A tibble: 6 x 4
   year gender action amount
  <dbl> <fct>  <chr>   <dbl>
1  1989 F      hired      40
2  1989 F      termed      0
3  1989 M      hired      46
4  1989 M      termed      0
5  1990 F      hired     118
6  1990 F      termed      0
```

Figure 27 Output of the merged data frame in wide and long format

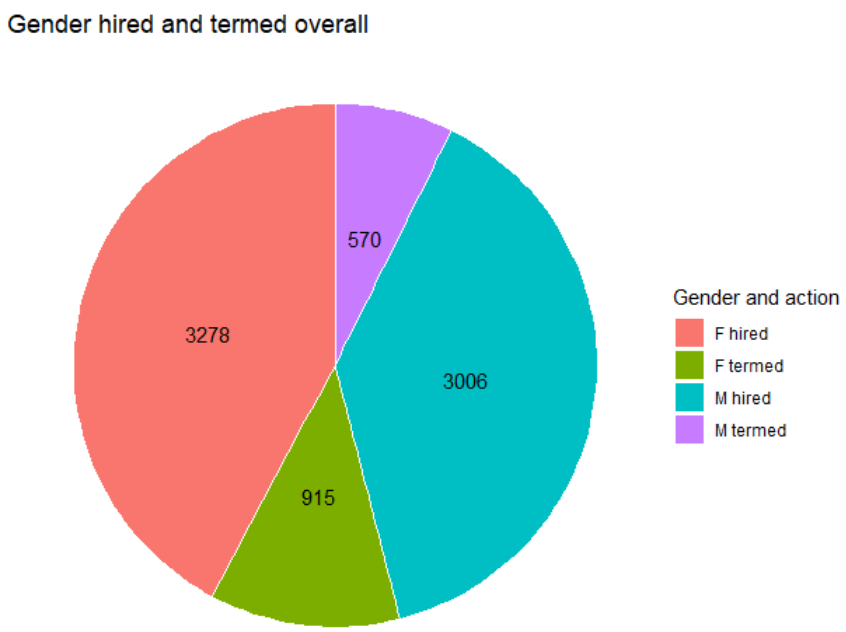


Figure 28 Pie chart showing the total number of hired and terminated employees based on gender

Analysis 1-4: Find the frequency of termination reason variation each year and the overall.

In this analysis, we will look at the data in terms of termination reason. We will filter out the reason *Not Applicable* since it means that the employee is still working/active.

```r
reason_type_variation <- factor(paste(emp_table$termreason_desc,
emp_table$termtype_desc)) %>% levels()

term_var <- emp_table %>%
    filter(termreason_desc != "Not Applicable") %>%
    mutate(year = year(STATUS_YEAR)) %>%
    group_by(termreason_desc, year) %>%
    summarise(n = n()) %>%
    arrange(year)

term_var %>%
    ggplot(aes(x=year, y=n, fill=termreason_desc)) +
    geom_bar(stat="Identity", position = "dodge", width=0.4) +
    scale_x_continuous(breaks = seq(min(term_var$year), max(term_var$year))) +
    scale_y_continuous(breaks = seq(0,
                                    round_any(max(term_var$n), 10, ceiling),
                                    10)) +
    labs(title = "Termination variation per year", y = "amount")
```

Figure 29 Code for plotting the number of employees based on termination reason each year using side-by-side bar chart

```
> term_var
# A tibble: 22 x 3
# Groups:   termreason_desc [3]
   termreason_desc  year     n
   <fct>           <dbl> <int>
 1 Resignaton       2006    12
 2 Retirement       2006   122
 3 Resignaton       2007    25
 4 Retirement       2007   137
 5 Resignaton       2008    26
 6 Retirement       2008   138
 7 Resignaton       2009    18
 8 Retirement       2009   124
 9 Resignaton       2010    29
10 Retirement       2010    94
# ... with 12 more rows
```

Figure 30 Output of the data frame from Figure 29

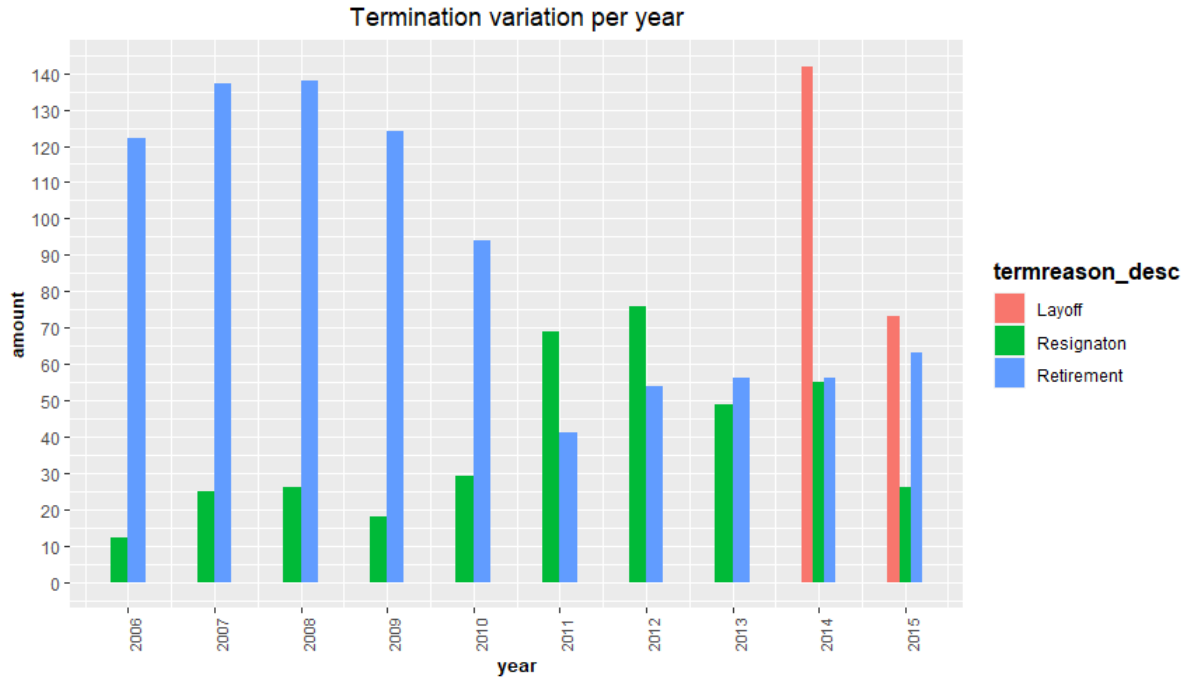


Figure 31 Bar chart showing termination variation per year

Analysis 1-5: Find the relation against department.

```r
hire_by_dept <- hire_by_var(emp_table, c("year", "department_name")) %>%
  summarise(n = n_distinct(EmployeeID))

term_by_dept <- term_by_var(emp_table, c("year", "department_name")) %>%
  filter(year != 1900) %>%
  summarise(n = n_distinct(EmployeeID))

dept_merged <- merge(hire_by_dept, term_by_dept,
      by = c("year", "department_name"),
      all = TRUE)

colnames(dept_merged) <- c("year", "dept", "hired", "termed")
dept_merged$hired <- ifelse(is.na(dept_merged$hired), 0, dept_merged$hired)
dept_merged$termed <- ifelse(is.na(dept_merged$termed), 0, dept_merged$termed)

dept_merged %>%
  pivot_longer(cols = c("hired", "termed"),
               names_to = "action",
               values_to = "number") %>%
  group_by(year, action) %>%
  ggplot(aes(x=year, y=number)) +
  geom_point(aes(colour=action, shape=action, size=number)) +
  scale_size(range = c(1, 3)) +
  labs(title = "Number of hired and termed based on department") +
  facet_wrap(~dept)
```

Figure 32 Code for finding the relation between number of employees and department then plotting it using scatterplot

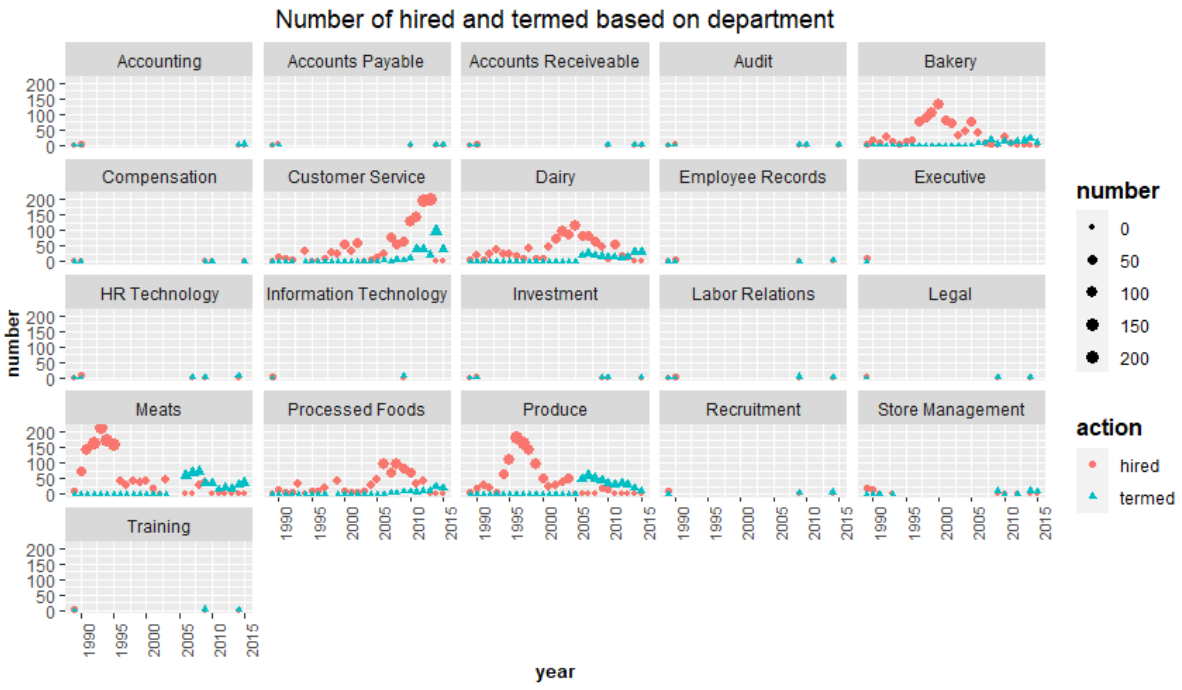


Figure 33 Scatterplot showing number of hired and terminated employees based on department

Conclusion

The company tends to hire more than employees leaving except for the last two years. Furthermore, the company has not hired a single employee for *Headoffice* unit since 1990 but had some employees leaving *Headoffice* unit for the last 5 years. In terms of gender, both hired and leaving employees are mostly dominated by female. The company started to lay off employees for the last two years and most employees leaving are due to retirement. In terms of department, there were 6 departments that had more activities compared to the other.

Question 2: How is the age trend for hired and terminated throughout the years and the overall statistics?

Analysis 2-1: Find the age hired throughout the years.

To get the age when the employee is hired, we can take the hired year subtract and subtract with birth year. This can be achieve using *difftime* then pass it to *time_length* to convert it into years unit. Next, we will replace the *age* column with the hired age value and pass it to *unique* to eliminate duplicate entries.

```
year_diff <- function(date1, date2) {
    return(difftime(date1, date2) %>%
               time_length("years") %>%
               abs %>%
               floor)
}

hired_by_age <- emp_table %>%
  mutate(year = year(orighiredate_key),
         hired_age = year_diff(orighiredate_key, birthdate_key)) %>%
  group_by(year, BUSINESS_UNIT, EmployeeID) %>%
  summarise(age = hired_age) %>%
  unique

hba <- cbind(hired_by_age, action=rep("hired", nrow(hired_by_age)))
```

Figure 34 Code for constructing the data frame with the value of hired age

```
> head(hba)
# A tibble: 6 x 5
# Groups:   year, BUSINESS_UNIT [1]
   year BUSINESS_UNIT EmployeeID   age action
  <dbl> <fct>              <int> <dbl> <chr>
1  1989 HEADOFFICE          1318    35 hired
2  1989 HEADOFFICE          1319    32 hired
3  1989 HEADOFFICE          1320    34 hired
4  1989 HEADOFFICE          1321    30 hired
5  1989 HEADOFFICE          1322    31 hired
6  1989 HEADOFFICE          1323    27 hired
> tail(hba)
# A tibble: 6 x 5
# Groups:   year, BUSINESS_UNIT [1]
   year BUSINESS_UNIT EmployeeID   age action
  <dbl> <fct>              <int> <dbl> <chr>
1  2013 STORES              8331    18 hired
2  2013 STORES              8332    18 hired
3  2013 STORES              8333    18 hired
4  2013 STORES              8334    18 hired
5  2013 STORES              8335    18 hired
6  2013 STORES              8336    18 hired
```

Figure 35 Output of data frame from Figure 34

Analysis 2-2: Find the age terminated throughout the years.

Similar as hired age, to find terminated age, we can take the termination date instead of hired date. Then, we filter out the year 1900.

```
termed_by_age <- emp_table %>%
  mutate(year = year(terminationdate_key),
         termed_age = year_diff(terminationdate_key, birthdate_key)) %>%
  filter(year != 1900) %>%
  group_by(year, BUSINESS_UNIT, EmployeeID) %>%
  summarise(age = termed_age) %>%
  unique

tba <- cbind(termed_by_age, action=rep("termed", nrow(termed_by_age)))
```

Figure 36 Code for constructing the data frame with the value of terminated age

```
> sample_frac(tba, .1)
# A tibble: 146 x 5
# Groups:   year, BUSINESS_UNIT [14]
    year BUSINESS_UNIT EmployeeID   age action
   <dbl> <fct>              <int> <dbl> <chr>
 1  2006 STORES              4140    60 termed
 2  2006 STORES              2578    64 termed
 3  2006 STORES              2540    64 termed
 4  2006 STORES              2440    64 termed
 5  2006 STORES              2467    64 termed
 6  2006 STORES              4199    60 termed
 7  2006 STORES              2348    64 termed
 8  2006 STORES              4419    60 termed
 9  2006 STORES              5604    33 termed
10  2006 STORES              4230    60 termed
# ... with 136 more rows
```

Figure 37 Output for Figure 36

Analysis 2-3: Find the age trend based on business unit.

To plot both hired and terminated, we can use *rbind* to combine the two data frames into one.

```r
htba <- rbind(hba, tba)

age_trend <- htba %>%
    group_by(action, BUSINESS_UNIT, year) %>%
    summarise(mean_age = mean(age)) %>%
    mutate(BU_action = factor(paste(action, BUSINESS_UNIT))) %>%
    ggplot(aes(x=year, y=mean_age)) +
    geom_line(aes(color=BU_action), size=1.2) +
    labs(title = "Age trend")
```

Figure 38 Code for plotting the age average for each business unit yearly based on hired and termination using line plot

```
> sample_frac(htba, .5)
# A tibble: 14 x 5
# Groups:   year, BUSINESS_UNIT, EmployeeID [14]
    year BUSINESS_UNIT EmployeeID   age action
   <dbl> <fct>              <int> <dbl> <chr>
 1  2006 STORES              6540    25 termed
 2  2008 STORES              6914    24 hired
 3  2008 STORES              6918    24 termed
 4  2010 STORES              7388    23 termed
 5  2010 STORES              7416    22 hired
 6  2011 STORES              7659    21 termed
 7  2011 STORES              7741    21 hired
 8  2011 STORES              7801    21 termed
 9  2012 STORES              8036    20 hired
10  2013 STORES              8181    19 hired
11  2013 STORES              8223    19 hired
12  2013 STORES              8226    19 termed
13  2013 STORES              8264    19 termed
14  2013 STORES              8296    19 hired
```

Figure 39 Sample output for Figure 38

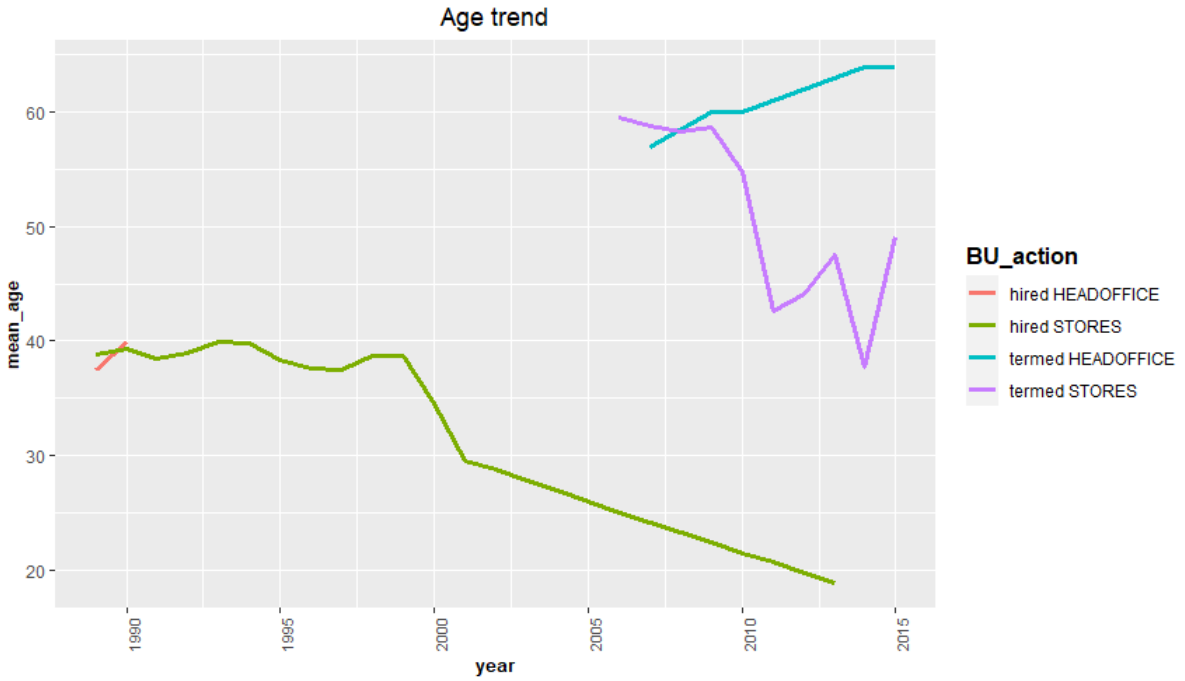


Figure 40 Line plot showing the age mean for each business unit based on hired and termination yearly

From this data frame, we can plot how the age were distributed throughout the years into a scatterplot using *ggplot*, *geom_point*, and *stat_smooth*.

```
age_dist <- htba %>%
  group_by(action, BUSINESS_UNIT, year) %>%
  mutate(BU_action = factor(paste(action, BUSINESS_UNIT))) %>%
  ggplot(aes(x=year, y=age)) +
  geom_point(aes(color=age), size=2.5) +
  scale_color_gradient("age", low = "blue", high = "red") +
  stat_smooth(method = lm, fullrange = TRUE, aes(fill=BU_action)) +
  scale_x_continuous(breaks = seq(min(htba$year), max(htba$year), 3)) +
  scale_fill_discrete(name = "Business Unit\non action") +
  labs(title = "Age Distribution") +
  facet_wrap(~BU_action)
```

Figure 41 Code for plotting the age distribution based on business unit using scatterplot

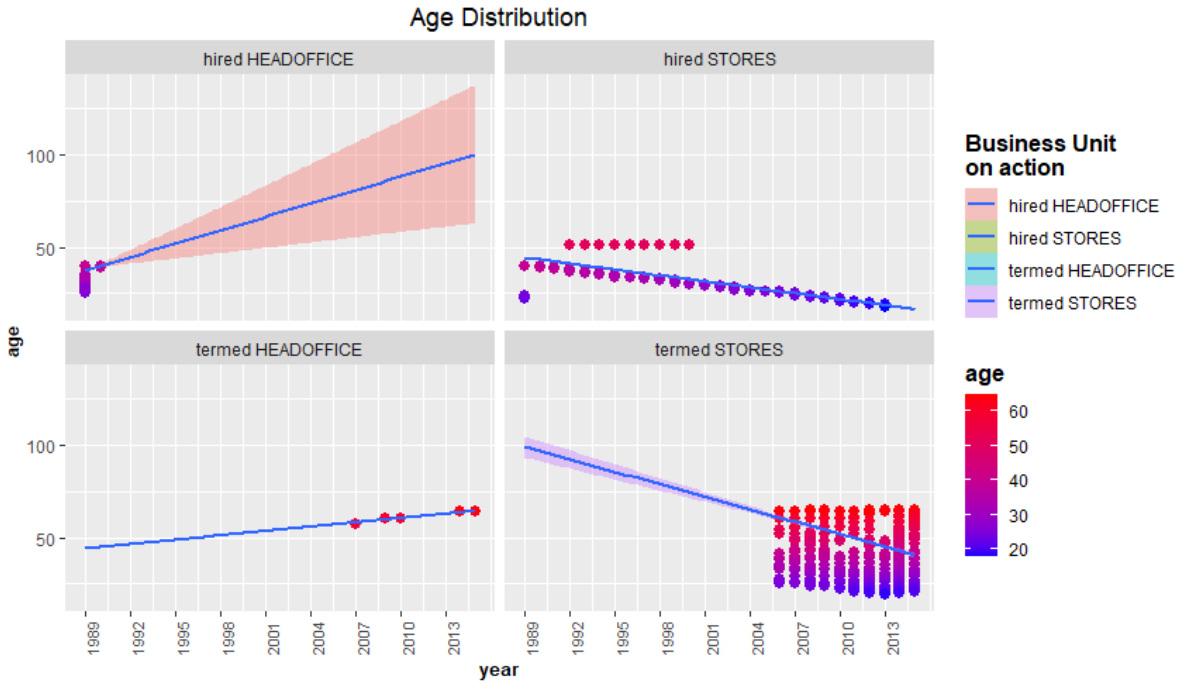


Figure 42 Scatterplot showing age distribution based on Business Unit and hired/termination

Next, we will see the genders average age on each business unit each year with the help of *facet_wrap* to display each year on separate graph.

```
term_test <- term_by_var(
  emp_table,
  c("year", "termreason_desc", "BUSINESS_UNIT", "gender_short")) %>%
  filter(termreason_desc != "Not Applicable") %>%
  ungroup %>%
  mutate(BU_gender = factor(paste(gender_short, BUSINESS_UNIT))) %>%
  group_by(year, termreason_desc, BU_gender) %>%
  summarise(mean_age = mean(age)) %>%
  ggplot(aes(x=termreason_desc, y=mean_age, fill=BU_gender)) +
  geom_bar(stat="Identity", position = "dodge", color="black") +
  scale_fill_discrete(name = "Business Unit\nby gender") +
  labs(title = "Terminated employees yearly age average\nbased on Termination
Reason by Business Unit and Gender",
       x = "Termination Reason", y = "Mean Age") +
  theme(axis.text.x = element_text(angle=45, vjust=0.5)) +
  facet_wrap(~year)
```

Figure 43 Code for plotting the average age of terminated employees yearly by termination reason, Business Unit and gender

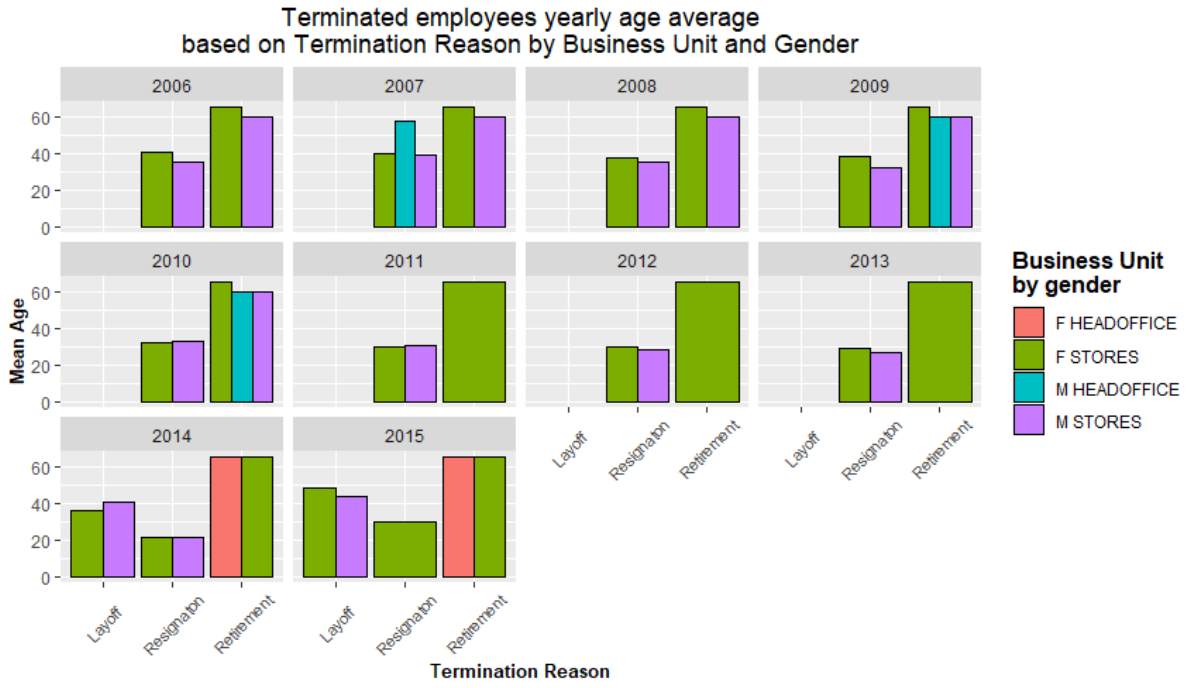


Figure 44 Bar chart showing the average age of terminated employees based on termination reason, gender and business unit

```
hire_test <- hire_by_var(
    emp_table,
    c("year", "BUSINESS_UNIT", "gender_full")
  ) %>%
  mutate(hired_age = year_diff(orighiredate_key, birthdate_key)) %>%
  group_by(year, BUSINESS_UNIT, gender_full, EmployeeID) %>%
  summarise(hired_age = min(hired_age)) %>%
  ungroup %>%
  group_by(year, BUSINESS_UNIT, gender_full) %>%
  summarise(mean_hired = mean(hired_age)) %>%
  ggplot(aes(x=BUSINESS_UNIT, y=mean_hired, fill=gender_full)) +
  geom_bar(stat="Identity", position = "dodge", color="black") +
  scale_fill_discrete(name = "Gender") +
  labs(title = "Hired employees yearly age average\nbased on Business Unit and
Gender",
       x = "Business Unit", y = "Mean Age") +
  theme(axis.text.x = element_text(angle=45, vjust=0.5)) +
  facet_wrap(~year)
```

Figure 45Code for plotting the average age of hired employees yearly by Business Unit and gender

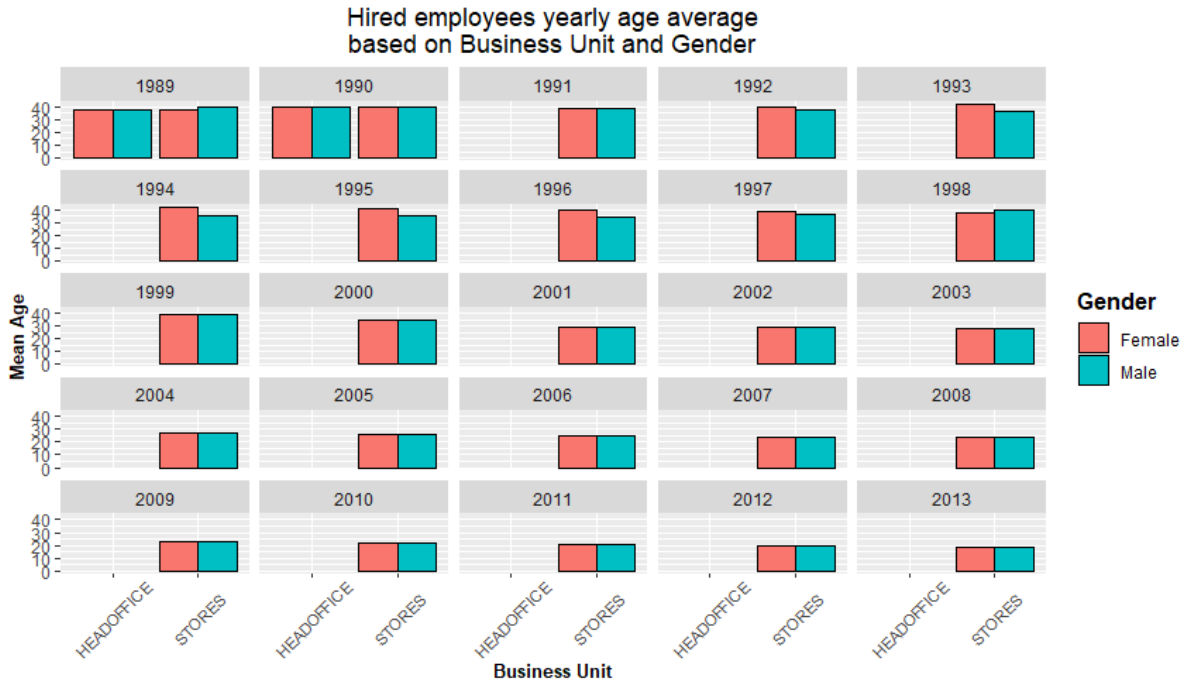


Figure 46 Bar chart showing the average age of hired employees yearly based on business unit and gender

Analysis 2-4: Find the overall age statistics.

In this analysis, we will look at the age as overall. We will group the data by business unit and action (hired/termed). We will add a label showing the mean for each group with the help of *stat_summary*. To get the mean in certain precision, we use round with the parameter "*..y..*" which is a special variables for *stat_summary* which stores the mean value.

```
overall_age_box <- htba %>%
    mutate(BU_action = factor(paste(BUSINESS_UNIT, action))) %>%
    ggplot(aes(x=BU_action, y=age, fill=BU_action)) +
    geom_boxplot() +
    stat_summary(fun=mean, geom="point", aes(shape="mean"),
                 size=3, show.legend=TRUE) +
    stat_summary(fun=mean, geom="text",
                 hjust = -0.5, aes(label=round(..y.., digits=1)),
                 show.legend = FALSE) +
    scale_shape_manual("Statistics", values=c("mean"="x")) +
    scale_fill_discrete(name = "Business Unit\non\naction") +
    labs(title = "Age statistics by Business Unit",
         x = "Business Unit on Action") +
    theme(axis.text.x = element_text(angle=0))
```

Figure 47 Code for plotting the overall age statistics using boxplot

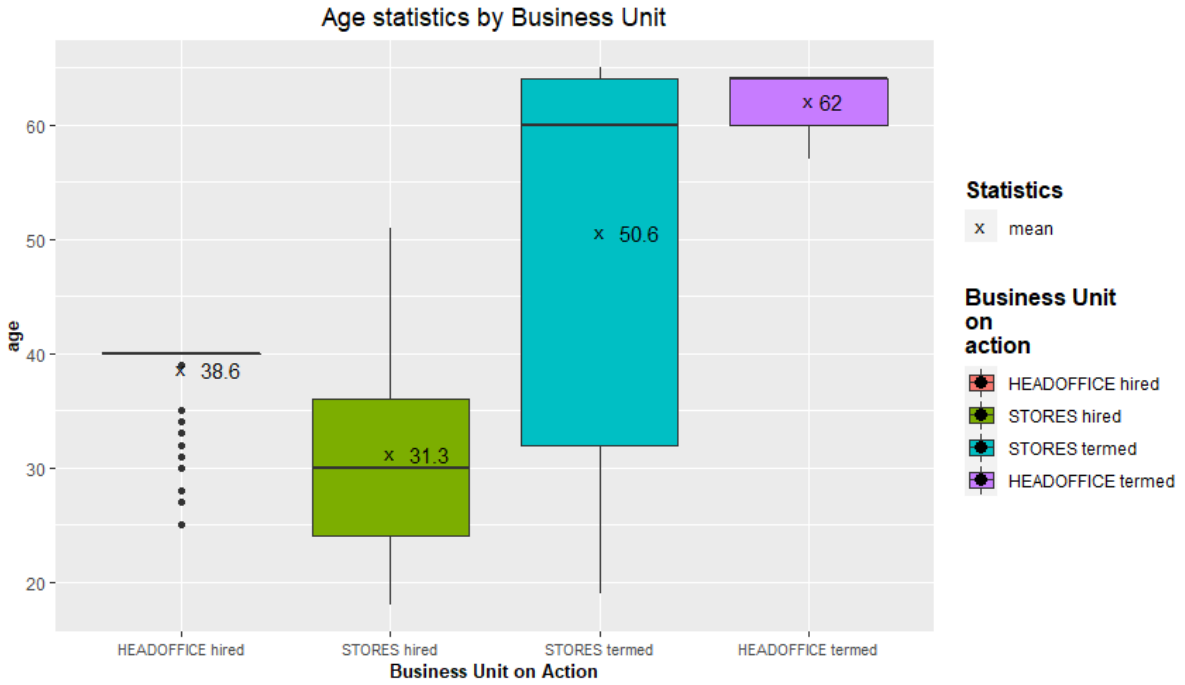


Figure 48 Boxplot showing the overall age statistics based on business unit

Next is analysis based on age group. We will create a new column with the value of the corresponding age group using *case_when*. Then, we will pass the column *agegroup* into *pie3D*. Since it only accepts vector as parameter, we will subset the data frame using double square bracket, since single square bracket retains the original data type, i.e., data frame.

```
hired_agegroup <- hired_by_age %>%
    mutate(agegroup = case_when(
        age < 20 ~ '<20',
        age < 30 ~ '20-29',
        age < 40 ~ '30-39',
        age < 50 ~ '40-49',
        age < 60 ~ '50-59',
        age < 70 ~ '60-69',
        age < 80 ~ '70-79',
        age < 90 ~ '80-89',
    )) %>%
    group_by(agegroup) %>%
    summarise(n = n())

overall_age_hired_pie <- pie3D(hired_agegroup[["n"]],
      labels = factor(hired_agegroup$agegroup),
      main = "Overall hired agegroup")
```

Figure 49 Code for adding a column for age group and plot it in 3D pie chart for hired employees

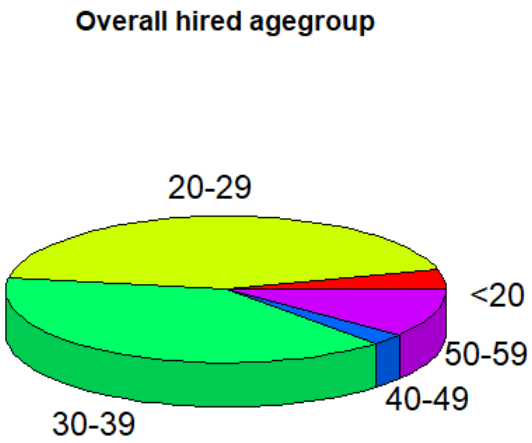


Figure 50 3D pie chart showing the portion of hired age group

```
termed_agegroup <- termed_by_age %>%
    mutate(agegroup = case_when(
        age < 20 ~ '<20',
        age < 30 ~ '20-29',
        age < 40 ~ '30-39',
        age < 50 ~ '40-49',
        age < 60 ~ '50-59',
        age < 70 ~ '60-69',
        age < 80 ~ '70-79',
        age < 90 ~ '80-89',
    )) %>%
    group_by(agegroup) %>%
    summarise(n = n())

overall_age_termed_pie <- pie3D(termed_agegroup[["n"]],
      labels = factor(termed_agegroup$agegroup),
      main = "Overall termed agegroup")
```

Figure 51 Code for adding a column for age group and plotting it in 3D pie chart for terminated employees

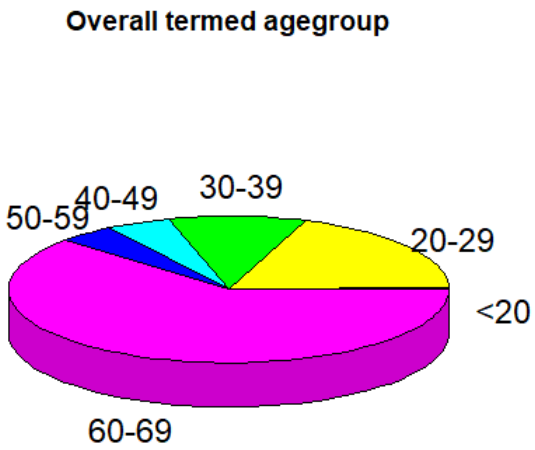


Figure 52 3D pie chart showing the overall terminated age group

Conclusion:

The average age hired for *Stores* unit were favouring younger people with neutral preference over gender and the gender preference applies to *Headoffice* unit as well. As for terminated employees, most retirements were on higher age group. In terms of business unit, *Stores* unit has an equally age termination distribution. However, if seen as overall, it is more dominated by those in the late 60s.

Question 3: What are the characteristics of employees based on termination reason?

Analysis 3-1: Find the number of termination reasons based on gender.

We will utilize the function created earlier to now get the terminated employees data based on termination reasons and gender. Once again, we will only take employees which has been terminated by subsetting out *Not Applicable* data.

```
term_gender_count <- term_by_var(emp_table,
                                 c("termreason_desc", "gender_short")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n())

pie_gender_term <- pie(term_gender_count$n,
    labels = paste0(term_gender_count$termreason_desc,
                    " (", term_gender_count$gender_short, "): ",
                    term_gender_count$n),
    main = "Gender Chart on Termination Reason")
```

Figure 53 Code for plotting the number of termination reasons based on gender

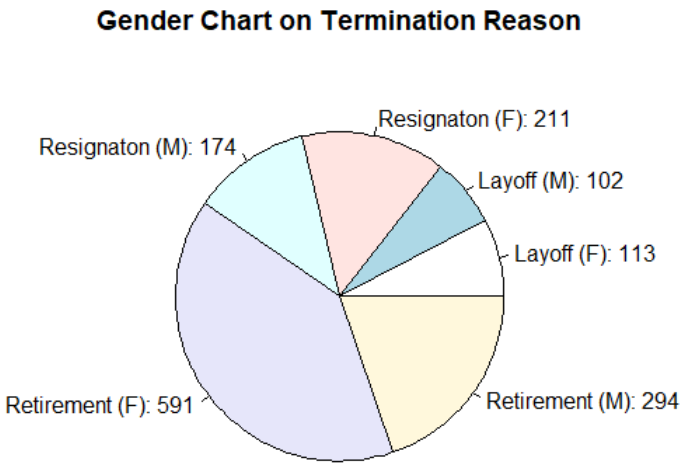


Figure 54 Pie chart showing the number of termination reasons based on gender

Analysis 3-2: Find the age statistics for each termination reason.

Again, using the previous function, we will now group by termination reason and filter out some data. Next, we plot our data into boxplot with additional labelling for mean, min, and max age.

```
term_age <- term_by_var(emp_table, c("termreason_desc")) %>%
  filter(termreason_desc != "Not Applicable") %>%
  summarise(age)

means <- term_age %>% summarise(age = mean(age))

term_age %>%
  ggplot(aes(x = termreason_desc, y = age, color=termreason_desc)) +
  geom_boxplot() +
  stat_summary(fun=mean, geom="point", aes(shape="mean"),
               size=3, show.legend=TRUE) +
  stat_summary(fun=min, geom="point", aes(shape="min"),
               size=3, show.legend=TRUE) +
  stat_summary(fun=max, geom="point", aes(shape="max"),
               size=3, show.legend=TRUE) +
  stat_summary(fun=mean, geom="text",
               hjust = -0.5, aes(label=round(..y.., digits=1)),
               show.legend = FALSE) +
  stat_summary(fun=min, geom="text",
               vjust = 1.4, aes(label=round(..y.., digits=1)),
               show.legend = FALSE) +
  stat_summary(fun=max, geom="text",
               vjust = -0.7, aes(label=round(..y.., digits=1)),
               show.legend = FALSE) +
  scale_shape_manual("Statistics", values=c("mean"="x", "min"="m", "max"="M")) +
  scale_color_discrete(name = "Term Reason") +
  labs(title = "Age overall statistics on termination reason",
       x = "Termination Reason") +
  theme(axis.text.x = element_text(angle=0))
```

Figure 55 Code for plotting the age statistics based on termination reasons using boxplot

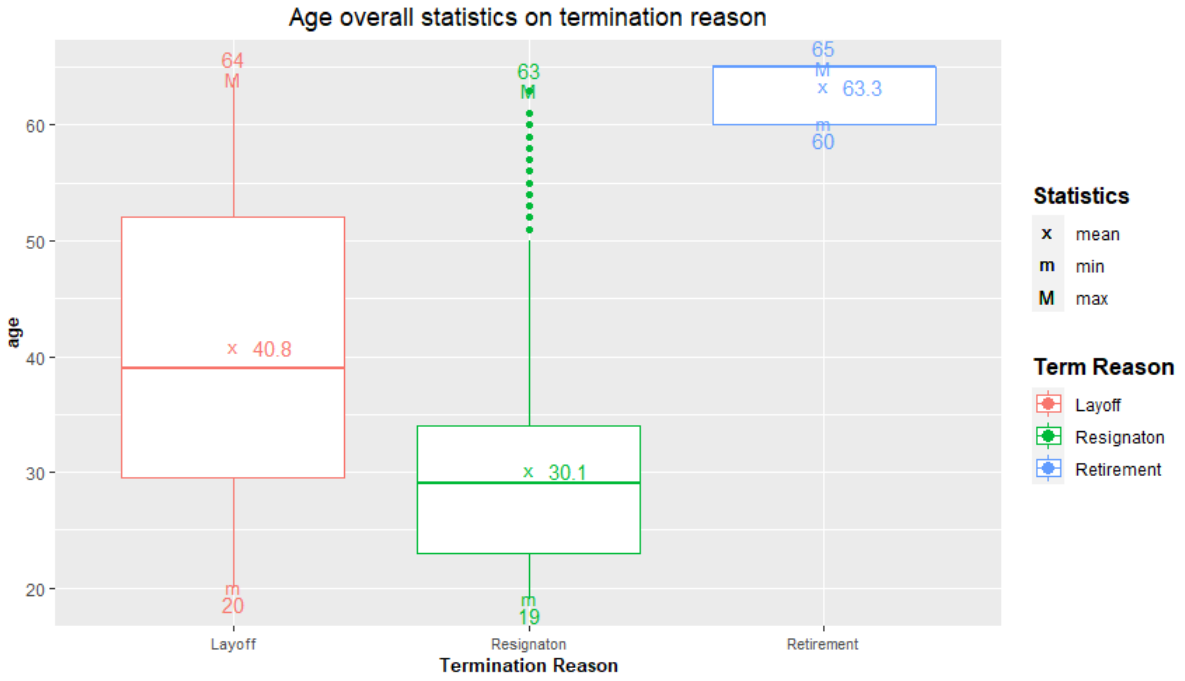


Figure 56 Boxplot showing the age statistic for each termination reason

Analysis 3-3: Find the number of termination reasons based on business unit.

```
term_unit <- term_by_var(emp_table, c("termreason_desc", "BUSINESS_UNIT")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n())
```

Figure 57 Code for creating data frame with number of terminations based on business unit and reasons

```
> term_unit
# A tibble: 5 x 3
# Groups:   termreason_desc [3]
  termreason_desc BUSINESS_UNIT     n
  <fct>           <fct>         <int>
1 Layoff          STORES          215
2 Resignaton      HEADOFFICE        1
3 Resignaton      STORES          384
4 Retirement      HEADOFFICE       68
5 Retirement      STORES          817
```

Figure 58 Output for Figure 57

Analysis 3-4: Find the top and the least terminated department based on termination reason.

After grouping the data by termination reason and department name, we will count the number of occurrences by summarising it. To get the department name with highest termination number based on each reason, we will ungroup the data and then regroup it by termination reason. Next, using *slice(which.max(n))*, we can get the entry with the max number on each group. The same goes for min number.

```
term_dept_n <- term_by_var(emp_table,
                           c("termreason_desc", "department_name")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n()) %>%
  ungroup %>%
  group_by(termreason_desc)

term_dept_n %>% slice(which.max(n))
term_dept_n %>% slice(which.min(n))
```

Figure 59 Code for finding the top and least terminated department for each reason

```
> term_dept_n %>% slice(which.max(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc department_name      n
  <fct>           <fct>            <int>
1 Layoff          Customer Service    70
2 Resignaton      Customer Service   179
3 Retirement      Meats              317
> term_dept_n %>% slice(which.min(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc department_name      n
  <fct>           <fct>            <int>
1 Layoff          Store Management     6
2 Resignaton      HR Technology        1
3 Retirement      Legal                3
```

Figure 60 Output for the top and least terminated department for each reason

Analysis 3-5: Find the top and the least terminated city based on termination reasons.

This analysis uses the same logic as previous analysis.

```r
term_city_n <- term_by_var(emp_table, c("termreason_desc", "city_name")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n()) %>%
  ungroup %>%
  group_by(termreason_desc) %>%
  arrange(n, .by_group = TRUE)

term_city_n %>% slice(which.max(n))
term_city_n %>% slice(which.min(n))
```

Figure 61 Code for finding the top and least terminated city for each reason

```
> term_city_n %>% slice(which.max(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc city_name       n
  <fct>           <fct>       <int>
1 Layoff          Fort Nelson    39
2 Resignaton      Vancouver      71
3 Retirement      Vancouver     219
> term_city_n %>% slice(which.min(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc city_name       n
  <fct>           <fct>       <int>
1 Layoff          Blue River      1
2 Resignaton      Aldergrove      1
3 Retirement      Ocean Falls     1
```

Figure 62 Output for the top and least terminated city for each reason

Analysis 3-6: Find the top and the least terminated job title based on termination reasons.

This analysis uses the same logic as previous analysis.

```r
term_job_n <- term_by_var(emp_table, c("termreason_desc", "job_title")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n()) %>%
  ungroup %>%
  group_by(termreason_desc) %>%
  arrange(n, .by_group = TRUE)

term_job_n %>% slice(which.max(n))
term_job_n %>% slice(which.min(n))
```

Figure 63 Code for finding the top and least terminated job title for each reason

```
> term_job_n %>% slice(which.max(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc job_title       n
  <fct>           <fct>       <int>
1 Layoff          Cashier        67
2 Resignaton      Cashier       179
3 Retirement      Meat Cutter   298
> term_job_n %>% slice(which.min(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc job_title                   n
  <fct>           <fct>                   <int>
1 Layoff          Customer Service Manager     3
2 Resignaton      HRIS Analyst                 1
3 Retirement      Dairy Manager                1
```

Figure 64 Output for the top and least terminated job title for each reason

Conclusion

Most employees leaving the company was due to retirement, and for each reason, female seems to contribute more than male. In terms of business unit, there is yet to be employee being laid off from *Headoffice* unit. Most employees retired at age 63, but varied age for both laid off and resigned employees. As for department, job title, and city, employees from Customer Service, Cashier mostly being laid off and resign, and the city Vancouver has the highest number of retirements.

Question 4: What are the characteristics of hired employees?

Analysis 4-1: Find the top 20% of hired employees based on job title.

Since our initial assumption is *"Every employee only has one job throughout their career in the company,"* we just need to have a data frame with a single entry for every employee ID. This can be done by taking the minimal length of service for each employee. After we get the number of employees for each job, we will filter out the top 20% by using checking if the number is equal or greater than 20th percentile which can be achieved by using *quantile* with *0.8* as the prob.

```
hire_n_job_title <- hire_by_var(emp_table, c("EmployeeID", "job_title")) %>%
  slice(which.min(length_of_service)) %>%
  group_by(job_title) %>%
  summarise(n = n())

hire_n_job_title %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
```

Figure 65 Code for finding the top 20% hired job

```
> hire_n_job_title %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
# A tibble: 10 x 2
   job_title           n
   <fct>           <int>
 1 Meat Cutter      1218
 2 Cashier          1158
 3 Dairy Person     1032
 4 Produce Clerk    1027
 5 Baker             865
 6 Shelf Stocker     704
 7 Store Manager      35
 8 Meats Manager      34
 9 Bakery Manager     33
10 Produce Manager    33
```

Figure 66 Output of the top 20% hired job

Analysis 4-2: Find the top 20% of hired employees based on department.

Similar logic as previous analysis with different grouping.

```r
hire_n_department_name <- hire_by_var(emp_table, c("EmployeeID",
"department_name")) %>%
  slice(which.min(length_of_service)) %>%
  group_by(department_name) %>%
  summarise(n = n())

hire_n_department_name %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
```

Figure 67 Code for finding the top 20% hired department

```
> hire_n_department_name %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
# A tibble: 5 x 2
  department_name      n
  <fct>            <int>
1 Meats             1252
2 Customer Service  1190
3 Produce           1060
4 Dairy             1033
5 Bakery             898
```

Figure 68 Output for the top 20% hired department

Analysis 4-3: Find the top 20% of hired employees based on city.

Similar logic as previous analysis with different grouping.

```r
hire_n_city_name <- hire_by_var(emp_table, c("EmployeeID", "city_name")) %>%
  slice(which.min(length_of_service)) %>%
  group_by(city_name) %>%
  summarise(n = n())

hire_n_city_name %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
```

Figure 69 Code for finding the top 20% hired city

```
> hire_n_city_name %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
# A tibble: 8 x 2
  city_name          n
  <fct>          <int>
1 Vancouver       1392
2 Victoria         624
3 Nanaimo          481
4 New Westminster  403
5 Kelowna          305
6 Kamloops         267
7 Prince George    264
8 Burnaby          258
```

Figure 70 Output for the top 20% hired city

Analysis 4-4: Find the top 20% of hired job title based on gender.

In this case, to view the data in better format, we will spread the column *gender_short* into new column using *spread*.

```
hire_n_gender_job <- hire_by_var(emp_table, c("EmployeeID")) %>%
  slice(which.min(length_of_service)) %>%
  group_by(gender_short, job_title) %>%
  summarise(n = n())

spread_format <- hire_n_gender_job %>%
  spread(gender_short, n, fill=0)

colnames(spread_format) <- c("job", "Female", "Male")
spread_format <- cbind(spread_format,
                       total=spread_format$Female + spread_format$Male)

spread_format %>%
  filter(total >= quantile(total, 0.8)) %>%
  arrange(desc(total))
```

Figure 71 Code for finding the top 20% hired job based on gender

```
> hire_n_gender_job
# A tibble: 70 x 3
# Groups:   gender_short [2]
   gender_short job_title                    n
   <fct>        <fct>                    <int>
 1 F            Accounting Clerk             5
 2 F            Accounts Payable Clerk       2
 3 F            Accounts Receiveable Clerk   2
 4 F            Auditor                      1
 5 F            Baker                      438
 6 F            Bakery Manager              18
 7 F            Benefits Admin               3
 8 F            Cashier                    573
 9 F            CHief Information Officer    1
10 F            Compensation Analyst         1
# ... with 60 more rows
```

Figure 72 Output for the number of hired employees for each job based on gender in long format

```
> spread_format %>%
+   filter(total >= quantile(total, 0.8)) %>%
+   arrange(desc(total))
               job Female Male total
1      Meat Cutter    720  498  1218
2          Cashier    573  585  1158
3     Dairy Person    524  508  1032
4    Produce Clerk    537  490  1027
5            Baker    438  427   865
6    Shelf Stocker    341  363   704
7    Store Manager     18   17    35
8    Meats Manager     18   16    34
9   Bakery Manager     18   15    33
10 Produce Manager     16   17    33
```

Figure 73 Output for the top 20% hired job based on gender in wide format

Conclusion

The hired employees are mostly meat cutters and cashier from meat and customer service department respectively. Moreover, Vancouver seems to be recruiting the most out of the other cities. As for the gender, there is no significant difference between female and male.

Question 5: What are the characteristics which affect the length of service for terminated employees?

Analysis 5-1: Find the average age for the top 10% longest and shortest service.

In this analysis, we will get the newest length of service by grouping the data by employee ID and get the maximum length of service. Next, we filter out employee by their status and rename the column *length_of_service* into *service*. Then, we define two functions. The first function is to get the top *x*% of the data grouped by *byCol*. And the other functions is just the opposite, i.e., last *x*%.

```
service_df <- emp_table %>%
  group_by(EmployeeID) %>%
  slice(which.max(length_of_service))

term_service <- service_df %>%
  filter(STATUS == "TERMINATED") %>%
  rename(service = length_of_service)

top_n_los <- function(df, x, byCol) {
  df %>%
    ungroup %>%
    filter(service >= quantile(service, x)) %>%
    group_by_at(byCol) %>%
    summarise(n = n()) %>%
    arrange(desc(n))
}

last_n_los <- function(df, x, byCol) {
  df %>%
    ungroup %>%
    filter(service <= quantile(service, x)) %>%
    group_by_at(byCol) %>%
    summarise(n = n()) %>%
    arrange(desc(n))
}
```

Figure 74 Function to get the top and last x% based on attritions

```
term_service %>%
  ungroup %>%
  filter(service >= quantile(service, 0.9)) %>%
  pull(age) %>%
  summary

term_service %>%
  ungroup %>%
  filter(service <= quantile(service, 0.1)) %>%
  pull(age) %>%
  summary
```

Figure 75 Code for finding the average age top 10% longest and shortest service

```
> term_service %>%
+   ungroup %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   pull(age) %>%
+   summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  61.00   64.00   65.00   64.56   65.00   65.00
> term_service %>%
+   ungroup %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   pull(age) %>%
+   summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  19.00   21.00   21.50   22.04   23.00   28.00
```

Figure 76 Output for the average age in top 10% longest and shortest service

Analysis 5-2: Find the gender count for the top 10% longest and shortest service.

```
top_n_los(term_service, 0.9, "gender_short")

last_n_los(term_service, 0.1, "gender_short")
```

Figure 77 Code for finding the top and last 10% service based on gender

```
> top_n_los(term_service, 0.9, "gender_short")
# A tibble: 2 x 2
  gender_short     n
  <fct>        <int>
1 F               87
2 M               12
>
> last_n_los(term_service, 0.1, "gender_short")
# A tibble: 2 x 2
  gender_short     n
  <fct>        <int>
1 M               61
2 F               37
```

Figure 78 Output for the top and last 10% service based on gender

Analysis 5-3: Find the average length of service based on the gender.

To compute the average, we can use *aggregate* to save us some line of code. The parameters meaning is to compute the *service* mean of the group *gender_short*. The dot (.) parameter is the placeholder for previous code output piped in. Similar results can be achieved by using *group_by* and *summarise* function.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ gender_short, ., mean) %>%
  arrange(desc(service))
```

Figure 79 Code for finding the average length of service based on gender

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ gender_short, ., mean) %>%
+   arrange(desc(service))
  gender_short   service
1            F 12.609541
2            M  9.445026
```

Figure 80 Output for the average length of service by gender

Analysis 5-4: Find the business unit count for the top 10% longest and shortest service.

```
top_n_los(term_service, 0.9, "BUSINESS_UNIT")

last_n_los(term_service, 0.1, "BUSINESS_UNIT")
```

Figure 81 Code for finding the top and last 10% service length by business unit

```
> top_n_los(term_service, 0.9, "BUSINESS_UNIT")
# A tibble: 2 x 2
  BUSINESS_UNIT     n
  <fct>         <int>
1 STORES           83
2 HEADOFFICE       16
>
> last_n_los(term_service, 0.1, "BUSINESS_UNIT")
# A tibble: 1 x 2
  BUSINESS_UNIT     n
  <fct>         <int>
1 STORES           98
```

Figure 82 Output for the top and last 10% service length by business unit

Analysis 5-5: Find the average length of service based on the business unit.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ BUSINESS_UNIT, ., mean) %>%
  arrange(desc(service))
```

Figure 83 Code for finding the average length of service by business unit

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ BUSINESS_UNIT, ., mean) %>%
+   arrange(desc(service))
  BUSINESS_UNIT  service
1    HEADOFFICE 22.22222
2        STORES 11.01520
```

Figure 84 Output for the average length of service by business unit

Analysis 5-6: Find the city count for the top 1% longest and shortest service.

```
top_n_los(term_service, 0.99, "city_name")

last_n_los(term_service, 0.01, "city_name")
```

Figure 85 Code for finding the top and last 1% service length by city

```
> top_n_los(term_service, 0.99, "city_name")
# A tibble: 18 x 2
   city_name           n
   <fct>           <int>
 1 Vancouver          21
 2 Victoria           12
 3 Kamloops            3
 4 New Westminster     3
 5 Prince George       3
 6 Terrace             3
 7 Aldergrove          2
 8 Dawson Creek        2
 9 Nanaimo             2
10 Burnaby             1
11 Cranbrook           1
12 Fort St John        1
13 Kelowna             1
14 Quesnel             1
15 Squamish            1
16 Surrey              1
17 Vernon              1
18 West Vancouver      1
```

Figure 86 Output for the top 1% city on service length

```
> last_n_los(term_service, 0.01, "city_name")
# A tibble: 11 x 2
   city_name           n
   <fct>           <int>
 1 Trail               2
 2 Vancouver           2
 3 Victoria            2
 4 Abbotsford          1
 5 Chilliwack          1
 6 Fort St John        1
 7 Grand Forks         1
 8 Kelowna             1
 9 Nanaimo             1
10 New Westminister    1
11 Prince George       1
```

Figure 87 Output for the last 1% city on service length

Analysis 5-7: Find the best and worst average length of service based on the city.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ city_name, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

term_service %>%
  ungroup %>%
  aggregate(service ~ city_name, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 88 Code for finding the best and worst average service length by city

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ city_name, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
    city_name  service
1 Ocean Falls 19.66667
2  Dease Lake 17.00000
3  Aldergrove 15.71429
4      Vernon 13.69231
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ city_name, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
         city_name  service
1 New Westminister 8.794872
2       Chilliwack 7.375000
3          Langley 7.307692
4           Nelson 6.666667
```

Figure 89 Output for the best and worst average service length by city

Analysis 5-8: Find the job title count for the top 1% longest and shortest service.

```
top_n_los(term_service, 0.99, "job_title")

last_n_los(term_service, 0.01, "job_title")
```

Figure 90 Code for finding the top and last 1% service length by job title

```
> top_n_los(term_service, 0.99, "job_title")
# A tibble: 10 x 2
   job_title                   n
   <fct>                   <int>
 1 Meat Cutter                22
 2 Produce Manager             6
 3 Store Manager               6
 4 Accounting Clerk            5
 5 Bakery Manager              5
 6 Meats Manager               5
 7 Customer Service Manager    4
 8 Dairy Person                3
 9 Processed Foods Manager     3
10 Investment Analyst          1
>
> last_n_los(term_service, 0.01, "job_title")
# A tibble: 4 x 2
  job_title         n
  <fct>         <int>
1 Cashier           9
2 Dairy Person      2
3 Shelf Stocker     2
4 Baker             1
```

Figure 91 Output for the top and last 1% service length by job title

Analysis 5-9: Find the best and worst average length of service based on the job title.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ job_title, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

term_service %>%
  ungroup %>%
  aggregate(service ~ job_title, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 92 Code for finding the best and worst average service length by job title

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ job_title, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
         job_title service
1 Accounting Clerk      25
2    Dairy Manager      24
>
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ job_title, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
      job_title  service
1 Shelf Stocker 6.208333
2       Cashier 4.052326
```

Figure 93 Output for the best and worst average service length by job title

Analysis 5-10: Find the department count for the top 1% longest and shortest service.

```
top_n_los(term_service, 0.99, "department_name")

last_n_los(term_service, 0.01, "department_name")
```

Figure 94 Code for finding the top and last 1% service length by department

```
> top_n_los(term_service, 0.99, "department_name")
# A tibble: 9 x 2
  department_name      n
  <fct>            <int>
1 Meats               27
2 Produce              6
3 Store Management     6
4 Accounting           5
5 Bakery               5
6 Customer Service     4
7 Dairy                3
8 Processed Foods      3
9 Investment           1
>
> last_n_los(term_service, 0.01, "department_name")
# A tibble: 4 x 2
  department_name      n
  <fct>            <int>
1 Customer Service     9
2 Dairy                2
3 Processed Foods      2
4 Bakery               1
```

Figure 95 Output for the top and last 1% service length by department

Analysis 5-11: Find the best and worst average length of service based on the department.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ department_name, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

term_service %>%
  ungroup %>%
  aggregate(service ~ department_name, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 96 Code for finding the best and worst average service length by department

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ department_name, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
   department_name service
1       Accounting 25.0000
2 Store Management 23.5625
>
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ department_name, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
   department_name  service
1  Processed Foods 9.101695
2 Customer Service 5.038462
```

Figure 97Output for the best and worst average service length by department

Analysis 5-12: Find the store name count for the top 1% longest and shortest service.

```
top_n_los(term_service, 0.99, "store_name")

last_n_los(term_service, 0.01, "store_name")
```

Figure 98 Code for finding the top and last 1% service length by store name

```
> top_n_los(term_service, 0.99, "store_name")
# A tibble: 21 x 2
   store_name     n
        <int> <int>
 1         37    12
 2         35     6
 3         42     6
 4         41     5
 5         45     4
 6         15     3
 7         21     3
 8         26     3
 9         32     3
10          2     2
# ... with 11 more rows
>
> last_n_los(term_service, 0.01, "store_name")
# A tibble: 12 x 2
   store_name     n
        <int> <int>
 1         33     2
 2         46     2
 3          1     1
 4          6     1
 5         12     1
 6         13     1
 7         16     1
 8         18     1
 9         20     1
10         26     1
11         43     1
12         44     1
```

Figure 99 Output for the top and last 1% service length by store name

Analysis 5-13: Find the best and worst average length of service based on the store name.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ store_name, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

term_service %>%
  ungroup %>%
  aggregate(service ~ store_name, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 100 Code for finding the best and worst average service length by store name

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ store_name, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
  store_name  service
1         45 24.66667
2         41 23.28571
3         23 19.66667
4         10 17.00000
5          2 15.71429
>
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ store_name, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
  store_name  service
1         17 7.307692
2         19 6.666667
3         46 5.870968
4         43 4.437500
5         44 1.800000
```

Figure 101 Output for the best and worst average service length by store name

Analysis 5-14: Find the term reason count for the top 10% longest and shortest service.

```
top_n_los(term_service, 0.9, "termreason_desc")

last_n_los(term_service, 0.1, "termreason_desc")
```

Figure 102 Code for finding the top and last 10% service length by termination reason

```
> top_n_los(term_service, 0.9, "termreason_desc")
# A tibble: 2 x 2
  termreason_desc     n
  <fct>           <int>
1 Retirement         73
2 Layoff             26
>
> last_n_los(term_service, 0.1, "termreason_desc")
# A tibble: 2 x 2
  termreason_desc     n
  <fct>           <int>
1 Resignaton         94
2 Layoff              4
```

Figure 103 Output for the top and last 10% service length by termination reason

Analysis 5-15: Find the average length of service based on the term reason.

```
term_service %>%
  ungroup %>%
  aggregate(service ~ termreason_desc, ., mean) %>%
  arrange(desc(service))
```

Figure 104 Code for finding the average service length by termination reason

```
> term_service %>%
+   ungroup %>%
+   aggregate(service ~ termreason_desc, ., mean) %>%
+   arrange(desc(service))
  termreason_desc   service
1      Retirement 13.962891
2          Layoff 11.948837
3      Resignaton  4.647059
```

Figure 105 Output for the average length of service by termination reason

Analysis 5-16: Find the age group count for the top 10% longest and shortest service.

```
term_ageG_service <-  term_service %>%
  mutate(agegroup = case_when(
    age < 20 ~ '<20',
    age < 30 ~ '20-29',
    age < 40 ~ '30-39',
    age < 50 ~ '40-49',
    age < 60 ~ '50-59',
    age < 70 ~ '60-69',
    age < 80 ~ '70-79',
    age < 90 ~ '80-89',
))

term_ageG_service %>% top_n_los(0.9, "agegroup")
term_ageG_service %>% last_n_los(0.1, "agegroup")
```

Figure 106 Code for finding the top and last 10% service length by age group

```
> term_ageG_service %>% top_n_los(0.9, "agegroup")
# A tibble: 1 x 2
  agegroup     n
  <chr>    <int>
1 60-69       99
> term_ageG_service %>% last_n_los(0.1, "agegroup")
# A tibble: 2 x 2
  agegroup     n
  <chr>    <int>
1 20-29       94
2 <20          4
```

Figure 107 Output for the top and last 10% service length by age group

Analysis 5-17: Find the average length of service based on the age group.

```
term_ageG_service %>%
  aggregate(service ~ agegroup, ., mean) %>%
  arrange(desc(service))
```

Figure 108 Code for finding the average length of service by age group

```
> term_ageG_service %>%
+   aggregate(service ~ agegroup, ., mean) %>%
+   arrange(desc(service))
  agegroup   service
1    50-59 18.358491
2    60-69 14.487037
3    40-49 12.672727
4    30-39  6.962963
5    20-29  1.937888
6      <20  0.000000
```

Figure 109 Output for the average service length by age group

Analysis 5-18: Find the reason of termination for the 4 less than 20 years old employees with less than one year experience.

```
term_ageG_service %>%
  filter(agegroup == "<20") %>%
  select(termreason_desc, city_name, job_title, terminationdate_key)
```

Figure 110 Cide for finding the reason of termination for the 4 less than 20 years old employees with less than one year experience

```
> term_ageG_service %>%
+   filter(agegroup == "<20") %>%
+   select(termreason_desc, city_name, job_title, terminationdate_key)
Adding missing grouping variables: `EmployeeID`
# A tibble: 4 x 5
# Groups:   EmployeeID [4]
  EmployeeID termreason_desc city_name     job_title terminationdate_key
       <int> <fct>           <fct>         <fct>     <date>
1       8181 Resignaton      Prince George Cashier   2013-07-02
2       8223 Resignaton      Trail         Cashier   2013-10-17
3       8226 Resignaton      Victoria      Cashier   2013-09-14
4       8264 Resignaton      Vancouver     Cashier   2013-08-30
```

Figure 111 Output for the reason of termination for the 4 less than 20 years old employees with less than one year experience

Conclusion

Employees leaving the company with the longest service mostly falls in age group 60-69 and the average for those with shortest service is 22 years old. However, the age group 50-59 has the best average length of service and surprisingly there are 4 employees less than 20 years old which resigned at less than one year experience. These 4 employees worked as cashier in various city and resigned in 2013.

In terms of gender, female has the better length of service average compared to male. As for business unit, even though *Stores* unit has higher number of longest service employees, *Headoffice* unit still wins in terms of overall average. For jobs and departments, employees working as Customer Service has the greatest number of employees with lowest length of service as well as the overall average. Finally, for termination reason, it is won by resignation for both lowest top 10% and overall average.

Question 6: What are the characteristics which affect the length of service based on active employees?

Analysis 6-1: Find the average age for the top 10% longest and shortest service.

```
emp_table %>% group_by(EmployeeID) %>% slice(which.max(length_of_service))
hire_service <- service_df %>%
  filter(STATUS == "ACTIVE") %>%
  rename(service = length_of_service)

hire_service %>%
  ungroup %>%
  filter(service >= quantile(service, 0.9)) %>%
  pull(age) %>%
  summary

hire_service %>%
  ungroup %>%
  filter(service <= quantile(service, 0.1)) %>%
  pull(age) %>%
  summary
```

Figure 112 Code for finding the average age for the old and new employees

```
> hire_service %>%
+   ungroup %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   pull(age) %>%
+   summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  48.00   60.00   61.00   61.26   63.00   64.00
>
> hire_service %>%
+   ungroup %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   pull(age) %>%
+   summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  19.00   22.00   24.00   23.37   25.00   30.00
```

Figure 113 Output for the average age for the old and new employees

Analysis 6-2: Find the gender count for the top 10% longest and shortest service.

```
top_n_los(hire_service, 0.9, "gender_short")

last_n_los(hire_service, 0.1, "gender_short")
```

Figure 114 Code for finding the gender count for old and new employees

```
> top_n_los(hire_service, 0.9, "gender_short")
# A tibble: 2 x 2
  gender_short     n
  <fct>        <int>
1 F              403
2 M              323
>
> last_n_los(hire_service, 0.1, "gender_short")
# A tibble: 2 x 2
  gender_short     n
  <fct>        <int>
1 F              364
2 M              322
```

Figure 115 Output for the gender count for old and new employees

Analysis 6-3: Find the average length of service based on the gender.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ gender_short, ., mean) %>%
  arrange(desc(service))
```

Figure 116 Code for finding the average service length of active employees based on gender

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ gender_short, ., mean) %>%
+   arrange(desc(service))
  gender_short  service
1            F 13.27434
2            M 12.92645
```

Figure 117 Output for the average service length of active employees by gender

Analysis 6-4: Find the business unit count for the top 10% longest and shortest service.

```
top_n_los(hire_service, 0.9, "BUSINESS_UNIT")

last_n_los(hire_service, 0.1, "BUSINESS_UNIT")
```

Figure 118 Code for finding the business unit count for old and new employees

```
> top_n_los(hire_service, 0.9, "BUSINESS_UNIT")
# A tibble: 2 x 2
  BUSINESS_UNIT     n
  <fct>         <int>
1 STORES          695
2 HEADOFFICE       31
>
> last_n_los(hire_service, 0.1, "BUSINESS_UNIT")
# A tibble: 1 x 2
  BUSINESS_UNIT     n
  <fct>         <int>
1 STORES          686
```

Figure 119 Output for the business unit count for old and new employees

Analysis 6-5: Find the average length of service based on the business unit.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ BUSINESS_UNIT, ., mean) %>%
  arrange(desc(service))
```

Figure 120 Code for finding the active employees' average service length by business unit

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ BUSINESS_UNIT, ., mean) %>%
+   arrange(desc(service))
  BUSINESS_UNIT  service
1    HEADOFFICE 22.33962
2        STORES 13.01060
```

Figure 121 Output for the active employees' average service length by business unit

Analysis 6-6: Find the city count for the top 1% longest and shortest service.

```
top_n_los(hire_service, 0.99, "city_name")

last_n_los(hire_service, 0.01, "city_name")
```

Figure 122 Code for finding the city count for old and new employees

```
> top_n_los(hire_service, 0.99, "city_name")
# A tibble: 18 x 2
   city_name           n
   <fct>           <int>
 1 Vancouver          24
 2 Nanaimo             5
 3 Kamloops            4
 4 Cranbrook           3
 5 Kelowna             3
 6 Richmond            3
 7 Terrace             3
 8 Burnaby             2
 9 New Westminster     2
10 Port Coquitlam      2
11 Prince George       2
12 West Vancouver      2
13 Abbotsford          1
14 Aldergrove          1
15 Langley             1
16 Quesnel             1
17 Squamish            1
18 Vernon              1
```

Figure 123 Output for the city count based on old employees

```
> last_n_los(hire_service, 0.01, "city_name")
# A tibble: 22 x 2
   city_name           n
   <fct>           <int>
 1 Vancouver          10
 2 Prince George       7
 3 Victoria            6
 4 Nanaimo             5
 5 Terrace             5
 6 Chilliwack          4
 7 Kamloops            4
 8 Kelowna             4
 9 Quesnel             4
10 New Westminster     3
# ... with 12 more rows
```

Figure 124 Output for the city count based on new employees

Analysis 6-7: Find the best and worst average length of service based on the city.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ city_name, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

hire_service %>%
  ungroup %>%
  aggregate(service ~ city_name, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 125 Code for finding the best and worst average service length by city for active employees

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ city_name, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
       city_name  service
1 Port Coquitlam 14.61017
2         Vernon 13.95833
3         Surrey 13.84302
4        Kelowna 13.79554
>
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ city_name, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
         city_name  service
1       White Rock 11.33333
2      Bella Bella 10.85714
3 New Westminister  9.80000
4      Fort Nelson  8.00000
```

Figure 126 Output for the best and worst average service length by city for active employees

Analysis 6-8: Find the job title count for the top 1% longest and shortest service.

```
top_n_los(hire_service, 0.99, "job_title")

last_n_los(hire_service, 0.01, "job_title")
```

Figure 127 Code for finding the job title count for old and new employees

```
> top_n_los(hire_service, 0.99, "job_title")
# A tibble: 19 x 2
   job_title                          n
   <fct>                          <int>
 1 Meat Cutter                       18
 2 Meats Manager                      7
 3 Produce Manager                    6
 4 Customer Service Manager           4
 5 Dairy Person                       4
 6 Processed Foods Manager            4
 7 Store Manager                      4
 8 Bakery Manager                     3
 9 CEO                                1
10 CHief Information Officer          1
11 Director, Recruitment              1
12 Exec Assistant, Finance            1
13 Exec Assistant, Human Resources    1
14 Exec Assistant, Legal Counsel      1
15 Exec Assistant, VP Stores          1
16 Legal Counsel                      1
17 VP Finance                         1
18 VP Human Resources                 1
19 VP Stores                          1
>
> last_n_los(hire_service, 0.01, "job_title")
# A tibble: 3 x 2
  job_title         n
  <fct>         <int>
1 Cashier          60
2 Shelf Stocker     7
3 Dairy Person      2
```

Figure 128 Output for the job title count for old employees

Analysis 6-9: Find the best and worst average length of service based on the job title.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ job_title, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

hire_service %>%
  ungroup %>%
  aggregate(service ~ job_title, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 129 Code for finding the best and worst average service length based on active employees' job title

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ job_title, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
                        job_title service
1                             CEO      26
2       CHief Information Officer      26
3           Director, Recruitment      26
4         Exec Assistant, Finance      26
5  Exec Assistant, Human Resources      26
6    Exec Assistant, Legal Counsel      26
7        Exec Assistant, VP Stores      26
8                   Legal Counsel      26
9                      VP Finance      26
10             VP Human Resources      26
11                      VP Stores      26
>
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ job_title, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
      job_title   service
1 Produce Clerk 16.218559
2         Baker 14.149626
3  Dairy Person 10.852679
4 Shelf Stocker  9.006098
5       Cashier  6.643002
```

Figure 130 Output for the best and worst average service length by active employees' job title

Analysis 6-10: Find the department count for the top 1% longest and shortest service.

```
top_n_los(hire_service, 0.99, "department_name")

last_n_los(hire_service, 0.01, "department_name")
```

Figure 131 Code for finding the department count for old and new employees

```
> top_n_los(hire_service, 0.99, "department_name")
# A tibble: 9 x 2
  department_name      n
  <fct>            <int>
1 Meats               25
2 Executive           10
3 Produce              6
4 Customer Service     4
5 Dairy                4
6 Processed Foods      4
7 Store Management     4
8 Bakery               3
9 Recruitment          1
>
> last_n_los(hire_service, 0.01, "department_name")
# A tibble: 3 x 2
  department_name      n
  <fct>            <int>
1 Customer Service    60
2 Processed Foods      7
3 Dairy                2
```

Figure 132 Output for the department count for old and new employees

Analysis 6-11: Find the best and worst average length of service based on the department.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ department_name, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))


hire_service %>%
  ungroup %>%
  aggregate(service ~ department_name, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 133 Code for finding the best and worst average service length based on active employees' department

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ department_name, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
   department_name service
1        Executive      26
2       Accounting      24
3 Employee Records      24
4    HR Technology      24
>
>
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ department_name, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
   department_name   service
1            Dairy 10.852679
2  Processed Foods  9.447563
3 Customer Service  7.016865
```

Figure 134 Output for the best and worst average service length based on active employees' department

Analysis 6-12: Find the store name count for the top 1% longest and shortest service.

```
top_n_los(hire_service, 0.99, "store_name")

last_n_los(hire_service, 0.01, "store_name")
```

Figure 135 Code for finding the store name count for old and new employees

```
> top_n_los(hire_service, 0.99, "store_name")
# A tibble: 20 x 2
   store_name     n
        <int> <int>
 1         35    11
 2         44     7
 3         43     6
 4         18     5
 5         15     4
 6          8     3
 7         16     3
 8         29     3
 9         32     3
10          5     2
11         21     2
12         25     2
13         26     2
14         38     2
15          1     1
16          2     1
17         17     1
18         28     1
19         30     1
20         36     1
```

Figure 136 Output for the store name count for old employees

```
> last_n_los(hire_service, 0.01, "store_name")
# A tibble: 23 x 2
   store_name     n
        <int> <int>
 1         26     7
 2         44     6
 3         46     6
 4         18     5
 5         32     5
 6          6     4
 7         15     4
 8         16     4
 9         28     4
10         43     4
# ... with 13 more rows
```

Figure 137 Output for the store name count for new employees

Analysis 6-13: Find the best and worst average length of service based on the store name.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ store_name, ., mean) %>%
  filter(service >= quantile(service, 0.9)) %>%
  arrange(desc(service))

hire_service %>%
  ungroup %>%
  aggregate(service ~ store_name, ., mean) %>%
  filter(service <= quantile(service, 0.1)) %>%
  arrange(desc(service))
```

Figure 138 Code for finding the best and worst average service length by active employees' store name

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ store_name, ., mean) %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   arrange(desc(service))
  store_name  service
1         41 22.59064
2         42 16.66667
3         35 16.65217
4         25 14.61017
>
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ store_name, ., mean) %>%
+   filter(service <= quantile(service, 0.1)) %>%
+   arrange(desc(service))
  store_name  service
1         43 9.874576
2         20 9.800000
3         11 8.000000
4         44 4.862454
```

Figure 139 Output for the best and worst average service length by active employees' store name

Analysis 6-14: Find the average length of service based on active employee.

```
hire_service %>%
  ungroup %>%
  aggregate(service ~ termreason_desc, ., mean) %>%
  arrange(desc(service))
```

Figure 140 Code for finding the average length of service based on active employees

```
> hire_service %>%
+   ungroup %>%
+   aggregate(service ~ termreason_desc, ., mean) %>%
+   arrange(desc(service))
  termreason_desc  service
1  Not Applicable 13.10326
```

Figure 141 Output for the average length of service based on active employees

Analysis 6-15: Find the age group count for the top 10% longest and shortest service.

```
hire_ageG_service <-  hire_service %>%
  mutate(agegroup = case_when(
    age < 20 ~ '<20',
    age < 30 ~ '20-29',
    age < 40 ~ '30-39',
    age < 50 ~ '40-49',
    age < 60 ~ '50-59',
    age < 70 ~ '60-69',
    age < 80 ~ '70-79',
    age < 90 ~ '80-89',
))

hire_ageG_service %>% top_n_los(0.9, "agegroup")
hire_ageG_service %>% last_n_los(0.1, "agegroup")
```

Figure 142 Code for finding the age group count for old and new employees

```
> hire_ageG_service %>% top_n_los(0.9, "agegroup")
# A tibble: 3 x 2
  agegroup     n
  <chr>    <int>
1 60-69      561
2 50-59      163
3 40-49        2
> hire_ageG_service %>% last_n_los(0.1, "agegroup")
# A tibble: 3 x 2
  agegroup     n
  <chr>    <int>
1 20-29      684
2 <20          1
3 30-39        1
```

Figure 143 Output for the age group count for old and new employees

Analysis 6-16: Find the average length of service based on the age group.

```
hire_ageG_service %>%
  aggregate(service ~ agegroup, ., mean) %>%
  arrange(desc(service))
```

Figure 144 Code for finding the average service length by active employees' age group

```
> hire_ageG_service %>%
+   aggregate(service ~ agegroup, ., mean) %>%
+   arrange(desc(service))
  agegroup   service
1    60-69 20.659298
2    50-59 18.964588
3    40-49 14.191042
4    30-39  8.984274
5    20-29  3.809524
6      <20  0.000000
```

Figure 145 Output for the average service length by active employees' age group

Conclusion

The current active employee with the best average length of service falls at age 61 and the least average length are dominated by age 23. Furthermore, employees with gender female seems to have longer length of service compared to man. This applies for the shortest service as well.

In terms of business unit, even though the current active employee with longer service has higher number for *Stores* unit, *Headoffice* unit still has the overall best average. As for jobs and departments, the longest service is still held by the CEO and the shortest service is held by Cashier from Customer Service department. Lastly, the overall average for the service is 13 years.

Extra Features

theme_update()

This function comes from *ggplot2* package. It is used to apply the specify theme to every other plot that is drawn (tidyverse, n.d.-f). Thus, our code will be less redundant.

```
theme_update(plot.title = element_text(hjust=0.5),
             axis.title.x = element_text(face="bold", color="black", size=10),
             axis.title.y = element_text(face="bold", color="black", size=10),
             legend.title = element_text(face="bold", size=12),
             axis.text.x = element_text(angle=90, size=8))
```

Figure 146 Usage of theme_update()

Without this function, we will need to specify it for each plot, for example:

```
bar_plot <- ggplot(num_emp, aes(x=year, y=total)) +
  geom_bar(stat="Identity", fill="light blue") +
  geom_text(aes(label=total), size=3) +
  scale_x_continuous(
      breaks = seq(min(num_emp$year), max(num_emp$year))
  ) +
  labs(x = "\nYear\n", y = "\nAmount\n", title = "\nTotal Employees\n") +
  theme(
    plot.title = element_text(hjust=0.5),
    axis.title.x = element_text(face="bold", color="black", size=10),
    axis.title.y = element_text(face="bold", color="black", size=10),
    legend.title = element_text(face="bold", size=12),
    axis.text.x = element_text(angle=90, size=8)
  )
```

Figure 147 Plotting without theme_update()

By using this function, our code can be simplified to:

```
bar_plot <- ggplot(num_emp, aes(x=year, y=total)) +
    geom_bar(stat="Identity", fill="light blue") +
    geom_text(aes(label=total), size=3) +
    scale_x_continuous(
        breaks = seq(min(num_emp$year), max(num_emp$year))
    ) +
    labs(x = "\nYear\n", y = "\nAmount\n", title = "\nTotal Employees\n")
```

Figure 148 Plotting with theme_update()

time_length()

This function comes from *lubridate* package. It is used to convert a time interval into another time unit (RDocumentation, n.d.-k).

```
> now()
[1] "2021-11-10 11:15:56 +07"
> difftime(now(), as.Date("2019-10-10"))
Time difference of 762.1779 days
> difftime(now(), as.Date("2019-10-10")) %>% lubridate::time_length(unit = "year")
[1] 2.08673
> difftime(now(), as.Date("2019-10-10")) %>% lubridate::time_length(unit = "year") %>%
 floor
[1] 2
```

Figure 149 Usage of time_length()

n_distinct()

This function comes from *dplyr* package. It counts the number of unique occurrences of the specified columns (tidyverse, n.d.-d).

```
> emp_table %>% summarise(n = n_distinct(EmployeeID))
# A tibble: 1 x 1
      n
  <int>
1  6284
> nrow(emp_table)
[1] 49653
```

Figure 150 Usage of n_distinct()

merge()

This function comes from *base* package. It is used to merged two data frames by the specified columns. To include all the values from the two data frames, we can specify the parameter "*all = TRUE*" (RDocumentation, n.d.-d).

```
by_year_gender <- merge(hire_by_year_gender, term_by_year_gender,
                        by = c("year", "gender"), all = TRUE)
colnames(by_year_gender) <- c("year", "gender", "hired", "termed")
by_year_gender$hired <- ifelse(is.na(by_year_gender$hired), 0,
                               by_year_gender$hired)
by_year_gender$termed <- ifelse(is.na(by_year_gender$termed), 0,
                               by_year_gender$termed)
```

```
> head(hire_by_year_gender, 3); head(term_by_year_gender, 3)
# A tibble: 3 x 3
# Groups:   year [2]
   year gender     n
  <dbl> <fct>  <int>
1  1989 F         40
2  1989 M         46
3  1990 F        118
# A tibble: 3 x 3
# Groups:   year [2]
   year gender     n
  <dbl> <fct>  <int>
1  2006 F         60
2  2006 M         74
3  2007 F         81
> head(by_year_gender, 3)
  year gender hired termed
1 1989      F    40      0
2 1989      M    46      0
3 1990      F   118      0
```

pivot_longer()

This function comes from *tidyr* package. It is used to convert data from wide format to long format. Data in wide format has the values of the variable act as the column headers or in other words there are multiple observations in a single row. On the other hand, data in long format has the variable values stacked into one column. Thus, each row represents a single observation (Wickham, n.d.).

In the example below, *by_year_gender* data frame already has the gender in long format which is good for our visualization. However, it still has two observations in a single row, i.e., data about hired and terminated amount.

```
> head(by_year_gender)
  year gender hired termed
1 1989      F    40      0
2 1989      M    46      0
3 1990      F   118      0
4 1990      M    96      0
5 1991      F   105      0
6 1991      M    95      0
```

Figure 151 Data in wide format

```r
by_year_gender_long <- by_year_gender %>%
    pivot_longer(cols = c("hired", "termed"),
                 names_to = "action", values_to = "amount")
```

Figure 152 Usage of pivot_longer() to convert data from wide to long format

```
> head(by_year_gender_long, 6)
# A tibble: 6 x 4
   year gender action amount
  <dbl> <fct>  <chr>   <dbl>
1  1989 F      hired      40
2  1989 F      termed      0
3  1989 M      hired      46
4  1989 M      termed      0
5  1990 F      hired     118
6  1990 F      termed      0
```

Figure 153 Data in long format

scale_x_continuous() and scale_y_continuous()

Both functions come from *ggplot2* package. They are used to adjust the scaling of *x*-axis and *y*-axis of a certain graph (RDocumentation, n.d.-f).

```r
line_plot <- num_emp %>%
    pivot_longer(
        cols = c("hired", "termed"),
        names_to = "action",
        values_to = "amount"
        ) %>%
    ggplot(aes(x=year, y=amount)) +
    geom_line(aes(color=action), size = 2) +
    labs(title = "\nHired vs Termed\n")
```

Figure 154 Without scaling function

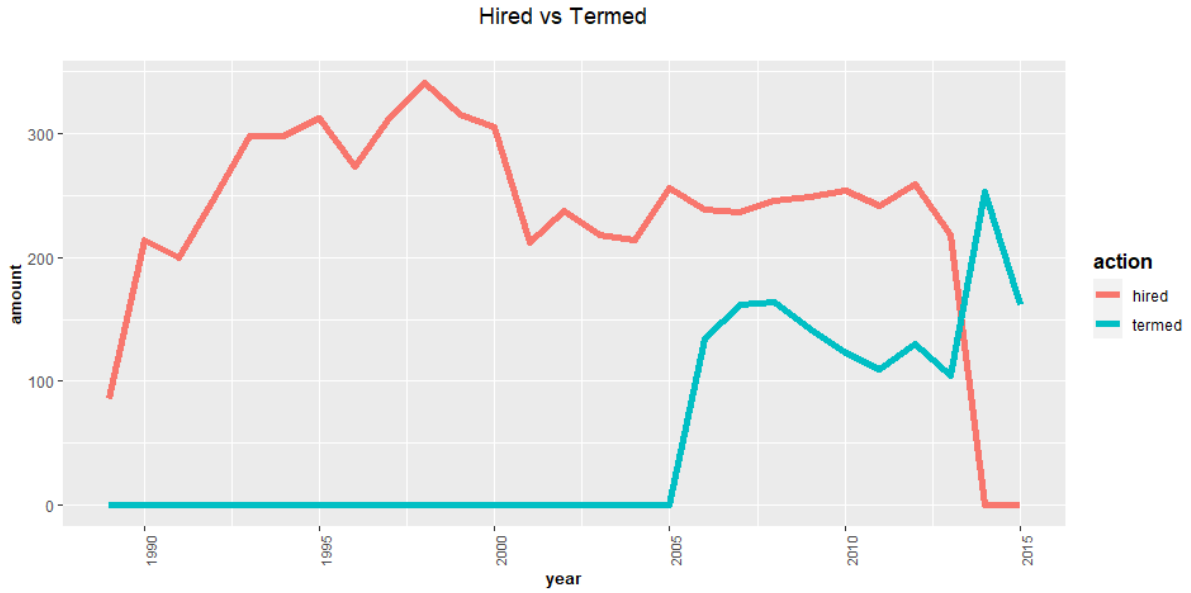


Figure 155 Graph without custom scaling

```
line_plot <- num_emp %>%
    pivot_longer(
        cols = c("hired", "termed"),
        names_to = "action",
        values_to = "amount"
        ) %>%
    ggplot(aes(x=year, y=amount)) +
    geom_line(aes(color=action), size = 2) +
    scale_x_continuous(
        name = "year",
        breaks = seq(min(num_emp$year), max(num_emp$year))
    ) +
    scale_y_continuous(
        name="amount",
        breaks = seq(
            0,
            round_any(
                max(c(num_emp[,"hired"], num_emp[,"termed"])), 25, ceiling
                ),
            25
        )
    ) +
    labs(title = "\nHired vs Termed\n")
```

Figure 156 With custom scaling

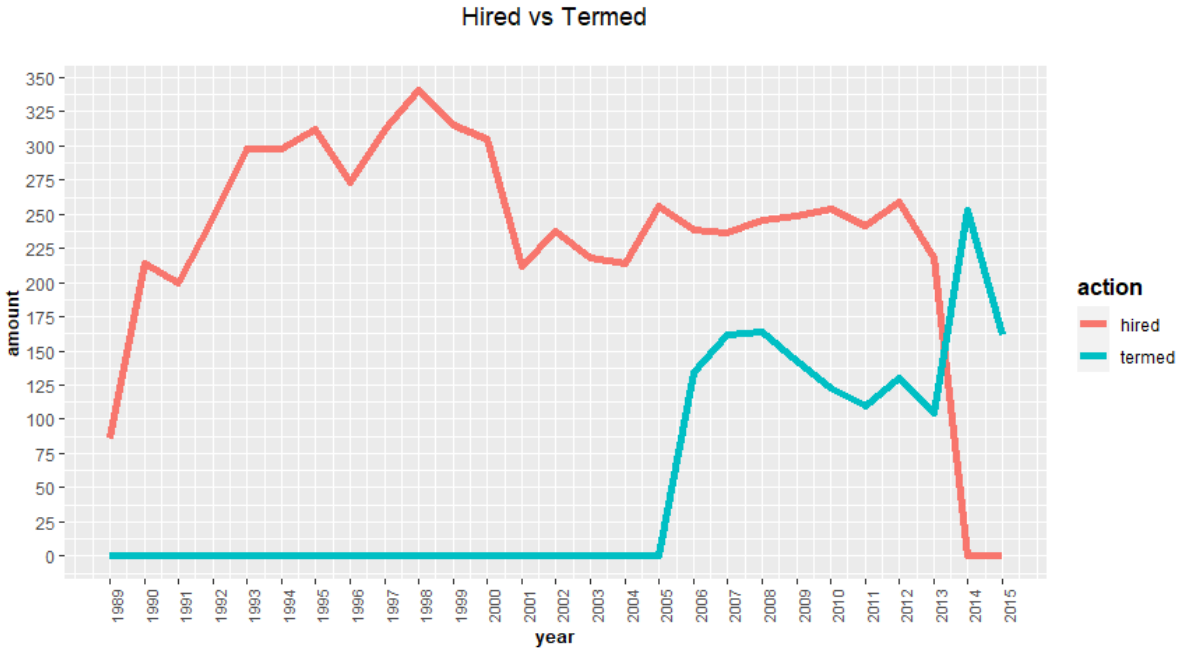


Figure 157 Graph with custom scaling

round_any

This function comes from *plyr* package. It is used to round a value to multiple of any number (RDocumentation, n.d.-h). Furthermore, we can specify additional parameter to use *floor* and *ceiling* function.

```
> mx <- max(c(num_emp$hired, num_emp$termed))
> mx
[1] 341
> round_any(mx, 25, ceiling)
[1] 350
> round_any(mx, 25, floor)
[1] 325
```

Figure 158 Usage of round_any() function

labs()

This function comes from *ggplot2* package. It is used to rename any label on our plot (RDocumentation, n.d.-c).

```r
bar_plot <- ggplot(num_emp, aes(x=year, y=total)) +
    geom_bar(stat="Identity", fill="light blue") +
    geom_text(aes(label=total), size=3) +
    scale_x_continuous(
        breaks = seq(min(num_emp$year), max(num_emp$year))
    ) +
    labs(x = "\nYear\n", y = "\nAmount\n", title = "\nTotal Employees\n")
```

Figure 159 Usage of labs() function

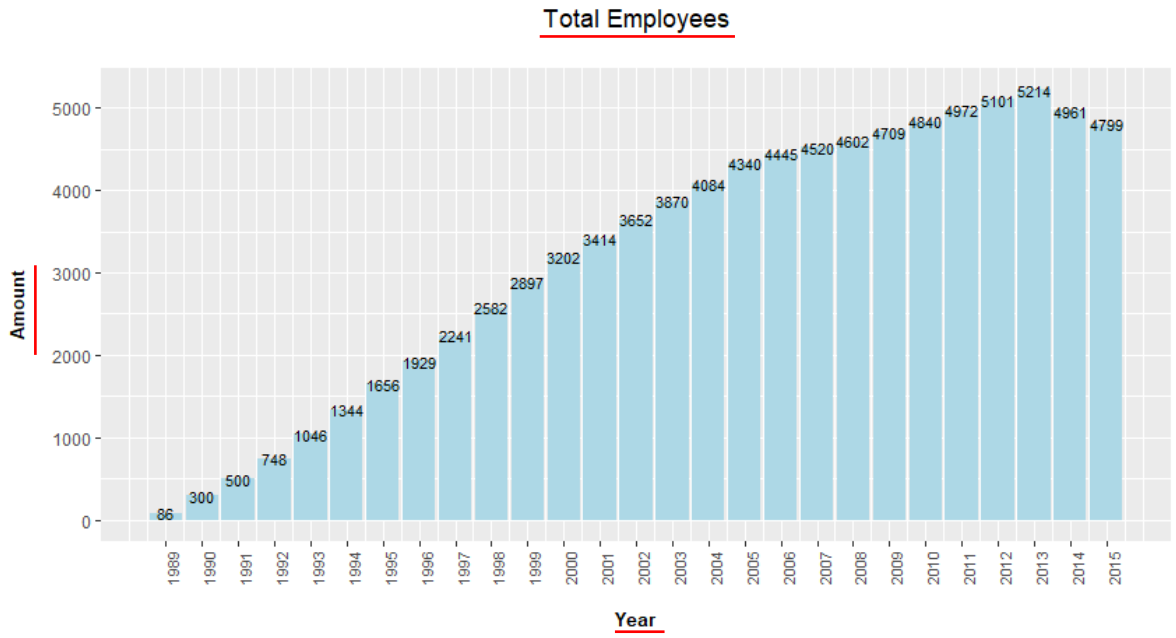


Figure 160 Graph with custom labelling

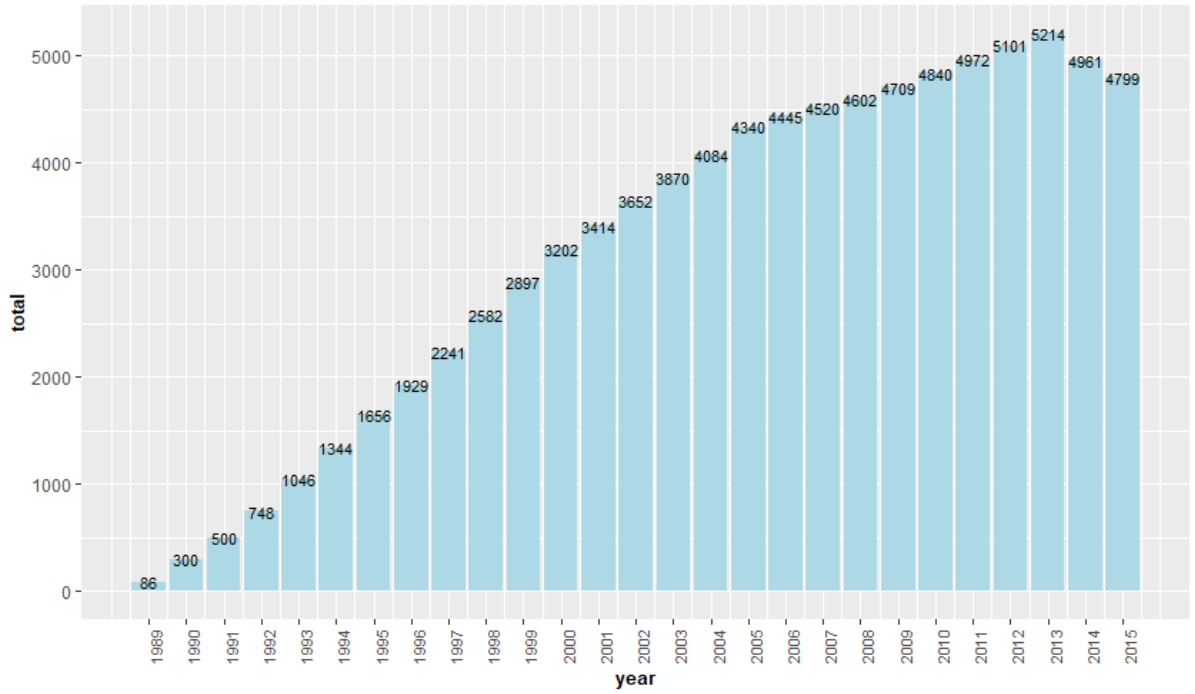


Figure 161 Graph without custom labelling

group_by_at()

This function comes from *dplyr* package. It can be used with variables as the parameters (tidyverse, n.d.-g).

```
hire_by_var <- function(emp_table, colVec) {
    return(
        emp_table %>%
        mutate(year = year(orighiredate_key)) %>%
        group_by_at(colVec)
    )
}
```

Figure 162 Usage of group_by_at() function

```
> head(hire_by_var(emp_table, c("age", "STATUS")))
# A tibble: 6 x 19
# Groups:   age, STATUS [6]
  EmployeeID recorddate_key birthdate_key origihiredate_key terminationdate_key   age
       <int> <date>         <date>        <date>            <date>              <int>
1       1318 2006-12-31     1954-01-03    1989-08-28        1900-01-01             52
2       1318 2007-12-31     1954-01-03    1989-08-28        1900-01-01             53
3       1318 2008-12-31     1954-01-03    1989-08-28        1900-01-01             54
4       1318 2009-12-31     1954-01-03    1989-08-28        1900-01-01             55
5       1318 2010-12-31     1954-01-03    1989-08-28        1900-01-01             56
6       1318 2011-12-31     1954-01-03    1989-08-28        1900-01-01             57
# ... with 13 more variables: length_of_service <int>, city_name <fct>,
#   department_name <fct>, job_title <fct>, store_name <int>, gender_short <fct>,
#   gender_full <fct>, termreason_desc <fct>, termtype_desc <fct>,
#   STATUS_YEAR <date>, STATUS <fct>, BUSINESS_UNIT <fct>, year <dbl>
```

Figure 163 Output of grouped data frame using group_by_at() function

tapply()

This function comes from *base* package. It is used to apply a certain function to a specific column grouped by other columns (RDocumentation, n.d.-j).

In the example below, we are applying the function *sum* on *amount* column, but group it by *year*.

```
total_amt_per_year <- tapply(
    long_format_temp$amount,
    long_format_temp$year,
    FUN = sum
)
```

Figure 164 Usage of tapply() function

```
> head(long_format_temp)
# A tibble: 6 x 4
   year BUSINESS_UNIT action amount
  <dbl> <fct>         <chr>   <dbl>
1  1989 HEADOFFICE    hired      41
2  1989 HEADOFFICE    termed      0
3  1989 STORES        hired      45
4  1989 STORES        termed      0
5  1990 HEADOFFICE    hired      39
6  1990 HEADOFFICE    termed      0
> total_amt_per_year
1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
  86  214  200  248  298  298  312  273  312  341  315  305  212  238  218  214  256
2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
 373  399  410  391  377  352  389  323  253  162
```

Figure 165 Output of tapply()

aggregate()

This function comes from *stats* package. It behaves similarly to *tapply()* function, but it can be used together with *pipe* operator (RDocumentation, n.d.-a).

```
long_format_temp %>% aggregate(amount ~ year, ., sum)
```

Figure 166 Usage of aggregate() function

```
> long_format_temp %>% aggregate(amount ~ year, ., sum) %>% head
  year amount
1 1989     86
2 1990    214
3 1991    200
4 1992    248
5 1993    298
6 1994    298
```

Figure 167 Output using aggregate() function

str_replace()

This function comes from *stringr* package. It is used to do replacing in a string (RDocumentation, n.d.-i). It receives a regex to search for the pattern to replace.

```
> long_format_temp %>%
+     mutate(
+         BU_action = factor(
+             str_replace(interaction(BUSINESS_UNIT, action), "\\.", "\n")
+         )
+     ) %>% head
# A tibble: 6 x 5
   year BUSINESS_UNIT action amount BU_action
  <dbl> <fct>         <chr>   <dbl> <fct>
1  1989 HEADOFFICE    hired      41 "HEADOFFICE\nhired"
2  1989 HEADOFFICE    termed      0 "HEADOFFICE\ntermed"
3  1989 STORES        hired      45 "STORES\nhired"
4  1989 STORES        termed      0 "STORES\ntermed"
5  1990 HEADOFFICE    hired      39 "HEADOFFICE\nhired"
6  1990 HEADOFFICE    termed      0 "HEADOFFICE\ntermed"
```

Figure 168 Usage of str_replace() function

interaction()

This function comes from *base* package. It is used to compute a factor which represents the interaction of the given factors (RDocumentation, n.d.-b).

```
> long_format_temp %>% mutate(BU_action = factor(interaction(BUSINESS_UNIT, action))) %
>% head
# A tibble: 6 x 5
   year BUSINESS_UNIT action amount BU_action
  <dbl> <fct>         <chr>   <dbl> <fct>
1  1989 HEADOFFICE    hired      41 HEADOFFICE.hired
2  1989 HEADOFFICE    termed      0 HEADOFFICE.termed
3  1989 STORES        hired      45 STORES.hired
4  1989 STORES        termed      0 STORES.termed
5  1990 HEADOFFICE    hired      39 HEADOFFICE.hired
6  1990 HEADOFFICE    termed      0 HEADOFFICE.termed
```

Figure 169 Usage of interaction() function

scale_fill_discrete()

This function comes from *ggplot2* package. It is used to customize the legends for *fill* aesthetic (tidyverse, n.d.-c).

```
analysis_1_3_a <- hire_by_year_gender %>%
    ggplot(aes(x=year, y=n, fill=gender)) +
    geom_bar(stat = "Identity", position = "dodge", width = 0.7) +
    scale_x_continuous(breaks = seq(min(hire_by_year_gender$year),
                                    max(hire_by_year_gender$year))) +
    scale_y_continuous(
        breaks = seq(0, round_any(max(hire_by_year_gender$n), 25, ceiling), 25)
    ) +
    scale_fill_discrete(name = "Gender") +
    labs(x="Year", title = "\nNumbers of hired employees by gender yearly")
```

Figure 170 Usage of scale_fill_discrete() function

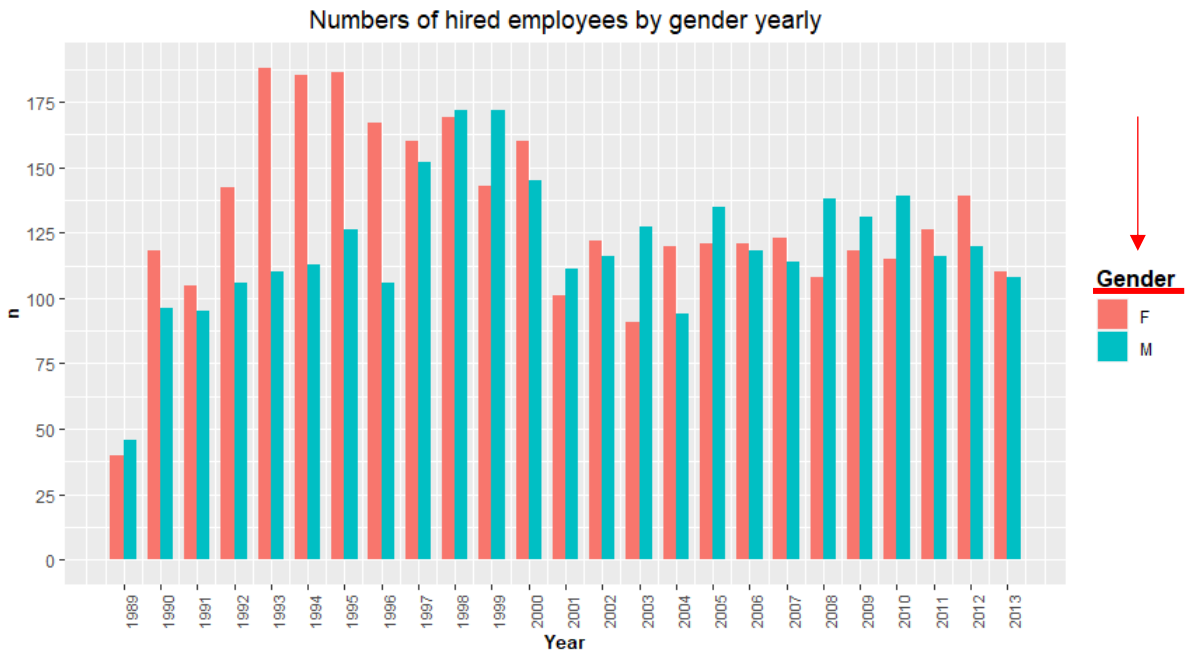
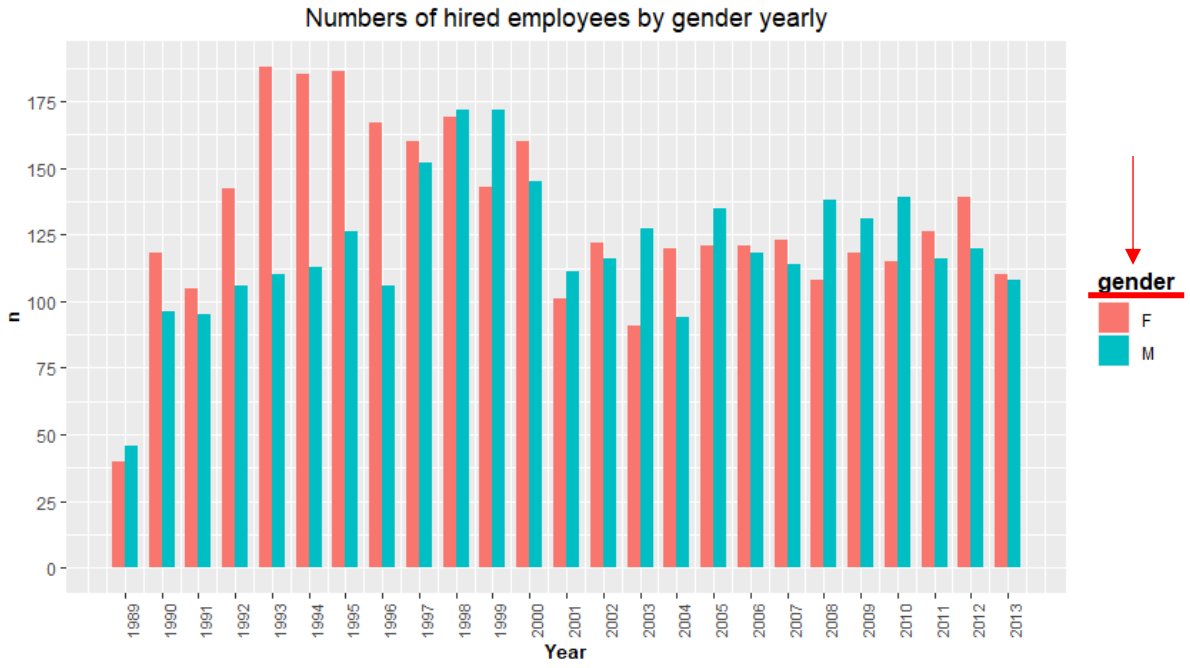


Figure 171 Graph with scale_fill_discrete()



theme() and element_text()

Both functions comes from *ggplot2* package and often used together. Using this function, we can customize the look of our graph (tidyverse, n.d.-p) (tidyverse, n.d.-i).

```
term_test <- term_by_var(
  emp_table,
  c("year", "termreason_desc", "BUSINESS_UNIT", "gender_short")) %>%
  filter(termreason_desc != "Not Applicable") %>%
  ungroup %>%
  mutate(BU_gender = factor(paste(gender_short, BUSINESS_UNIT))) %>%
  group_by(year, termreason_desc, BU_gender) %>%
  summarise(mean_age = mean(age)) %>%
  ggplot(aes(x=termreason_desc, y=mean_age, fill=BU_gender)) +
  geom_bar(stat="Identity", position = "dodge", color="black") +
  scale_fill_discrete(name = "Business Unit\nby gender") +
  labs(title = "Terminated employees yearly age average\nbased on Termination
Reason by Business Unit and Gender",
       x = "Termination Reason", y = "Mean Age") +
  theme(axis.text.x = element_text(angle=45, vjust=0.5)) +
  facet_wrap(~year)
```

Figure 172 Usage of theme() and element_text() functions

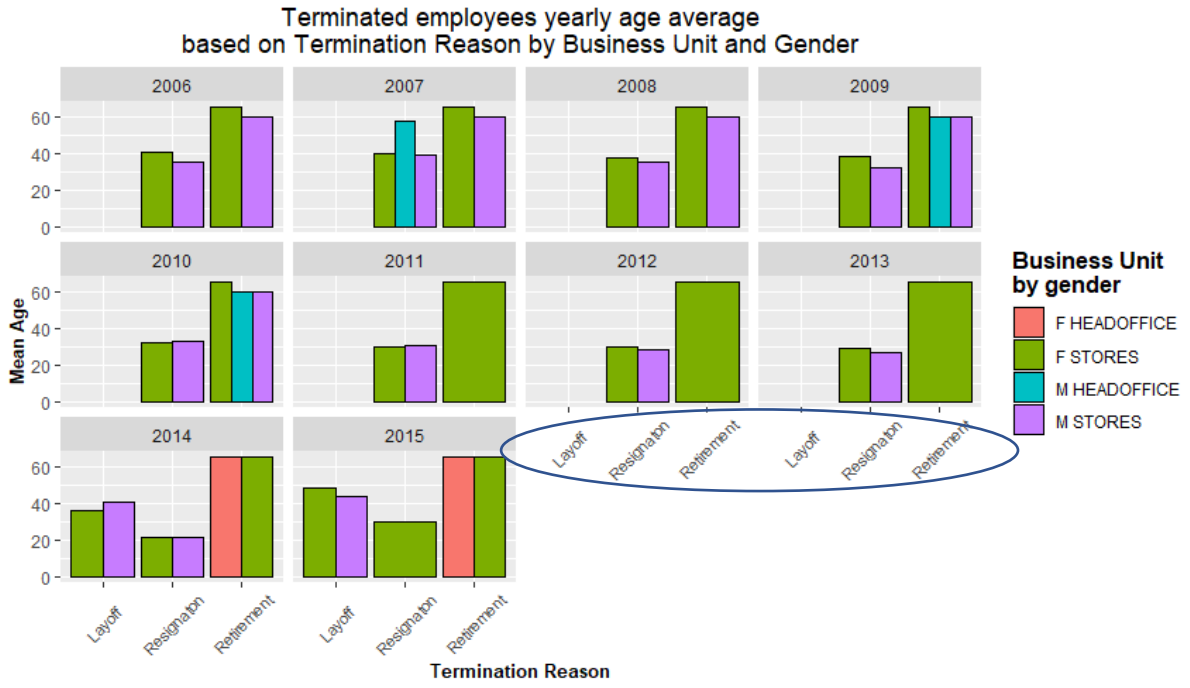


Figure 173 Customized graph using theme() and element_text()

ungroup()

This function comes from *dplyr* package. It is used to undo *group_by()* function (tidyverse, n.d.-h).

```
> emp_table %>% group_by(age) %>% head(2)
# A tibble: 2 x 18
# Groups:   age [2]
  EmployeeID recorddate_key birthdate_key origihiredate_key terminationdate_key   age
       <int> <date>         <date>        <date>            <date>               <int>
1       1318 2006-12-31     1954-01-03    1989-08-28        1900-01-01              52
2       1318 2007-12-31     1954-01-03    1989-08-28        1900-01-01              53
# ... with 12 more variables: length_of_service <int>, city_name <fct>,
#   department_name <fct>, job_title <fct>, store_name <int>, gender_short <fct>,
#   gender_full <fct>, termreason_desc <fct>, termtype_desc <fct>,
#   STATUS_YEAR <date>, STATUS <fct>, BUSINESS_UNIT <fct>
> emp_table %>% group_by(age) %>% head(2) %>% ungroup
# A tibble: 2 x 18
  EmployeeID recorddate_key birthdate_key origihiredate_key terminationdate_key   age
       <int> <date>         <date>        <date>            <date>               <int>
1       1318 2006-12-31     1954-01-03    1989-08-28        1900-01-01              52
2       1318 2007-12-31     1954-01-03    1989-08-28        1900-01-01              53
# ... with 12 more variables: length_of_service <int>, city_name <fct>,
#   department_name <fct>, job_title <fct>, store_name <int>, gender_short <fct>,
#   gender_full <fct>, termreason_desc <fct>, termtype_desc <fct>,
#   STATUS_YEAR <date>, STATUS <fct>, BUSINESS_UNIT <fct>
```

Figure 174 Usage of ungroup() function

coord_polar()

This function comes from *ggplot2* package. It is used to transform bar chart into a pie chart (tidyverse, n.d.-j).

```
overall_gender <- by_year_gender_long %>%
    group_by(gender, action) %>%
    summarise(total = sum(amount)) %>%
    ungroup() %>%
    mutate(gender_action = factor(paste(gender, action))) %>%
    select(gender_action, total) %>%
    ggplot(aes(x="", y=total, fill=gender_action)) +
    geom_bar(stat = "Identity", color="white", width=0.1) +
    geom_text(aes(label=total), position = position_stack(vjust=0.5)) +
    coord_polar("y") +
    labs(title = "Gender hired and termed overall",
         x = "", y = "") +
    scale_fill_discrete(name = "Gender and action") +
    theme_void()
```

Figure 175 Usage of coord_polar() function

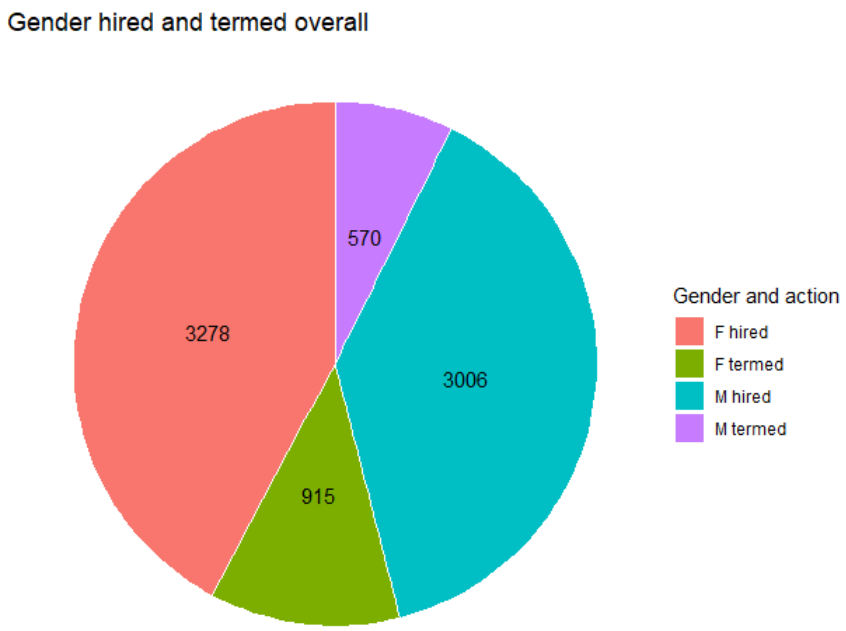


Figure 176 Graph using coord_polar() function

theme_void()

This function comes from *ggplot2* package. It is used to apply the plainest theme for the graph (tidyverse, n.d.-b). See Figure 176.

scale_size()

This function comes from *ggplot2* package. It is used to scale size based on define column (tidyverse, n.d.-l).

```
dept_merged %>%
  pivot_longer(cols = c("hired", "termed"),
               names_to = "action",
               values_to = "number") %>%
  group_by(year, action) %>%
  ggplot(aes(x=year, y=number)) +
  geom_point(aes(colour=action, shape=action, size=number)) +
  scale_size(range = c(1, 3)) +
  labs(title = "Number of hired and termed based on department") +
  facet_wrap(~dept)
```

Figure 177 Usage of scale_size() function

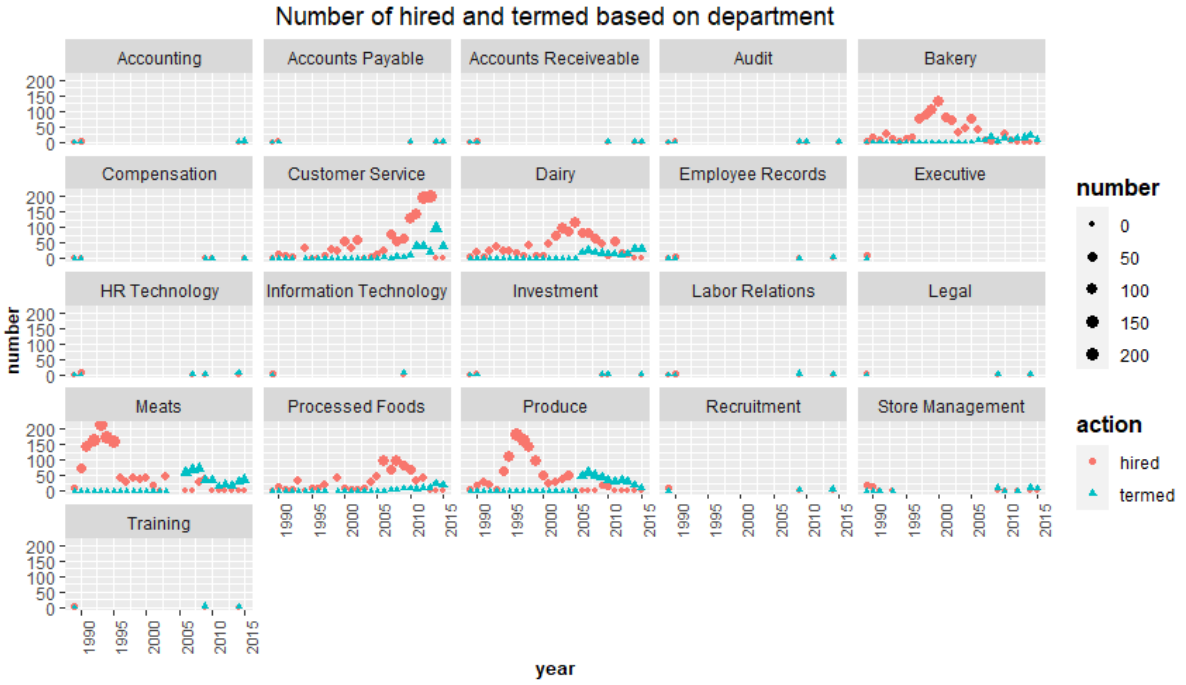


Figure 178 Scatterplot using scale_size() function

unique()

This function comes from *base* package. It is used to filter out only unique elements (RDocumentation, n.d.-l).

```r
hired_by_age <- emp_table %>%
  mutate(year = year(orighiredate_key),
         hired_age = year_diff(orighiredate_key, birthdate_key)) %>%
  group_by(year, BUSINESS_UNIT, EmployeeID) %>%
  summarise(age = hired_age) %>%
  unique
```

Figure 179 Usage of unique() function

```
>  emp_table %>%
+   mutate(year = year(orighiredate_key),
+          hired_age = year_diff(orighiredate_key, birthdate_key)) %>%
+   group_by(year, BUSINESS_UNIT, EmployeeID) %>%
+   summarise(age = hired_age) %>% head
`summarise()` has grouped output by 'year', 'BUSINESS_UNIT', 'EmployeeID'. You can over
ride using the `.groups` argument.
# A tibble: 6 x 4
# Groups:   year, BUSINESS_UNIT, EmployeeID [1]
   year BUSINESS_UNIT EmployeeID   age
  <dbl> <fct>              <int> <dbl>
1  1989 HEADOFFICE          1318    35
2  1989 HEADOFFICE          1318    35
3  1989 HEADOFFICE          1318    35
4  1989 HEADOFFICE          1318    35
5  1989 HEADOFFICE          1318    35
6  1989 HEADOFFICE          1318    35
```

Figure 180 Without unique()

```
> hired_by_age
# A tibble: 6,284 x 4
# Groups:   year, BUSINESS_UNIT, EmployeeID [6,284]
    year BUSINESS_UNIT EmployeeID   age
   <dbl> <fct>              <int> <dbl>
 1  1989 HEADOFFICE          1318    35
 2  1989 HEADOFFICE          1319    32
 3  1989 HEADOFFICE          1320    34
 4  1989 HEADOFFICE          1321    30
 5  1989 HEADOFFICE          1322    31
 6  1989 HEADOFFICE          1323    27
 7  1989 HEADOFFICE          1325    25
 8  1989 HEADOFFICE          1328    33
 9  1989 HEADOFFICE          1332    34
10  1989 HEADOFFICE          1334    28
# ... with 6,274 more rows
```

Figure 181 With unique()

scale_color_discrete()

Both functions come from *ggplot2* package. Both are used to customize the respective aesthetics (tidyverse, n.d.-c).

```r
age_dist <- htba %>%
  group_by(action, BUSINESS_UNIT, year) %>%
  mutate(BU_action = factor(paste(action, BUSINESS_UNIT))) %>%
  ggplot(aes(x=year, y=age)) +
  geom_point(aes(color=age), size=2.5) +
  scale_color_gradient("age", low = "blue", high = "red") +
  stat_smooth(method = lm, fullrange = TRUE, aes(fill=BU_action)) +
  scale_x_continuous(breaks = seq(min(htba$year), max(htba$year), 3)) +
  scale_fill_discrete(name = "Business Unit\non action") +
  labs(title = "Age Distribution") +
  facet_wrap(~BU_action)
```

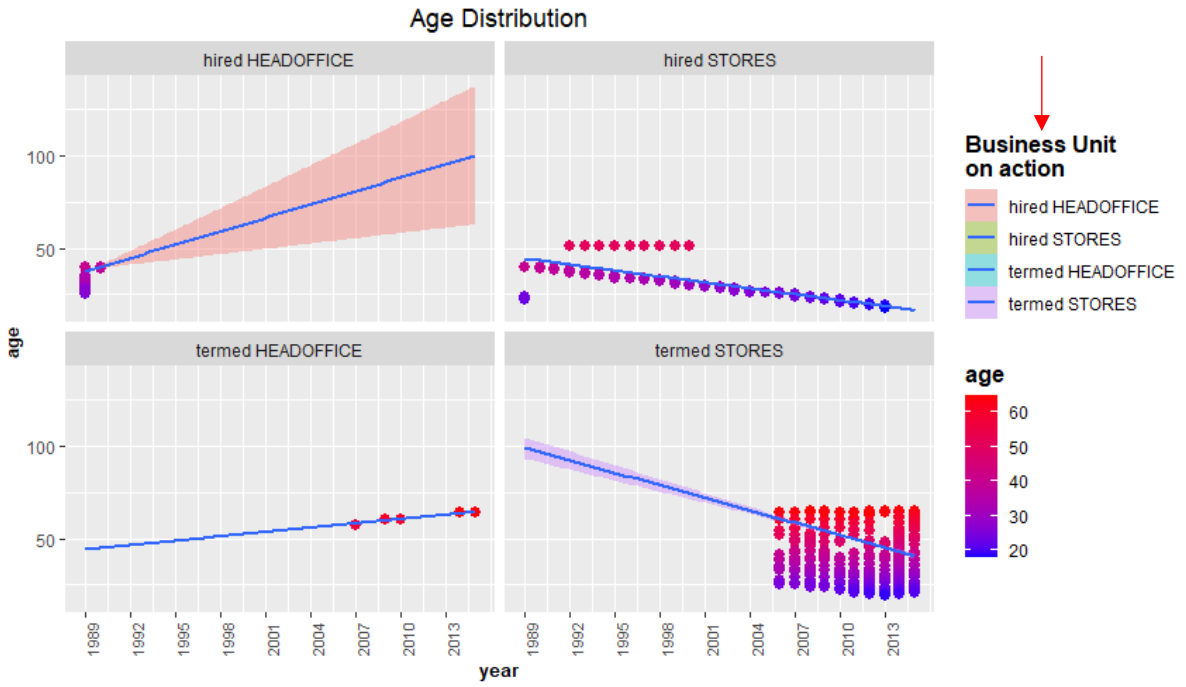


Figure 182 Graph using scale_fill_discrete()

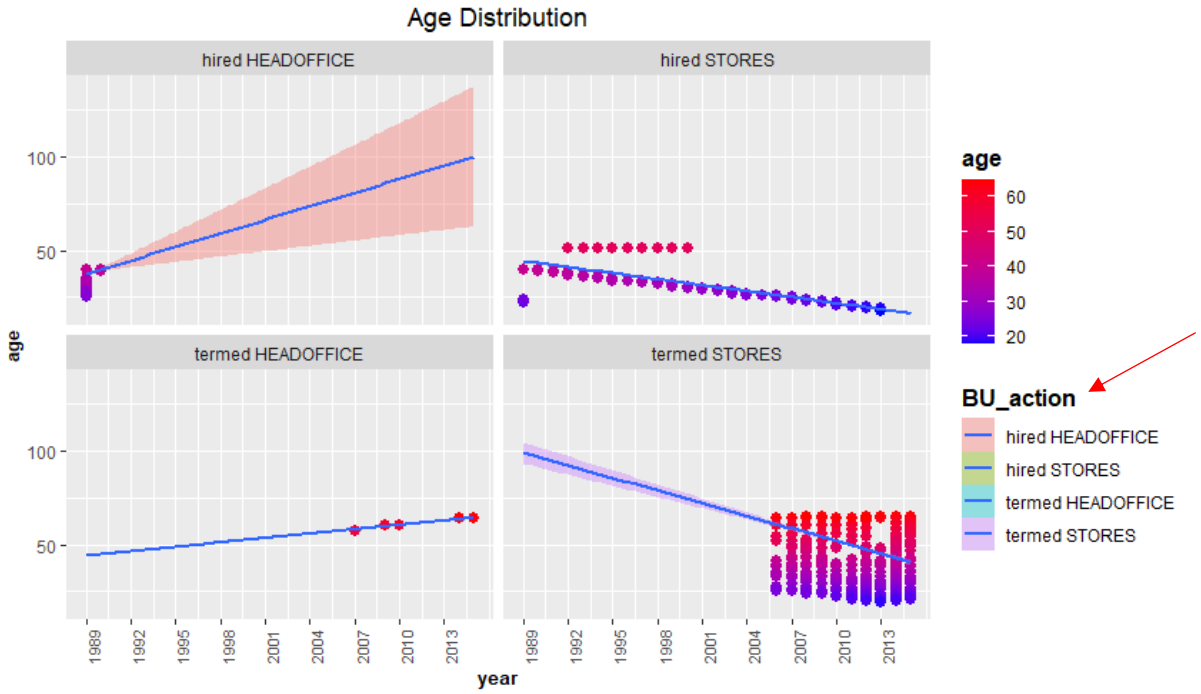


Figure 183 Without using scale_fill_discrete()

stat_summary()

This function comes from *ggplot2* package. It is used to summarise the *y* value based on unique *x* value and make labels on the graph (tidyverse, n.d.-o).

```
overall_age_box <- htba %>%
    mutate(BU_action = factor(paste(BUSINESS_UNIT, action))) %>%
    ggplot(aes(x=BU_action, y=age, fill=BU_action)) +
    geom_boxplot() +
    stat_summary(fun=mean, geom="point", aes(shape="mean"),
                 size=3, show.legend=TRUE) +
    stat_summary(fun=mean, geom="text",
                 hjust = -0.5, aes(label=round(..y.., digits=1)),
                 show.legend = FALSE) +
    scale_shape_manual("Statistics", values=c("mean"="x")) +
    scale_fill_discrete(name = "Business Unit\non\naction") +
    labs(title = "Age statistics by Business Unit",
         x = "Business Unit on Action") +
    theme(axis.text.x = element_text(angle=0))
```

Figure 184 Usage of stat_summary()

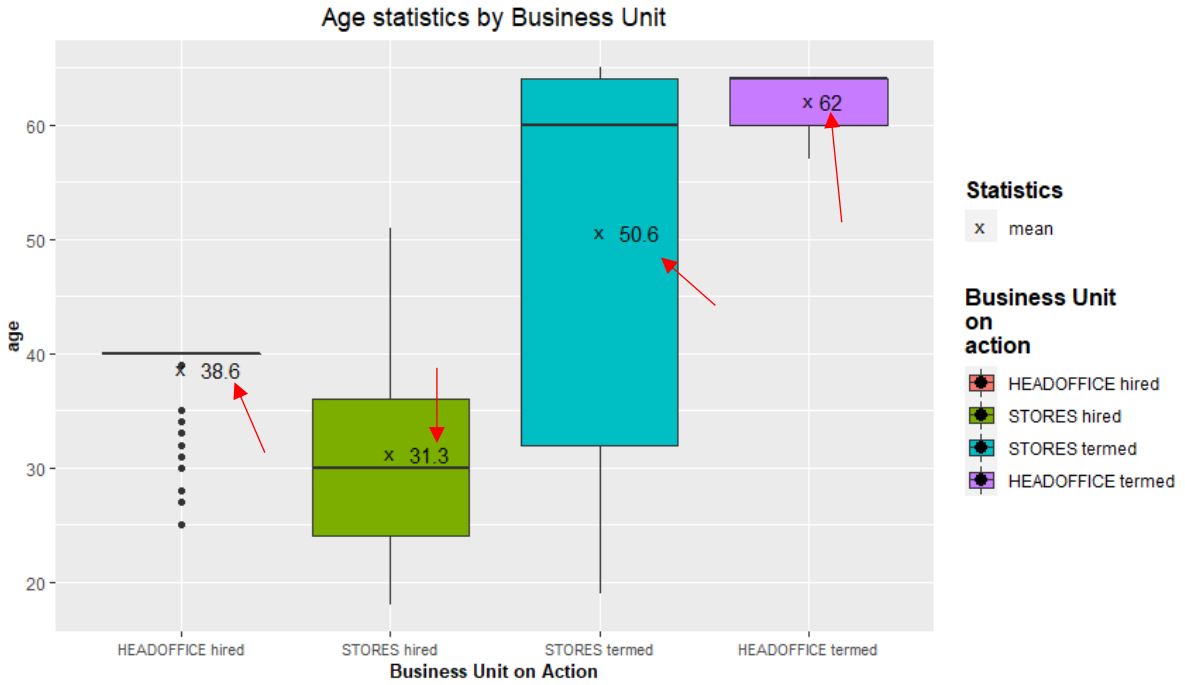


Figure 185 Graph using stat_summary()

scale_shape_manual()

This function comes from *ggplot2* package. It is used to customize the shape used on the graph (tidyverse, n.d.-c).

In this code snippet, we used *mean* as the shape aesthetic for *stat_summary*. Then, using *scale_shape_manual*, we can define the shape represented by *mean*, in this case, it is mapped to *x*.

```
overall_age_box <- htba %>%
    mutate(BU_action = factor(paste(BUSINESS_UNIT, action))) %>%
    ggplot(aes(x=BU_action, y=age, fill=BU_action)) +
    geom_boxplot() +
    stat_summary(fun=mean, geom="point", aes(shape="mean"),
                 size=3, show.legend=TRUE) +
    stat_summary(fun=mean, geom="text",
                 hjust = -0.5, aes(label=round(..y.., digits=1)),
                 show.legend = FALSE) +
    scale_shape_manual("Statistics", values=c("mean"="x")) +
    scale_fill_discrete(name = "Business Unit\non\naction") +
    labs(title = "Age statistics by Business Unit",
         x = "Business Unit on Action") +
    theme(axis.text.x = element_text(angle=0))
```

Figure 186 Usage of scale_shape_manual()

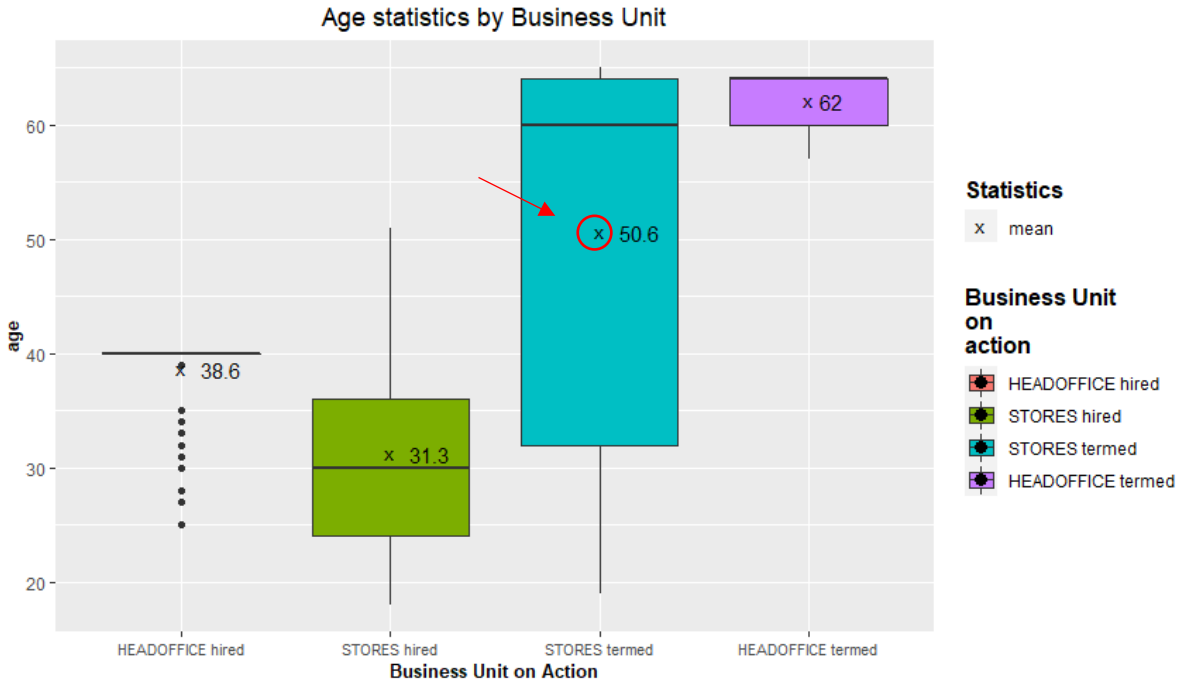


Figure 187 Graph using scale_shape_manual()

case_when()

This function comes from *dplyr* package. It has the similar functionality as *switch* statement. This function allows you to vectorise multiple if_else() statements. It is an R equivalent of the SQL CASE WHEN statement. If no cases match, NA is returned (tidyverse, n.d.-a).

```
hired_agegroup <- hired_by_age %>%
    mutate(agegroup = case_when(
        age < 20 ~ '<20',
        age < 30 ~ '20-29',
        age < 40 ~ '30-39',
        age < 50 ~ '40-49',
        age < 60 ~ '50-59',
        age < 70 ~ '60-69',
        age < 80 ~ '70-79',
        age < 90 ~ '80-89',
    )) %>%
    group_by(agegroup) %>%
    summarise(n = n())
```

Figure 188 Usage of case_when() function

```
> hired_by_age %>%
+     mutate(agegroup = case_when(
+         age < 20 ~ '<20',
+         age < 30 ~ '20-29',
+         age < 40 ~ '30-39',
+         age < 50 ~ '40-49',
+         age < 60 ~ '50-59',
+         age < 70 ~ '60-69',
+         age < 80 ~ '70-79',
+         age < 90 ~ '80-89',
+     ))
# A tibble: 6,284 x 5
# Groups:   year, BUSINESS_UNIT, EmployeeID [6,284]
    year BUSINESS_UNIT EmployeeID   age agegroup
   <dbl> <fct>              <int> <dbl> <chr>
 1  1989 HEADOFFICE          1318    35 30-39
 2  1989 HEADOFFICE          1319    32 30-39
 3  1989 HEADOFFICE          1320    34 30-39
 4  1989 HEADOFFICE          1321    30 30-39
 5  1989 HEADOFFICE          1322    31 30-39
 6  1989 HEADOFFICE          1323    27 20-29
 7  1989 HEADOFFICE          1325    25 20-29
 8  1989 HEADOFFICE          1328    33 30-39
 9  1989 HEADOFFICE          1332    34 30-39
10  1989 HEADOFFICE          1334    28 20-29
# ... with 6,274 more rows
```

Figure 189 Output using case_when() function

n()

This function comes from *dplyr* package. It is used to count the number of observations of a group and can only be used with *summarise()*, *mutate()*, and *filter()* (RDocumentation, n.d.-e).

```
hired_agegroup <- hired_by_age %>%
    mutate(agegroup = case_when(
        age < 20 ~ '<20',
        age < 30 ~ '20-29',
        age < 40 ~ '30-39',
        age < 50 ~ '40-49',
        age < 60 ~ '50-59',
        age < 70 ~ '60-69',
        age < 80 ~ '70-79',
        age < 90 ~ '80-89',
    )) %>%
    group_by(agegroup) %>%
    summarise(n = n())
```

Figure 190 Usage of n() function

```
> hired_agegroup
# A tibble: 5 x 2
  agegroup     n
  <chr>    <int>
1 <20        280
2 20-29     2695
3 30-39     2483
4 40-49      160
5 50-59      666
```

Figure 191 Output using n() function

slice()

This function comes from *dplyr* package. It is used to subset the data based on given indexes (tidyverse, n.d.-n).

In this code snippet, *which.max()* will return the indexes of the data and will be subsetted by *slice()*.

```
term_dept_n <- term_by_var(emp_table,
                           c("termreason_desc", "department_name")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n()) %>%
  ungroup %>%
  group_by(termreason_desc)

term_dept_n %>% slice(which.max(n))
term_dept_n %>% slice(which.min(n))
```

Figure 192 Usage of slice() function

```
> term_dept_n %>% slice(which.max(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc department_name      n
  <fct>           <fct>            <int>
1 Layoff          Customer Service    70
2 Resignaton      Customer Service   179
3 Retirement      Meats              317
```

Figure 193 Output using slice() function

which.max() and which.min()

These functions come from *base* package. It is used to retrieve the index of the specified condition, in this case, it is the max and min value (RDocumentation, n.d.-m).

```r
term_dept_n <- term_by_var(emp_table,
                           c("termreason_desc", "department_name")) %>%
  subset(termreason_desc != "Not Applicable") %>%
  summarise(n = n()) %>%
  ungroup %>%
  group_by(termreason_desc)

term_dept_n %>% slice(which.max(n))
term_dept_n %>% slice(which.min(n))
```

Figure 194 Usage of which.max() and which.min() functions

```
> term_dept_n %>% slice(which.max(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc department_name      n
  <fct>           <fct>            <int>
1 Layoff          Customer Service    70
2 Resignaton      Customer Service   179
3 Retirement      Meats              317
> ?which.max
> term_dept_n %>% slice(which.min(n))
# A tibble: 3 x 3
# Groups:   termreason_desc [3]
  termreason_desc department_name      n
  <fct>           <fct>            <int>
1 Layoff          Store Management     6
2 Resignaton      HR Technology        1
3 Retirement      Legal                3
```

Figure 195 Output using which.max() and which.min() functions

quantile()

This function comes from *stats* package. It is used to calculate the quantile of the data based on given probability (RDocumentation, n.d.-g).

In this code snippet, we supplied a probability of 0.8 which is equivalent to the 80

```r
hire_n_job_title %>% filter(n >= quantile(n, 0.8)) %>% arrange(desc(n))
```

Figure 196 Usage of quantile() function

```
> quantile(hire_n_job_title$n, 0.8)
 80%
32.8
```

Figure 197 Calculating 80-th percentile using quantile()

spread()

This function have the opposite effect of *pivot_longer()*. It converts data from long format to wide format (tidyverse, n.d.-m).

In the code snippet below, we are trying to spread the *gender_short* column into several columns and replace unknown value with 0.

```
hire_n_gender_job <- hire_by_var(emp_table, c("EmployeeID")) %>%
  slice(which.min(length_of_service)) %>%
  group_by(gender_short, job_title) %>%
  summarise(n = n())

spread_format <- hire_n_gender_job %>%
  spread(gender_short, n, fill=0)

colnames(spread_format) <- c("job", "Female", "Male")
spread_format <- cbind(spread_format,
                       total=spread_format$Female + spread_format$Male)
```

Figure 198 Usage of spread() function

```
> head(hire_n_gender_job)
# A tibble: 6 x 3
# Groups:   gender_short [1]
  gender_short job_title                     n
  <fct>        <fct>                     <int>
1 F            Accounting Clerk              5
2 F            Accounts Payable Clerk        2
3 F            Accounts Receiveable Clerk    2
4 F            Auditor                       1
5 F            Baker                       438
6 F            Bakery Manager               18
> head(spread_format)
                        job Female Male total
1           Accounting Clerk      5    0     5
2     Accounts Payable Clerk      2    1     3
3 Accounts Receiveable Clerk      2    2     4
4                   Auditor      1    2     3
5                     Baker    438  427   865
6             Bakery Manager     18   15    33
```

Figure 199 Before and after using spread() function

rename()

This function comes from *dplyr* package. This function can be used to rename variables using syntax *new_name = old_name* (tidyverse, n.d.-k).

```r
term_service <- service_df %>%
  filter(STATUS == "TERMINATED") %>%
  rename(service = length_of_service)
```

Figure 200 Usage of rename() function

```
> names(service_df)
 [1] "EmployeeID"          "recorddate_key"      "birthdate_key"
 [4] "orighiredate_key"    "terminationdate_key" "age"
 [7] "length_of_service"   "city_name"           "department_name"
[10] "job_title"           "store_name"          "gender_short"
[13] "gender_full"         "termreason_desc"     "termtype_desc"
[16] "STATUS_YEAR"         "STATUS"              "BUSINESS_UNIT"
> names(term_service)
 [1] "EmployeeID"          "recorddate_key"      "birthdate_key"
 [4] "orighiredate_key"    "terminationdate_key" "age"
 [7] "service"             "city_name"           "department_name"
[10] "job_title"           "store_name"          "gender_short"
[13] "gender_full"         "termreason_desc"     "termtype_desc"
[16] "STATUS_YEAR"         "STATUS"              "BUSINESS_UNIT"
```

Figure 201 Before and after renaming column using rename() functions

pull()

This function comes from *dplyr* package. This function is similar to extracting using dollar sign (*$*). It's mostly useful because it looks a little nicer in pipes, it also works with remote data frames, and it can optionally name the output (tidyverse, n.d.-e).

```r
term_service %>%
  ungroup %>%
  filter(service >= quantile(service, 0.9)) %>%
  pull(age) %>%
  summary
```

Figure 202 Usage of pull() function

```
> term_service %>%
+   ungroup %>%
+   filter(service >= quantile(service, 0.9)) %>%
+   pull(age)
 [1] 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65
[28] 65 64 65 65 65 65 65 65 65 65 65 65 65 65 65 65 64 65 65 65 65 65 65 65 65 65
[55] 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 64 64 64 64 64
[82] 64 64 64 64 64 64 64 64 64 64 63 62 62 62 61 62 61 61
```

Figure 203 Output of pull() function

References

RDocumentation. (n.d.-a). *aggregate function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/aggregate

RDocumentation. (n.d.-b). *interaction function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/interaction

RDocumentation. (n.d.-c). *labs function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/ggplot2/versions/3.3.5/topics/labs

RDocumentation. (n.d.-d). *merge function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/merge

RDocumentation. (n.d.-e). *n function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/dplyr/versions/0.7.8/topics/n

RDocumentation. (n.d.-f). *Position scales for continuous data (x & y) — scale_continuous • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/scale_continuous.html

RDocumentation. (n.d.-g). *quantile function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/quantile

RDocumentation. (n.d.-h). *round_any function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/plyr/versions/1.8.6/topics/round_any

RDocumentation. (n.d.-i). *str_replace function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/stringr/versions/1.4.0/topics/str_replace

RDocumentation. (n.d.-j). *tapply function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/tapply

RDocumentation. (n.d.-k). *time_length function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/lubridate/versions/1.8.0/topics/time_length

RDocumentation. (n.d.-l). *unique function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/unique

RDocumentation. (n.d.-m). *which.min function - RDocumentation*. Retrieved November 10, 2021, from https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/which.min

tidyverse. (n.d.-a). *A general vectorised if — case_when • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/case_when.html

tidyverse. (n.d.-b). *Complete themes — ggtheme • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/ggtheme.html

tidyverse. (n.d.-c). *Create your own discrete scale — scale_manual • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/scale_manual.html

tidyverse. (n.d.-d). *Efficiently count the number of unique values in a set of vectors — n_distinct • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/n_distinct.html

tidyverse. (n.d.-e). *Extract a single column — pull • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/pull.html

tidyverse. (n.d.-f). *Get, set, and modify the active theme — theme_get • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/theme_get.html

tidyverse. (n.d.-g). *Group by a selection of variables — group_by_all • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/group_by_all.html

tidyverse. (n.d.-h). *Group by one or more variables — group_by • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/group_by.html

tidyverse. (n.d.-i). *Modify components of a theme — theme • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/theme.html

tidyverse. (n.d.-j). *Polar coordinates — coord_polar • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/coord_polar.html

tidyverse. (n.d.-k). *Rename columns — rename • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/rename.html

tidyverse. (n.d.-l). *Scales for area or radius — scale_size • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/scale_size.html

tidyverse. (n.d.-m). *Spread a key-value pair across multiple columns — spread • tidyr*. Retrieved November 10, 2021, from https://tidyr.tidyverse.org/reference/spread.html

tidyverse. (n.d.-n). *Subset rows using their positions — slice • dplyr*. Retrieved November 10, 2021, from https://dplyr.tidyverse.org/reference/slice.html

tidyverse. (n.d.-o). *Summarise y values at unique/binned x — stat_summary_bin • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/stat_summary.html

tidyverse. (n.d.-p). *Theme elements — margin • ggplot2*. Retrieved November 10, 2021, from https://ggplot2.tidyverse.org/reference/element.html

Wickham, H. (n.d.). *Journal of Statistical Software Tidy Data*. Retrieved November 10, 2021, from http://www.jstatsoft.org/