DATA 608 Story 3

Fomba Kassoh

2024-11-04

Contents

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

# Load the data (BLS: Occupational Employment and Wage Statistics)
data <- read_csv('https://raw.githubusercontent.com/hawa1983/DATA-608/refs/heads/main/Story%204/state_da
```

```
## Rows: 453 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (10): state, state_abrev, ownership_type, occ_title, title, employment_t...
## lgl  (2): Pct_Ind_Employ, pct_rpt
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Replace non-numeric values (e.g., "*", "**") with NA in mean_salary, employment_total, and employment
data <- data %>%
  mutate(
    mean_salary = ifelse(grepl("[^0-9,]", mean_salary) | mean_salary == "*", NA, mean_salary),
    employment_total = ifelse(grepl("[^0-9,]", employment_total) | employment_total == "*", NA, employme
    employment_per_1000 = ifelse(grepl("[^0-9.]", employment_per_1000) | employment_per_1000 == "*", NA
  )

# Remove commas and convert columns to numeric
data <- data %>%
  mutate(
    mean_salary = as.numeric(gsub(",", "", mean_salary)),
    employment_total = as.numeric(gsub(",", "", employment_total)),
    employment_per_1000 = as.numeric(employment_per_1000)
  )

# Check for remaining NA values
na_counts <- data %>%
  summarise(
    mean_salary_NAs = sum(is.na(mean_salary)),
    employment_total_NAs = sum(is.na(employment_total)),
    employment_per_1000_NAs = sum(is.na(employment_per_1000))
  )
print(na_counts)
```

```
## # A tibble: 1 x 3
##   mean_salary_NAs employment_total_NAs employment_per_1000_NAs
##             <int>                <int>                   <int>
## 1               1                    4                       4
```

```r
# Define neighboring states for imputation (partial list; add all as needed)
neighbors <- list(
  "MT" = c("ID", "WY", "SD", "ND"),
  "MS" = c("TN", "AR", "LA", "AL"),
  "OK" = c("TX", "KS", "MO", "AR"),
  "KS" = c("NE", "MO", "OK", "CO"),
  "WY" = c("MT", "SD", "NE", "CO", "UT", "ID")
  # Add remaining states as needed
)

# Function to impute missing values based on neighboring states
impute_missing_value <- function(state_abrev, variable, data, neighbors_list) {
  # Get neighbors for the state
  neighbor_states <- neighbors_list[[state_abrev]]

  # If there are no neighbors, return NA
  if (is.null(neighbor_states) || length(neighbor_states) == 0) {
    return(NA)
  }

  # Get the values of the neighboring states for the specified variable
  neighbor_values <- data %>%
    filter(state_abrev %in% neighbor_states) %>%
    pull(!!sym(variable))

  # Return the mean of neighboring states, ignoring NAs
  if (all(is.na(neighbor_values))) {
    return(NA)
  } else {
    return(mean(neighbor_values, na.rm = TRUE))
  }
}

# Apply imputation for mean_salary, employment_total, and employment_per_1000
data <- data %>%
  mutate(
    mean_salary = ifelse(
      is.na(mean_salary),
      sapply(state_abrev, impute_missing_value, "mean_salary", data, neighbors),
      mean_salary
    ),
    employment_total = ifelse(
      is.na(employment_total),
      sapply(state_abrev, impute_missing_value, "employment_total", data, neighbors),
      employment_total
    ),
    employment_per_1000 = ifelse(
      is.na(employment_per_1000),
      sapply(state_abrev, impute_missing_value, "employment_per_1000", data, neighbors),
      employment_per_1000
    )
  )

# Check for remaining NA values in mean_salary, employment_total, and employment_per_1000
remaining_na_counts <- data %>%
  summarise(
    mean_salary_NAs = sum(is.na(mean_salary)),
    employment_total_NAs = sum(is.na(employment_total)),
    employment_per_1000_NAs = sum(is.na(employment_per_1000))
  )

# Print the count of remaining NAs in each column
print(remaining_na_counts)
```

```
## # A tibble: 1 x 3
##   mean_salary_NAs employment_total_NAs employment_per_1000_NAs
##             <int>                <int>                   <int>
## 1               0                    0                       0
```

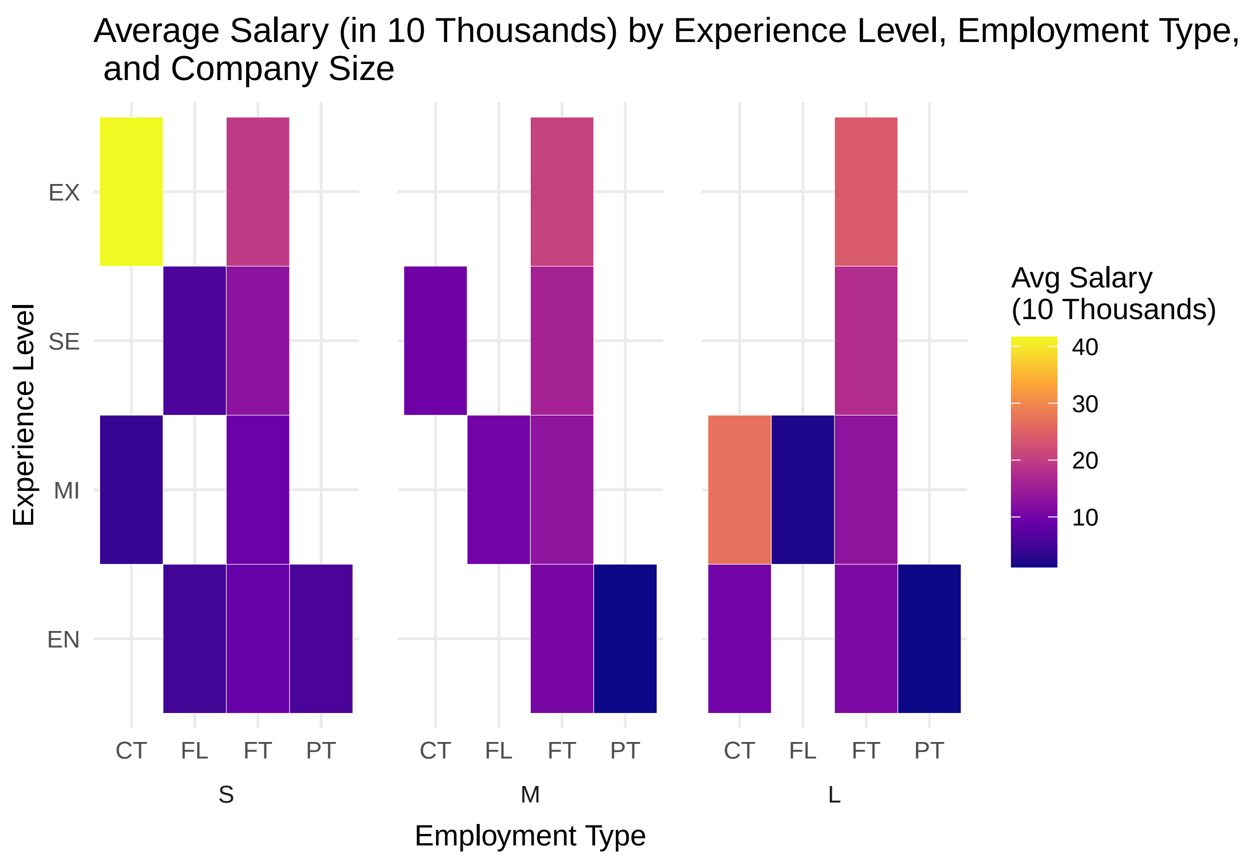
```
glimpse(data)
```

```
## Rows: 453
## Columns: 12
## $ state                              <chr> "Alabama", "Alaska", "Arizona", "~
## $ state_abrev                        <chr> "AL", "AK", "AZ", "AR", "CA", "CO~
## $ ownership_type                     <chr> "Federal, State, and Local Govern~
## $ occ_title                          <chr> "Computer and Information Systems~
## $ title                              <chr> "Data Architect", "Data Architect~
## $ employment_total                   <dbl> 5800, 640, 14180, 3230, 98430, 12~
## $ employment_per_1000                <dbl> 2.827, 2.045, 4.531, 2.540, 5.485~
## $ occupation_share_per_area_employment <chr> "0.72", "0.52", "1.16", "0.65", "~
## $ Pct_Ind_Employ                     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ pct_rpt                            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ mean_hourly_wage                   <chr> "67.92", "65.87", "78.8", "59.2",~
## $ mean_salary                        <dbl> 141270, 137010, 163900, 123130, 2~
```

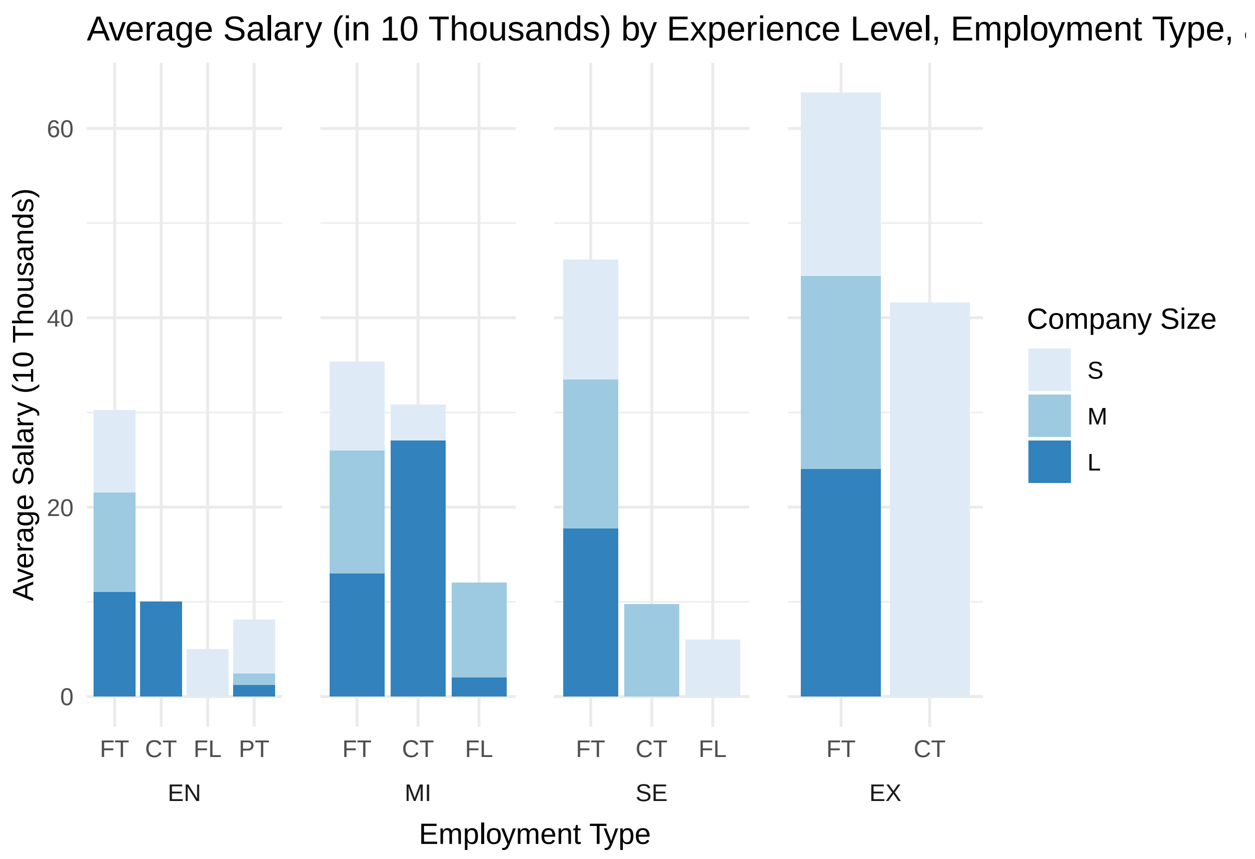
```
# Load the data (Source Kagle. No state information)
us_data <- read_csv('https://raw.githubusercontent.com/hawa1983/DATA-608/refs/heads/main/Story%204/ds_sa
```

```
## Rows: 3032 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): experience_level, employment_type, job_title, job_category, company...
## dbl (3): work_year, salary, remote_ratio
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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

# Prepare data for heatmap
heatmap_data <- us_data %>%
  mutate(
    company_size = factor(company_size, levels = c("S", "M", "L")),
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  ) %>%
  group_by(experience_level, employment_type, company_size) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE) / 10000, .groups = "drop")  # Divide by 10,000

# Plot heatmap
ggplot(heatmap_data, aes(x = employment_type, y = experience_level, fill = avg_salary)) +
  geom_tile(color = "white") +
  facet_wrap(~ company_size, ncol = 3, strip.position = "bottom") +
  scale_fill_viridis_c(option = "plasma", name = "Avg Salary\n(10 Thousands)") +
  labs(
    title = "Average Salary (in 10 Thousands) by Experience Level, Employment Type,\n and Company Size"
    x = "Employment Type",
    y = "Experience Level"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    strip.placement = "outside",
    panel.spacing = unit(1, "lines")
  )
```

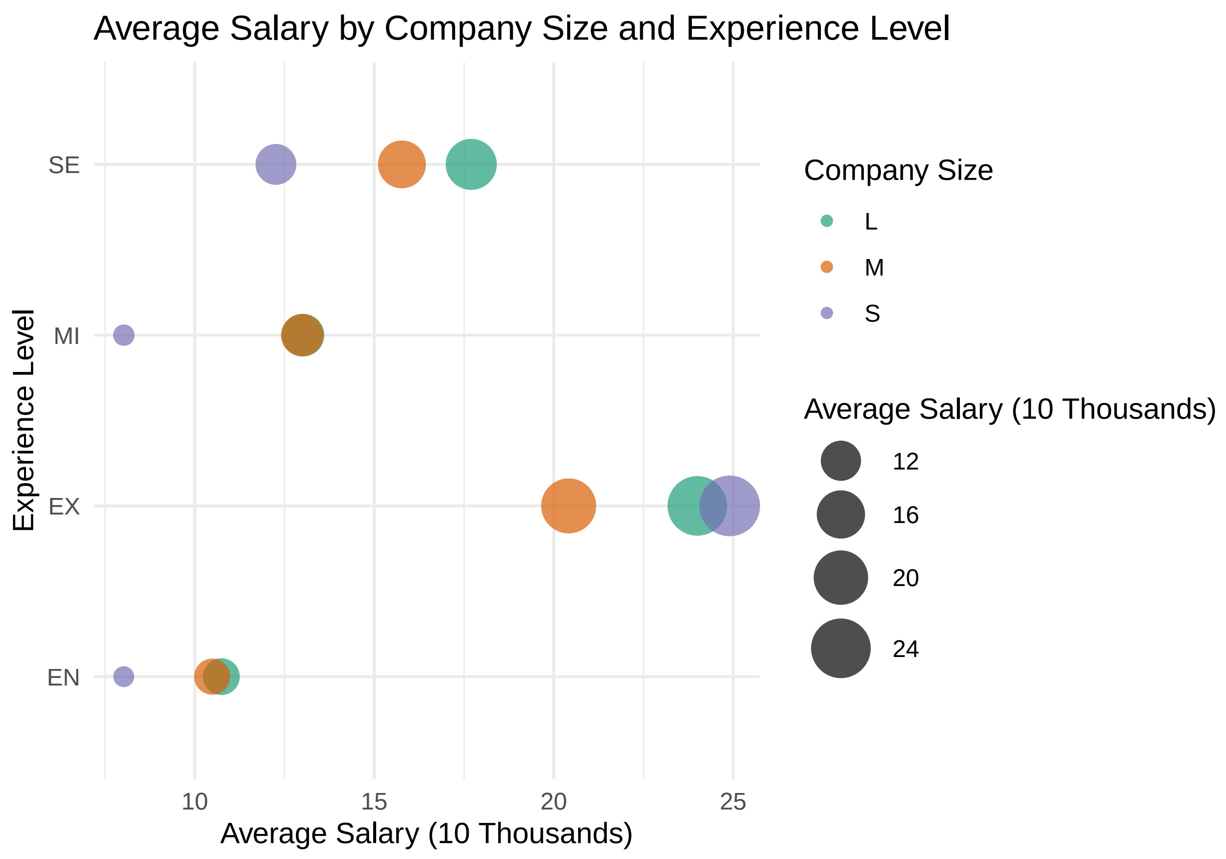


```r
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Prepare data for stacked bar plot with descending order within each experience level
stacked_data <- us_data %>%
  mutate(
    company_size = factor(company_size, levels = c("S", "M", "L")),
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  ) %>%
  group_by(experience_level, employment_type, company_size) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE) / 10000, .groups = "drop")  # Divide by 10,000 for

# Calculate total salary per employment type within each experience level for sorting
sorted_data <- stacked_data %>%
  group_by(experience_level, employment_type) %>%
  summarise(total_salary = sum(avg_salary), .groups = "drop") %>%
  arrange(experience_level, desc(total_salary)) %>%
  mutate(employment_type = factor(employment_type, levels = unique(employment_type))) # Order employmen

# Join sorted order back to stacked_data
stacked_data <- left_join(stacked_data, sorted_data, by = c("experience_level", "employment_type"))

# Plot stacked bar chart with reordered employment types
ggplot(stacked_data, aes(x = reorder(employment_type, -total_salary), y = avg_salary, fill = company_si
  geom_bar(stat = "identity", position = "stack") +
  facet_wrap(~ experience_level, ncol = 4, scales = "free_x", strip.position = "bottom") +
  scale_fill_brewer(palette = "Blues") +
  labs(
    title = "Average Salary (in 10 Thousands) by Experience Level, Employment Type, and Company Size",
    x = "Employment Type",
    y = "Average Salary (10 Thousands)",
    fill = "Company Size"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    strip.placement = "outside",
    panel.spacing = unit(1, "lines")
  )
```



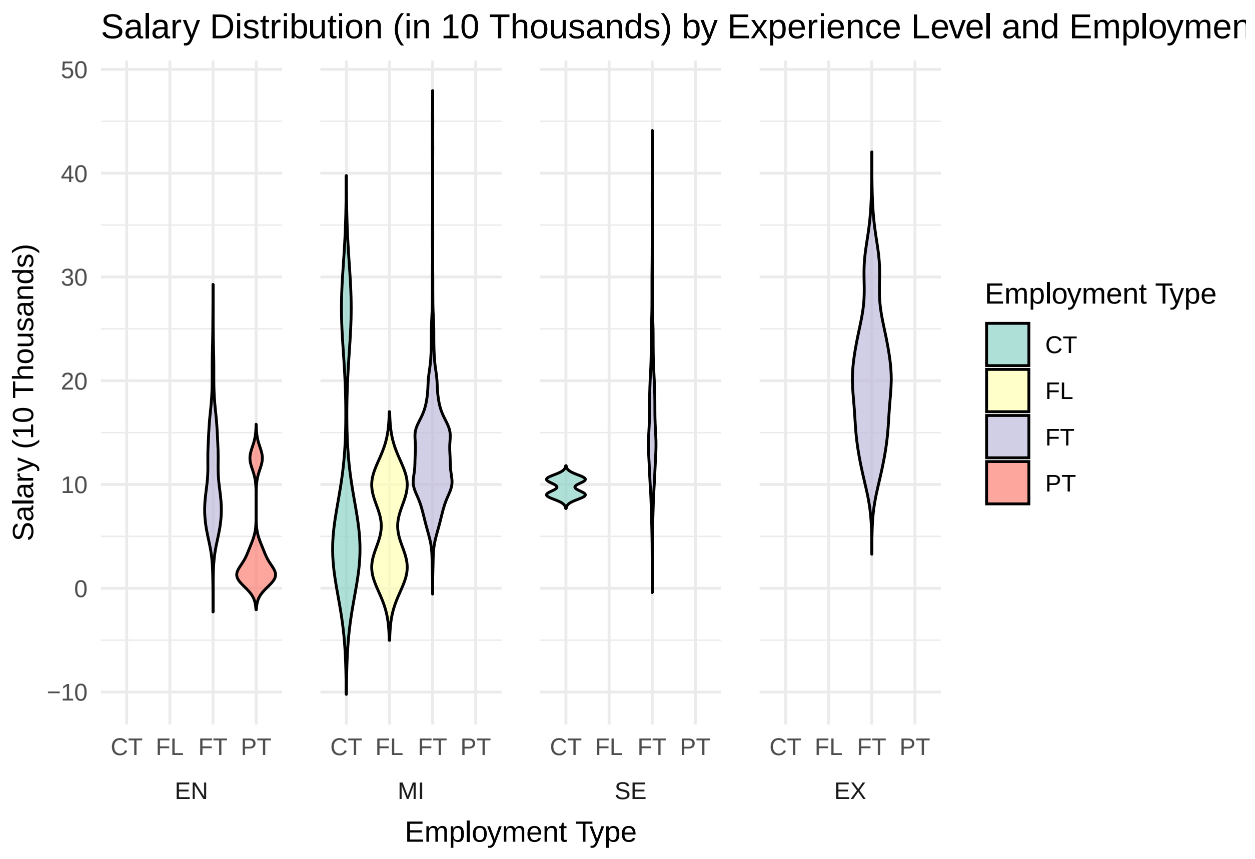
```r
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Prepare data for bubble plot
bubble_data <- us_data %>%
  group_by(company_size, experience_level) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE), .groups = "drop") %>%
  mutate(avg_salary_scaled = avg_salary / 10000)  # Scale salary to tens of thousands for display
```

```r
# Plot bubble chart
ggplot(bubble_data, aes(x = avg_salary_scaled, y = experience_level, size = avg_salary_scaled, color = c
  geom_point(alpha = 0.7) +
  scale_size(range = c(3, 10)) +
  scale_color_brewer(palette = "Dark2") +
  labs(
    title = "Average Salary by Company Size and Experience Level",
    x = "Average Salary (10 Thousands)",
    y = "Experience Level",
    size = "Average Salary (10 Thousands)",
    color = "Company Size"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5)  # Keep x-axis labels horizontal
  )
```



```r
# Prepare data for violin plot
violin_data <- us_data %>%
  mutate(
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  )

# Plot violin chart
ggplot(violin_data, aes(x = employment_type, y = salary / 10000, fill = employment_type)) +
  geom_violin(trim = FALSE, color = "black", alpha = 0.7) +
  facet_wrap(~ experience_level, ncol = 4, strip.position = "bottom") +
  scale_fill_brewer(palette = "Set3") +
  labs(
    title = "Salary Distribution (in 10 Thousands) by Experience Level and Employment Type",
    x = "Employment Type",
    y = "Salary (10 Thousands)",
    fill = "Employment Type"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    strip.placement = "outside",
    panel.spacing = unit(1, "lines")
  )
```

```
## Warning: Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
```



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

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

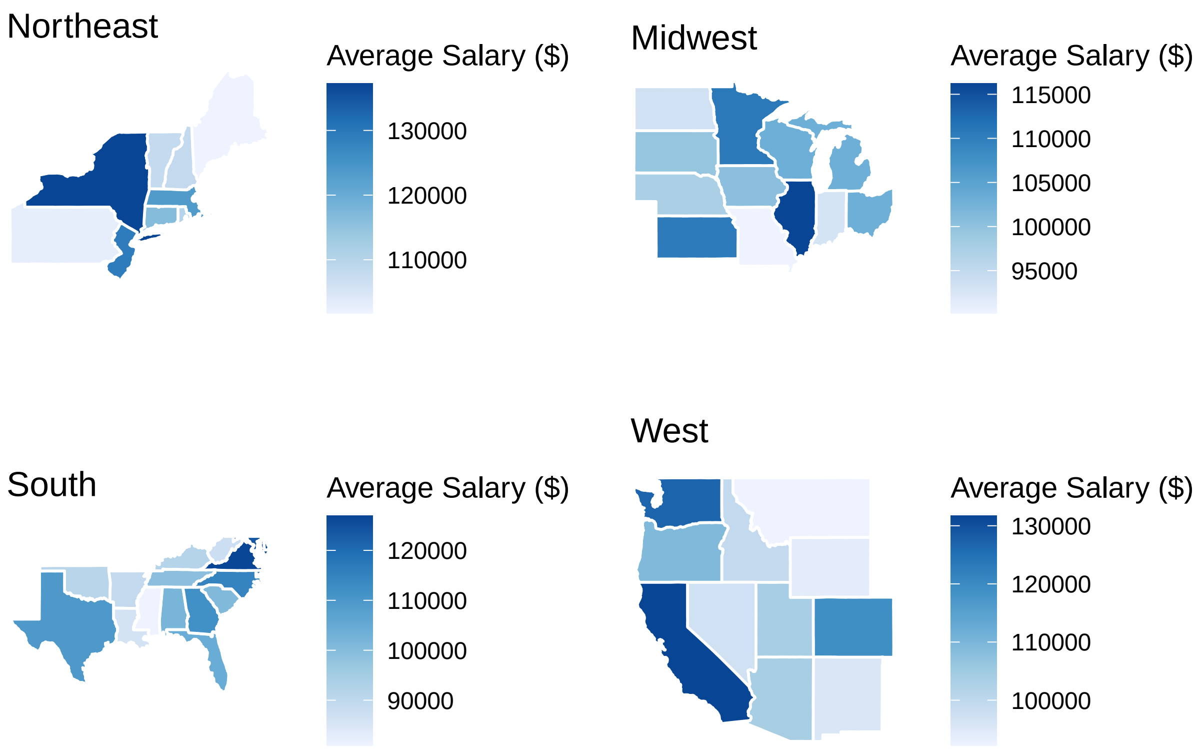
```
library(scales)
```

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(tidytext)

# Define region mapping for each state abbreviation
region_mapping <- data.frame(
  state_abrev = c("CT", "ME", "MA", "NH", "RI", "VT", "NJ", "NY", "PA",   # Northeast
                  "IL", "IN", "MI", "OH", "WI",                           # Midwest
                  "IA", "KS", "MN", "MO", "NE", "ND", "SD",               # Midwest
                  "DE", "FL", "GA", "MD", "NC", "SC", "VA", "WV",         # South Atlantic
                  "AL", "KY", "MS", "TN",                                 # East South Central
                  "AR", "LA", "OK", "TX",                                 # West South Central
                  "AZ", "CO", "ID", "MT", "NV", "NM", "UT", "WY",         # Mountain
                  "AK", "CA", "HI", "OR", "WA"),                          # Pacific
  region = c("Northeast", "Northeast", "Northeast", "Northeast", "Northeast", "Northeast", "Northeast",
             "Midwest", "Midwest", "Midwest", "Midwest", "Midwest",
             "Midwest", "Midwest", "Midwest", "Midwest", "Midwest", "Midwest", "Midwest",
             "South", "South", "South", "South", "South", "South", "South", "South",
             "South", "South", "South", "South",
             "South", "South", "South", "South",
             "West", "West", "West", "West", "West", "West", "West", "West",
             "West", "West", "West", "West", "West")
)

# Load state boundaries data and rename columns for clarity
states_map <- map_data("state") %>%
  rename(state_name = region) %>%
  mutate(state_name = tolower(state_name))  # Ensure all state names are in lowercase

# Prepare the average salary data and merge with region information
average_salary_data <- data %>%
```

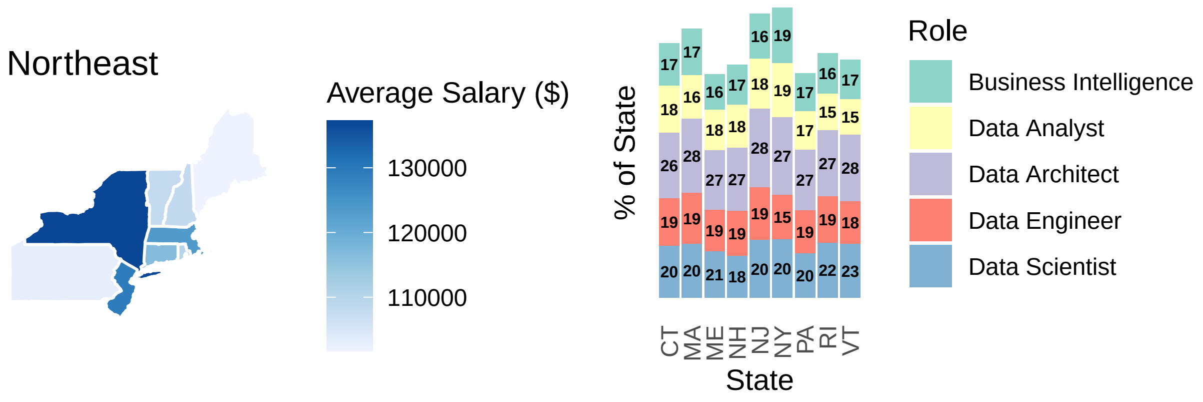
```r
  filter(state_abrev != "DC") %>%  # Exclude Washington, D.C.
  group_by(state_abrev) %>%
  summarise(mean_salary = mean(mean_salary, na.rm = TRUE)) %>%  # Calculate mean salary per state
  left_join(region_mapping, by = "state_abrev")  # Join with region mapping

# Map state abbreviations to lowercase state names for join with `states_map`
state_abbreviation_mapping <- data.frame(
  state_name = tolower(state.name),
  state_abrev = state.abb
)

# Merge the average salary data with the state names
average_salary_data <- average_salary_data %>%
  left_join(state_abbreviation_mapping, by = "state_abrev")  # Add state_name for compatibility with `s

# Function to filter and plot each region separately (Map Plot) with the "Blues" palette
plot_region <- function(region_name) {
  # Filter `average_salary_data` for the specified region
  region_salary_data <- average_salary_data %>% filter(region == region_name)

  # Join `states_map` with the filtered salary data by state name
  region_states <- states_map %>%
    inner_join(region_salary_data, by = c("state_name"))

  # Check if there is data for the specified region
  if (nrow(region_states) == 0) {
    message(paste("No data available for region:", region_name))
    return(NULL)
  }

  # Create a map for the selected region with the "Blues" palette
  ggplot() +
    geom_polygon(data = region_states, aes(x = long, y = lat, group = group, fill = mean_salary), color
    scale_fill_distiller(palette = "Blues", direction = 1, name = "Average Salary ($)") +
    labs(title = paste("", region_name)) +
    theme_minimal() +
    theme(
      legend.position = "right",
      axis.title = element_blank(),
      axis.text = element_blank(),
      axis.ticks = element_blank(),
      panel.grid = element_blank()
    ) +
    coord_fixed(1.3)
}

# Generate map plots for each region
plot_northeast <- plot_region("Northeast")
plot_midwest <- plot_region("Midwest")
plot_south <- plot_region("South")
plot_west <- plot_region("West")

# Display each region map plot individually or arrange them as needed
```

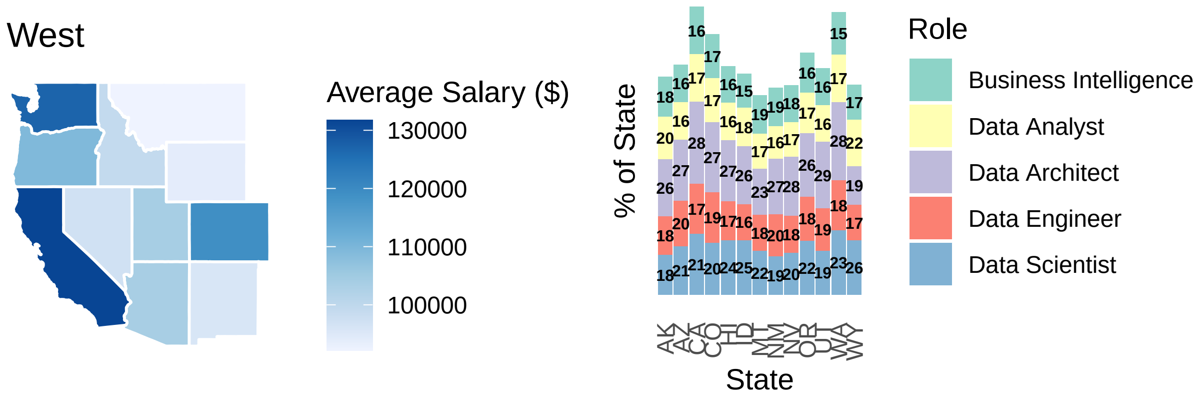
```r
grid.arrange(plot_northeast, plot_midwest, plot_south, plot_west, ncol = 2)
```



```r
# Filtering and preparing data for bar chart with suppressed warnings
region_data <- data %>%
  filter(state_abrev != "DC") %>%  # Exclude Washington, D.C.
  left_join(region_mapping, by = "state_abrev") %>%
  select(-starts_with("region."))  # Remove any extra region columns if they exist

stacked_data <- suppressWarnings(
  region_data %>%
    # Explicitly convert mean_salary to numeric and region to character before filtering
    mutate(
      mean_salary = as.numeric(mean_salary),
      region = as.character(region)
    ) %>%
    filter(!is.na(mean_salary) & !is.na(region)) %>%  # Ensure no missing values in key columns
    group_by(state_abrev, title) %>%
    summarise(
      mean_salary = mean(mean_salary, na.rm = TRUE),  # Calculate mean of numeric mean_salary
      .groups = "drop"  # Ungroup after summarizing to avoid grouping issues
    ) %>%
    # Calculate the total salary per state to use for percentage calculation
    group_by(state_abrev) %>%
    mutate(total_salary = sum(mean_salary, na.rm = TRUE)) %>%  # Ensure total_salary is numeric
    ungroup() %>%
    # Calculate the percentage for each role within the state
    mutate(
      salary_percent = round((mean_salary / total_salary) * 100)  # Calculate percentage
    ) %>%
    # Join with region information and select the relevant columns
    left_join(region_mapping, by = "state_abrev") %>%
    select(state_abrev, title, mean_salary, salary_percent, region) %>%  # Select relevant columns
    arrange(state_abrev)  # Sort by state_abrev
)



# Function to create bar chart plots for each region with suppressWarnings inside
plot_bar_region <- function(region_name) {
  region_data <- stacked_data %>% filter(region == region_name)

  suppressWarnings({
    ggplot(region_data, aes(x = reorder(state_abrev, state_abrev), y = mean_salary, fill = title)) +
      geom_bar(stat = "identity") +
      geom_text(aes(label = salary_percent),
                position = position_stack(vjust = 0.5),
                color = "black", size = 2, fontface = "bold") +  # Annotate with percentage at the cent
      labs(
        x = "State",
        y = "% of State",
        fill = "Role"
      ) +
      theme_minimal() +
      theme(
        axis.text.y = element_blank(),  # Remove y-axis text
        axis.ticks.y = element_blank(),  # Remove y-axis ticks
        panel.grid.major.y = element_blank(),  # Remove major grid lines for y-axis
        panel.grid.minor.y = element_blank(),   # Remove minor grid lines for y-axis
        axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5),  # Rotate state labels for read
        panel.grid.major.x = element_blank(),  # Remove major grid lines for x-axis
        panel.grid.minor.x = element_blank()   # Remove minor grid lines for x-axis
      ) +
      scale_fill_brewer(palette = "Set3")  # Use a color palette for distinct role colors
  })
}


# Generate bar chart plots for each region
plotbar_northeast <- plot_bar_region("Northeast")
plotbar_midwest <- plot_bar_region("Midwest")
plotbar_south <- plot_bar_region("South")
plotbar_west <- plot_bar_region("West")

# Arrange and display plots in 2x2 grid
grid.arrange(plot_northeast, plotbar_northeast, plot_west, plotbar_west, ncol = 2)
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```



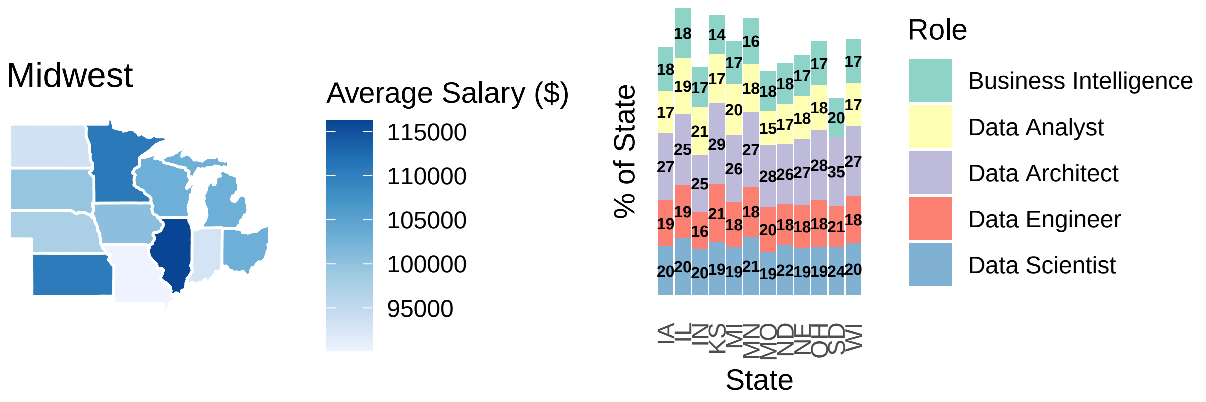


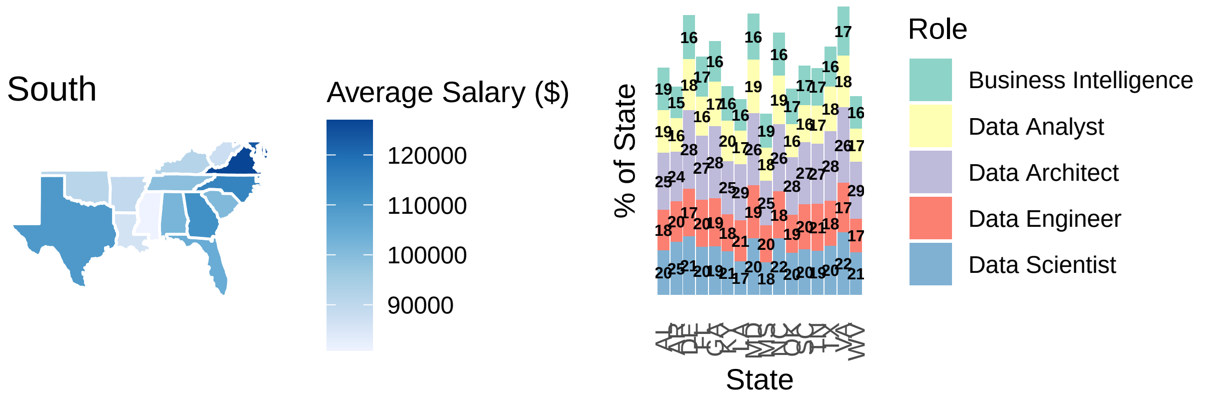
```r
grid.arrange(plot_midwest, plotbar_midwest, plot_south, plotbar_south,  ncol = 2)
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
```

```
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
```

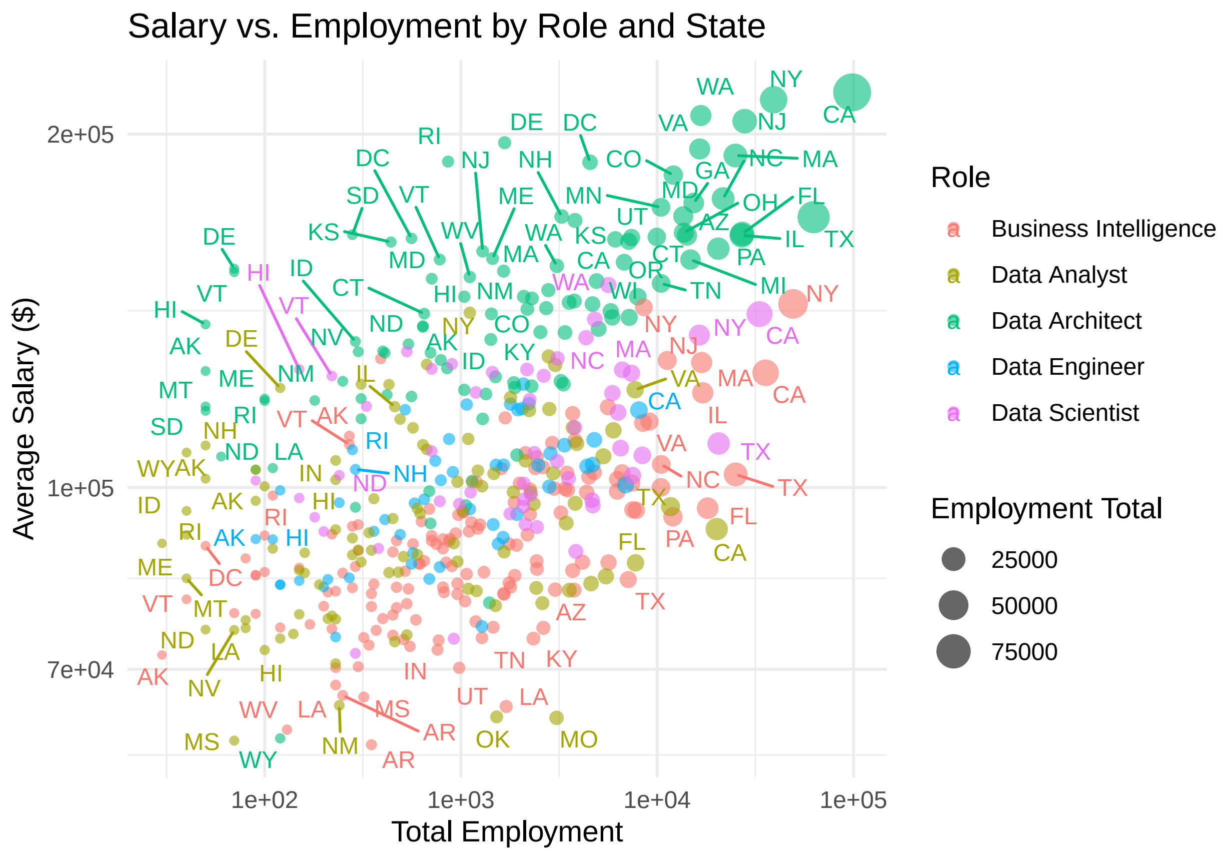
```
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```





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

# Bubble chart of salary vs employment with role, state coloring, and state labels
ggplot(data, aes(x = employment_total, y = mean_salary, color = title, size = employment_total)) +
  geom_point(alpha = 0.6) +
  scale_x_log10() +
  scale_y_log10() +
  geom_text_repel(aes(label = state_abrev), size = 3, max.overlaps = 15) +  # Add state labels with rep
  labs(
    title = "Salary vs. Employment by Role and State",
    x = "Total Employment",
    y = "Average Salary ($)",
    color = "Role",
    size = "Employment Total"
  ) +
  theme_minimal() +
  theme(legend.position = "right")
```

```
## Warning: ggrepel: 324 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

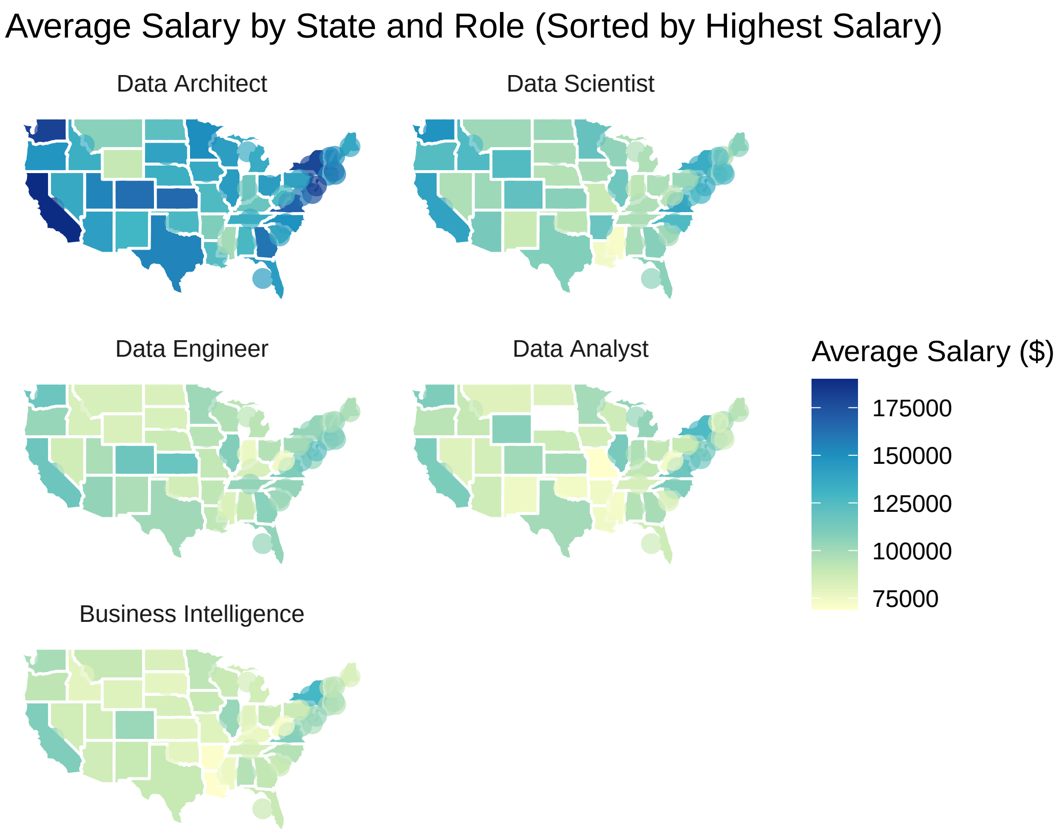


```
# Load necessary libraries
library(dplyr)
library(ggplot2)
library(maps)

# Assume 'data' is already loaded and contains salary information

# Step 1: Aggregate data by state and role (title)
role_data <- data %>%
  filter(state_abrev != "DC") %>%  # Exclude Washington, D.C.
  group_by(state_abrev, title) %>%
  summarise(
    mean_salary = mean(mean_salary, na.rm = TRUE),
    employment_total = sum(employment_total, na.rm = TRUE),
    .groups = "drop"
  )

# Step 2: Calculate overall average salary by title
title_salary_rank <- role_data %>%
  group_by(title) %>%
  summarise(avg_salary = mean(mean_salary, na.rm = TRUE)) %>%
  arrange(desc(avg_salary))  # Sort by highest average salary

# Step 3: Load the state map data
states_map <- map_data("state") %>%
  filter(region != "district of columbia")  # Exclude Washington, D.C. from map data
```

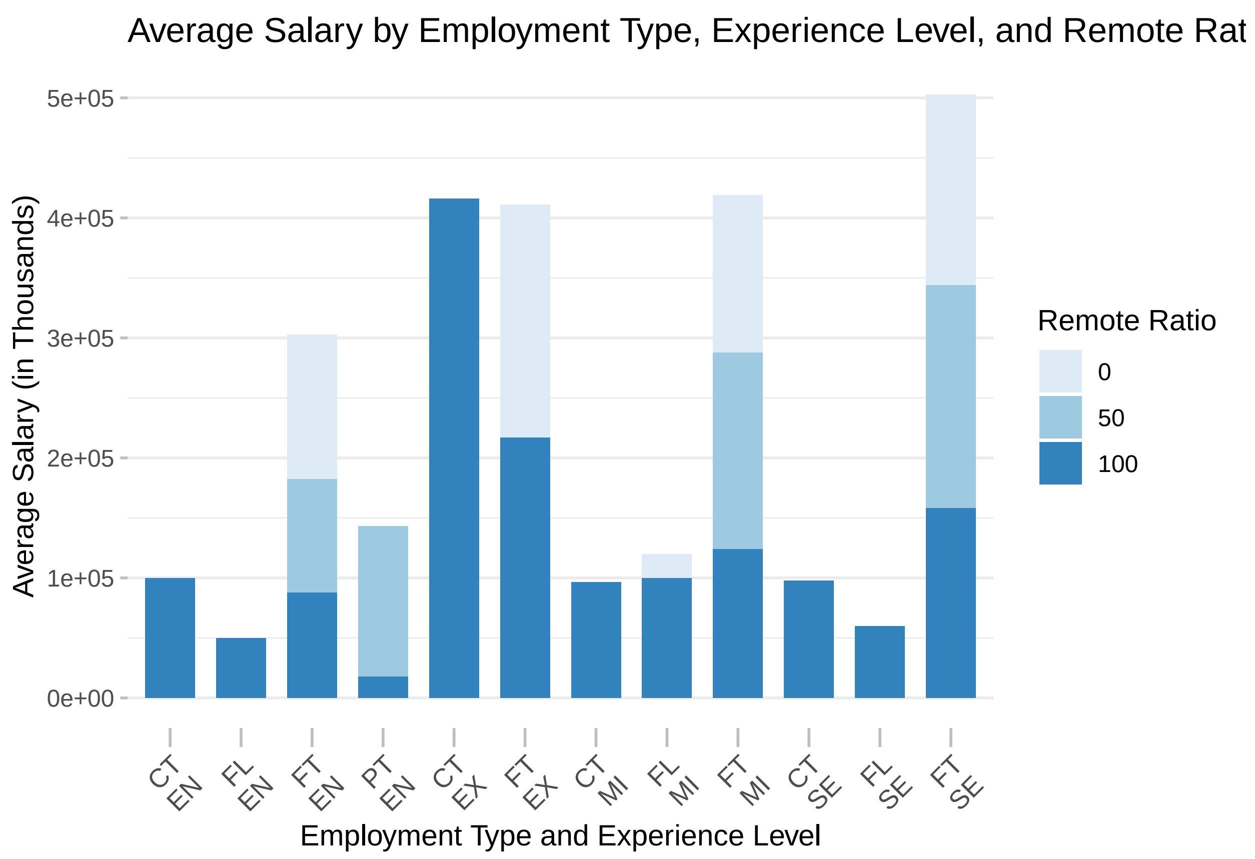
```r
# Step 4: Calculate centroids of states
state_centroids <- states_map %>%
  group_by(region) %>%
  summarize(
    long = mean(range(long)),
    lat = mean(range(lat))
  ) %>%
  filter(region != "district of columbia")  # Ensure D.C. is excluded

# Step 5: Map state names to abbreviations
state_abbreviation_mapping <- data.frame(
  region = tolower(state.name),
  state_abrev = state.abb
) %>%
  filter(region != "district of columbia")  # Exclude D.C. from abbreviation mapping

# Step 6: Merge centroids with role data
merged_data_centroids <- state_centroids %>%
  left_join(state_abbreviation_mapping, by = "region") %>%
  left_join(role_data, by = c("state_abrev" = "state_abrev")) %>%
  filter(!is.na(title))  # Remove any rows with missing titles due to exclusion

# Step 7: Merge role data with map data
map_data <- states_map %>%
  left_join(state_abbreviation_mapping, by = "region") %>%
  left_join(role_data, by = "state_abrev") %>%
  filter(!is.na(title))  # Remove any rows with missing titles
```

```
## Warning in left_join(., role_data, by = "state_abrev"): Detected an unexpected many-to-many relation
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 6 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```r
# Step 8: Reorder 'title' factor by salary rank
map_data$title <- factor(map_data$title, levels = title_salary_rank$title)
merged_data_centroids$title <- factor(merged_data_centroids$title, levels = title_salary_rank$title)

# Create the faceted geographic heatmap
ggplot() +
  # Fill states by mean salary for each role
  geom_polygon(data = map_data, aes(x = long, y = lat, group = group, fill = mean_salary), color = "whi

  # Overlay points at state centroids for each role
  geom_point(data = merged_data_centroids, aes(x = long, y = lat, color = mean_salary), size = 3, alpha

  # Use a color scale for mean salary
  scale_fill_distiller(palette = "YlGnBu", direction = 1, name = "Average Salary ($)") +
  scale_color_distiller(palette = "YlGnBu", direction = 1, name = "Average Salary ($)") +

  # Facet by job title, ordered by highest average salary
  facet_wrap(~ title, ncol = 2) +  # Adjust 'ncol' as needed for layout
  # Fix aspect ratio
  coord_fixed(1.3) +

  # Add title and theme settings
  labs(title = "Average Salary by State and Role (Sorted by Highest Salary)") +
  theme_minimal() +
  theme(
    legend.position = "right",
    axis.title = element_blank(),
    axis.text = element_blank(),
    axis.ticks = element_blank(),
    panel.grid = element_blank()
  )
```

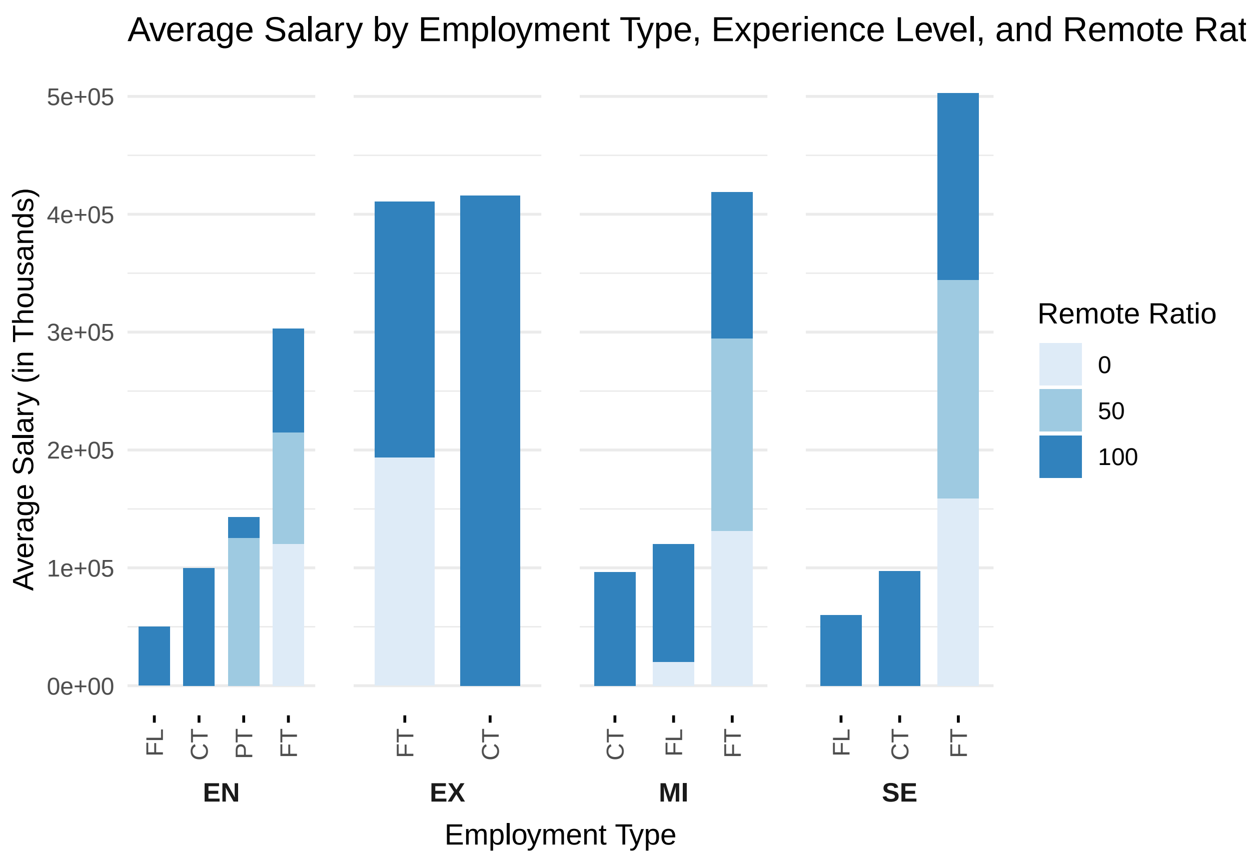


```r
# Load necessary libraries
library(dplyr)
library(ggplot2)

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

# Group and summarize data (using your dataset)
avg_salary_data <- us_data %>%
  group_by(employment_type, experience_level, remote_ratio) %>%
  summarise(avg_salary = mean(salary), .groups = "drop")

# Create a combined x-axis label with both `employment_type` and `experience_level`
avg_salary_data <- avg_salary_data %>%
  mutate(x_axis = interaction(employment_type, experience_level, sep = " - "))

# Plot the bar chart
ggplot(avg_salary_data, aes(x = x_axis, y = avg_salary, fill = factor(remote_ratio))) +
  geom_bar(stat = "identity", position = "stack", width = 0.7) +
  labs(
    title = "Average Salary by Employment Type, Experience Level, and Remote Ratio",
    x = "Employment Type and Experience Level",
    y = "Average Salary (in Thousands)",
    fill = "Remote Ratio"
  ) +
  scale_fill_brewer(palette = "Blues") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10), # Only specified once
    panel.grid.major.x = element_blank(),
    panel.spacing = unit(1, "lines"),
    axis.ticks.length.x = unit(0.25, "cm"),
    axis.ticks = element_line(color = "grey")
  ) +
  scale_x_discrete(labels = function(x) {
    # Split labels for employment_type on top and experience_level below
    sapply(strsplit(as.character(x), " - "), function(lbl) paste(lbl, collapse = "\n"))
  })
```



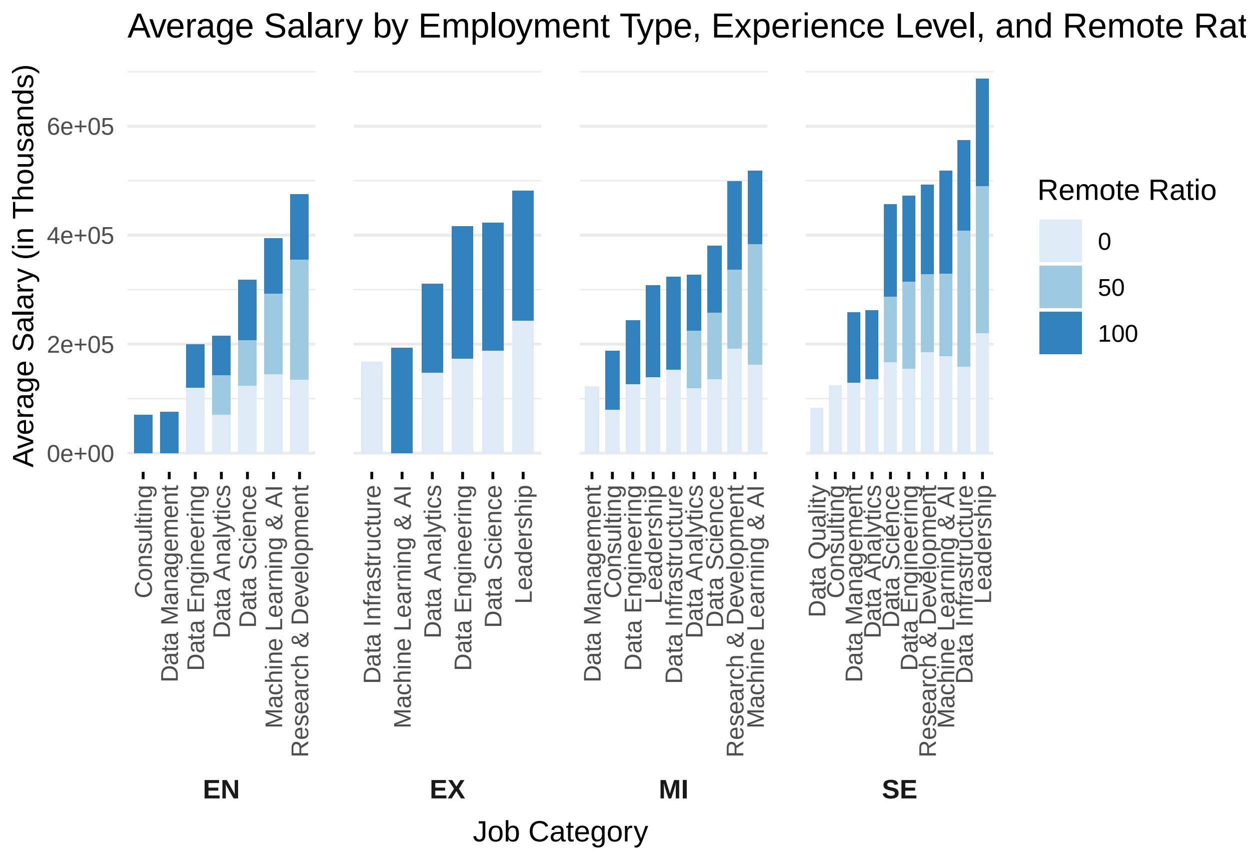
```
# Load necessary libraries
library(dplyr)
library(ggplot2)
library(tidytext)

# Prepare the data: replace NA, calculate total salary for ordering, and filter out zero heights
avg_salary_data <- us_data %>%
  mutate(
    experience_level = ifelse(is.na(experience_level), "Other", experience_level),
    employment_type = ifelse(is.na(employment_type), "Other", employment_type)
  ) %>%
  group_by(employment_type, experience_level, remote_ratio) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE), .groups = "drop") %>%
  ungroup() %>%
  group_by(employment_type, experience_level) %>%
  mutate(order_salary = sum(avg_salary)) %>%  # Calculate total salary for ordering
  ungroup() %>%
  arrange(desc(order_salary), .by_group = TRUE) %>%
  filter(avg_salary > 0)  # Filter out zero-height bars

# Plot the bar chart with employment_type reordered by total salary within each experience_level
ggplot(avg_salary_data, aes(x = reorder_within(employment_type, order_salary, experience_level), y = av
  geom_bar(stat = "identity", position = "stack", aes(group = experience_level), width = 0.7) +
  facet_grid(~ experience_level, scales = "free_x", switch = "x") +  # Free x-scale for each facet to m
  labs(
    title = "Average Salary by Employment Type, Experience Level, and Remote Ratio",
    x = "Employment Type",
    y = "Average Salary (in Thousands)",
    fill = "Remote Ratio"
  ) +
  scale_fill_brewer(palette = "Blues") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5),  # Display x-axis text vertically
    strip.placement = "outside",
    strip.text.x = element_text(size = 10, face = "bold"),
    panel.grid.major.x = element_blank(),
    panel.spacing = unit(1, "lines"),
    axis.ticks.x = element_line(size = 0.5)  # Add tick marks for each experience level
  ) +
  scale_x_reordered()  # Ensure the reordered levels within facets are a
```

```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



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

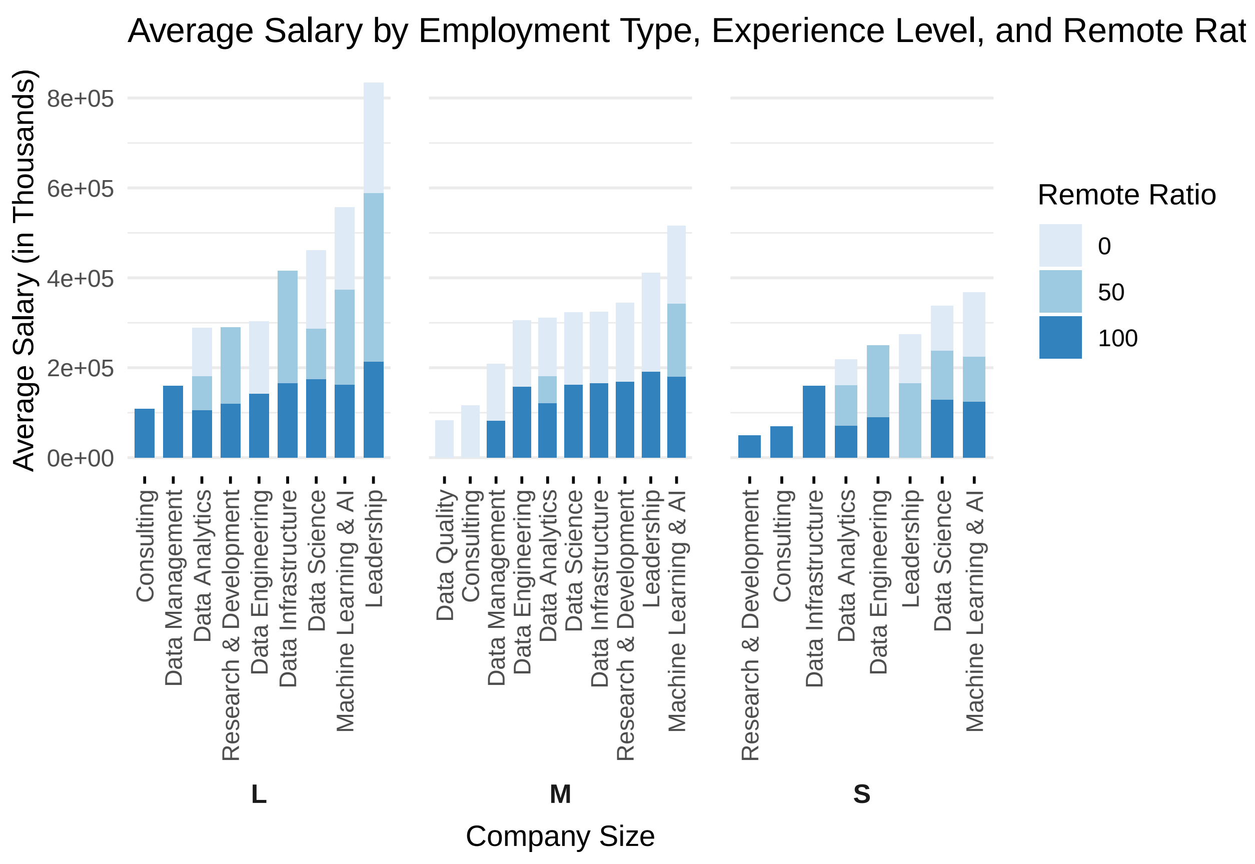
```r
library(ggplot2)
library(tidytext)

# Replace NA in job_category and experience_level with "Other"
# Calculate average salary across remote_ratio and use it for ordering
avg_salary_data <- us_data %>%
  mutate(
    experience_level = ifelse(is.na(experience_level), "Other", experience_level),
    job_category = ifelse(is.na(job_category), "Other", job_category)
  ) %>%
  group_by(job_category, experience_level, remote_ratio) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE), .groups = "drop") %>%
  ungroup() %>%
  group_by(job_category, experience_level) %>%
  mutate(order_salary = sum(avg_salary)) %>%  # New column for ordering within each experience level
  ungroup() %>%
  arrange(desc(order_salary), .by_group = TRUE) %>%
  filter(avg_salary > 0)  # Filter out zero-height bars

# Plot the bar chart with job_category reordered by avg_salary within each experience_level
ggplot(avg_salary_data, aes(x = reorder_within(job_category, order_salary, experience_level), y = avg_sa
  geom_bar(stat = "identity", position = "stack", aes(group = experience_level), width = 0.7) +
  facet_grid(~ experience_level, scales = "free_x", switch = "x") +  # Free x-scale for each facet to m
  labs(
    title = "Average Salary by Employment Type, Experience Level, and Remote Ratio",
    x = "Job Category",
    y = "Average Salary (in Thousands)",
    fill = "Remote Ratio"
  ) +
  scale_fill_brewer(palette = "Blues") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5),  # Display x-axis text vertically
    strip.placement = "outside",
    strip.text.x = element_text(size = 10, face = "bold"),
    panel.grid.major.x = element_blank(),
    panel.spacing = unit(1, "lines"),
    axis.ticks.x = element_line(size = 0.5)  # Add tick marks for each experience level
  ) +
  scale_x_reordered()  # Ensure the reordered levels within facets are applied correctly
```



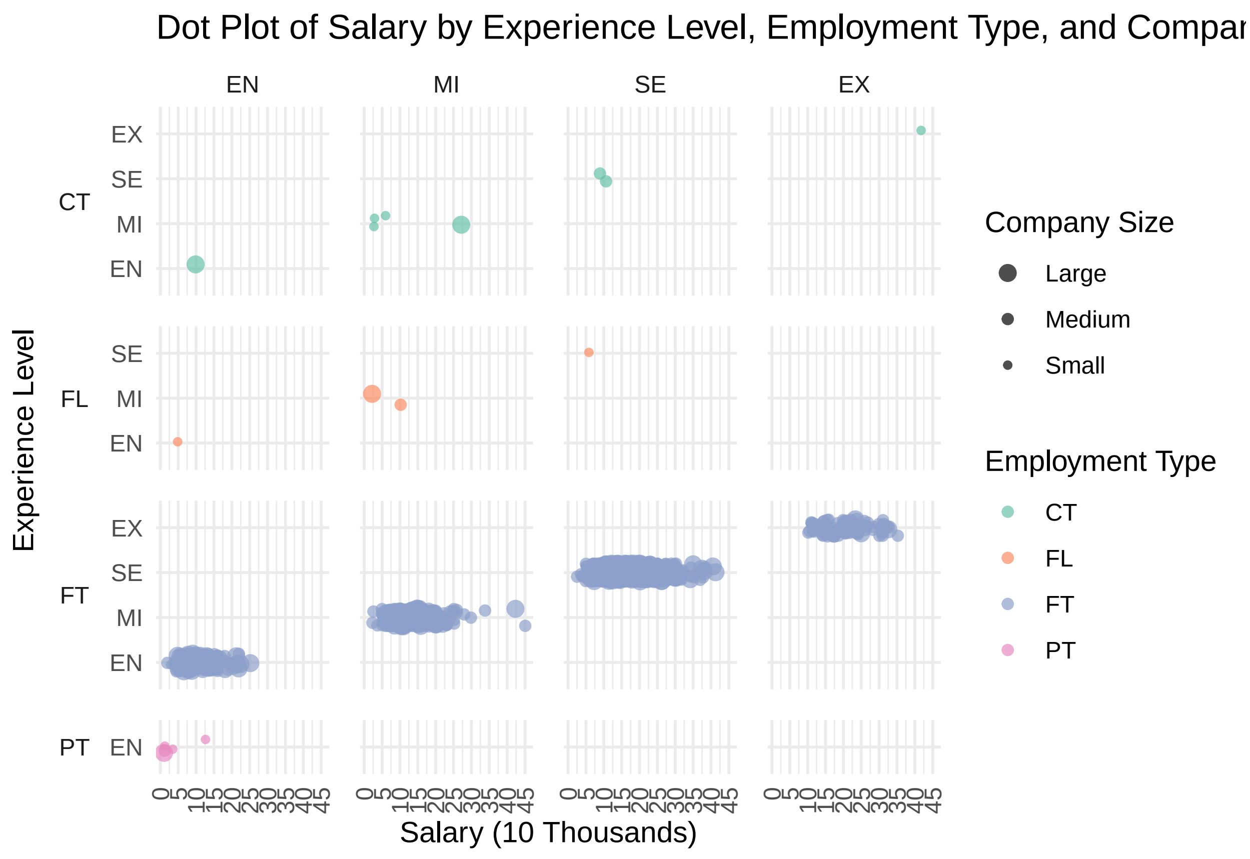
```r
avg_salary_data
```

```
## # A tibble: 71 x 5
##    job_category          experience_level remote_ratio avg_salary order_salary
##    <chr>                 <chr>                   <dbl>      <dbl>        <dbl>
##  1 Leadership            SE                          0    220754.      687478.
##  2 Leadership            SE                         50    270000       687478.
##  3 Leadership            SE                        100    196724.      687478.
##  4 Data Infrastructure   SE                          0    158615.      573972.
##  5 Data Infrastructure   SE                         50    250000       573972.
##  6 Data Infrastructure   SE                        100    165357.      573972.
##  7 Machine Learning & AI SE                          0    178164.      518734.
##  8 Machine Learning & AI SE                         50    151667.      518734.
##  9 Machine Learning & AI SE                        100    188903.      518734.
## 10 Machine Learning & AI MI                          0    162641.      518076.
## # i 61 more rows
```

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

# Replace NA in job_category and experience_level with "Other"
# Calculate average salary across remote_ratio and use it for ordering
avg_salary_data <- us_data %>%
  mutate(
    company_size = ifelse(is.na(company_size), "Other", company_size),
    job_category = ifelse(is.na(job_category), "Other", job_category)
  ) %>%
  group_by(job_category, company_size, remote_ratio) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE), .groups = "drop") %>%
  ungroup() %>%
  # Calculate the total (cumulative) salary for ordering within each company_size and job_category
  group_by(company_size, job_category) %>%
  mutate(order_salary = sum(avg_salary)) %>%  # New column for ordering within each company size
  ungroup() %>%
  arrange(desc(order_salary), company_size) %>%  # Sort by cumulative salary within each company size
  filter(avg_salary > 0)  # Filter out zero-height bars

# Plot the bar chart
ggplot(avg_salary_data, aes(x = reorder_within(job_category, order_salary, company_size), y = avg_salar
  geom_bar(stat = "identity", position = "stack", width = 0.7) +
  facet_grid(~ company_size, scales = "free_x", switch = "x") +  # Facet for company size on x-axis wit
  labs(
    title = "Average Salary by Employment Type, Experience Level, and Remote Ratio",
    x = "Company Size",
    y = "Average Salary (in Thousands)",
    fill = "Remote Ratio"
  ) +
  scale_fill_brewer(palette = "Blues") +
  scale_x_reordered() +  # Adjust x-axis ordering within each facet
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5),  # Display x-axis text vertically
    strip.placement = "outside",
    strip.text.x = element_text(size = 10, face = "bold"),
    panel.grid.major.x = element_blank(),
    panel.spacing = unit(1, "lines"),
    axis.ticks.x = element_line(size = 0.5)  # Add tick marks for each experience level
  )
```



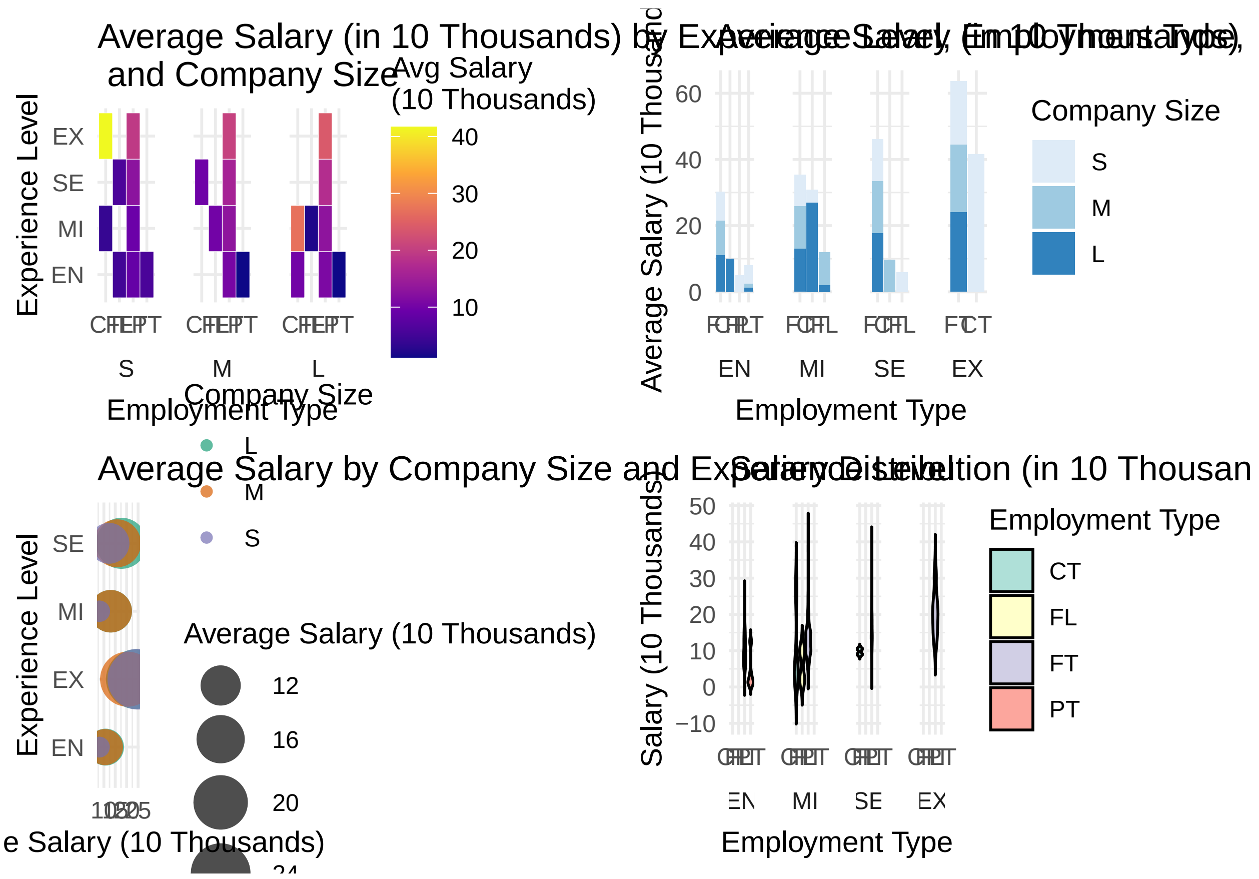
```
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Prepare data: Convert company size to a label format and order experience levels
dot_plot_data <- us_data %>%
  mutate(
    company_size_label = case_when(
      company_size == "L" ~ "Large",
      company_size == "M" ~ "Medium",
      company_size == "S" ~ "Small",
      TRUE ~ as.character(company_size)
    ),
    # Set experience level order
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  )

# Create the dot plot with jitter and reduced x-axis tick marks
ggplot(dot_plot_data, aes(x = salary / 10000, y = experience_level)) +
  geom_jitter(aes(size = company_size_label, color = employment_type),
              alpha = 0.7,
              width = 0.2,  # Adjust jitter width
              height = 0.2) +  # Adjust jitter height
  scale_size_manual(
    values = c("Large" = 2.5, "Medium" = 1.5, "Small" = 1),  # Proportionately smaller dot sizes for L,
    name = "Company Size"
  ) +
  facet_grid(employment_type ~ experience_level, scales = "free_y", space = "free_y", switch = "y") +
  labs(
    title = "Dot Plot of Salary by Experience Level, Employment Type, and Company Size",
    x = "Salary (10 Thousands)",
    y = "Experience Level"
  ) +
  scale_x_continuous(
    breaks = seq(0, max(dot_plot_data$salary) / 10000, by = 5),  # Set x-axis ticks at every 5 units
    labels = scales::comma_format()
  ) +
  scale_color_brewer(palette = "Set2", name = "Employment Type") +
  theme_minimal() +
  theme(
    strip.text.y.left = element_text(angle = 0),  # Position secondary y-axis labels
    strip.placement = "outside",  # Place secondary y-axis labels outside plot area
    panel.spacing = unit(0.8, "lines"),
    legend.position = "right",
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)  # Rotate x-axis tick labels vertica
  )
```



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

```r
# Prepare data for heatmap
heatmap_data <- us_data %>%
  mutate(
    company_size = factor(company_size, levels = c("S", "M", "L")),
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  ) %>%
  group_by(experience_level, employment_type, company_size) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE) / 10000, .groups = "drop")  # Divide by 10,000

# Plot heatmap
heatmap_plot <- ggplot(heatmap_data, aes(x = employment_type, y = experience_level, fill = avg_salary))
  geom_tile(color = "white") +
  facet_wrap(~ company_size, ncol = 3, strip.position = "bottom") +
  scale_fill_viridis_c(option = "plasma", name = "Avg Salary\n(10 Thousands)") +
  labs(
    title = "Average Salary (in 10 Thousands) by Experience Level, Employment Type,\n and Company Size"
    x = "Employment Type",
    y = "Experience Level"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    strip.placement = "outside",
    panel.spacing = unit(1, "lines")
  )

# Prepare data for stacked bar plot with descending order within each experience level
stacked_data <- us_data %>%
  mutate(
    company_size = factor(company_size, levels = c("S", "M", "L")),
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  ) %>%
  group_by(experience_level, employment_type, company_size) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE) / 10000, .groups = "drop")  # Divide by 10,000 for

# Calculate total salary per employment type within each experience level for sorting
sorted_data <- stacked_data %>%
  group_by(experience_level, employment_type) %>%
  summarise(total_salary = sum(avg_salary), .groups = "drop") %>%
  arrange(experience_level, desc(total_salary)) %>%
  mutate(employment_type = factor(employment_type, levels = unique(employment_type))) # Order employmen

# Join sorted order back to stacked_data
stacked_data <- left_join(stacked_data, sorted_data, by = c("experience_level", "employment_type"))

# Plot stacked bar chart with reordered employment types
stacked_plot <- ggplot(stacked_data, aes(x = reorder(employment_type, -total_salary), y = avg_salary, f
  geom_bar(stat = "identity", position = "stack") +
  facet_wrap(~ experience_level, ncol = 4, scales = "free_x", strip.position = "bottom") +
  scale_fill_brewer(palette = "Blues") +
  labs(
    title = "Average Salary (in 10 Thousands) by Experience Level, Employment Type, and Company Size",
    x = "Employment Type",
    y = "Average Salary (10 Thousands)",
    fill = "Company Size"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    strip.placement = "outside",
    panel.spacing = unit(1, "lines")
  )

# Prepare data for bubble plot
bubble_data <- us_data %>%
  group_by(company_size, experience_level) %>%
  summarise(avg_salary = mean(salary, na.rm = TRUE), .groups = "drop") %>%
  mutate(avg_salary_scaled = avg_salary / 10000)  # Scale salary to tens of thousands for display

# Plot bubble chart
bubble_plot <- ggplot(bubble_data, aes(x = avg_salary_scaled, y = experience_level, size = avg_salary_sc
  geom_point(alpha = 0.7) +
  scale_size(range = c(3, 10)) +
  scale_color_brewer(palette = "Dark2") +
  labs(
    title = "Average Salary by Company Size and Experience Level",
    x = "Average Salary (10 Thousands)",
    y = "Experience Level",
    size = "Average Salary (10 Thousands)",
    color = "Company Size"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5)  # Keep x-axis labels horizontal
  )


# Prepare data for violin plot
violin_data <- us_data %>%
  mutate(
    experience_level = factor(experience_level, levels = c("EN", "MI", "SE", "EX"))
  )

# Plot violin chart
violin_plot <- ggplot(violin_data, aes(x = employment_type, y = salary / 10000, fill = employment_type)
  geom_violin(trim = FALSE, color = "black", alpha = 0.7) +
  facet_wrap(~ experience_level, ncol = 4, strip.position = "bottom") +
  scale_fill_brewer(palette = "Set3") +
  labs(
    title = "Salary Distribution (in 10 Thousands) by Experience Level and Employment Type",
    x = "Employment Type",
    y = "Salary (10 Thousands)",
    fill = "Employment Type"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    strip.placement = "outside",
    panel.spacing = unit(1, "lines")
  )

# Arrange the four plots in a 2x2 grid
grid.arrange(heatmap_plot, stacked_plot, bubble_plot, violin_plot, ncol = 2)
```

```
## Warning: Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
## Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
```


Average Salary (in 10 Thousands) by Experience Level, Employment Type, and Company Size
Avg Salary (10 Thousands)
Experience Level
EX
SE
MI
EN
CTFLFTPT
S
M
L
Company Size
Employment Type
Average Salary (in 10 Thousands)
Average Salary (10 Thousands)
FT CT FL PT
EN
MI
SE
EX
Company Size
S
M
L
Average Salary by Company Size and Experience Level
Company Size
L
M
S
Average Salary (10 Thousands)
Salary Distribution (in 10 Thousands)
Salary (10 Thousands)
Employment Type
CT
FL
FT
PT