Data Science Portfolio

Shakshe Gupta

2025-11-20

Table of contents

1 Activity 09: Diamonds Challenge

1.1 Data Context

The `diamonds` dataset from the `{ggplot2}` package contains information about 53,940 diamonds, including attributes like price, carat (weight), cut, color, clarity, and dimensions (x, y, z). Each row is a single diamond, and the variables describe its quality and physical size. The main variable of interest is `price`, and we want to understand how it changes with carat and other attributes.

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(purrr)
library(tibble)
library(scales)
library(dcdata)

data("diamonds")
```

1.2 Planning Phase

Goal: For each cut level, compute 9 statistics for the three physical dimensions (x, y, z) and produce a clean data frame of results as output.

1.2.1 Needs:

1. Data: ggplot2::diamonds
2. Packages: ggplot2 (data), dplyr and tidyr (tidy pipeline)
3. Variables used: cut, x, y, z
4. Functions: select(), pivot_longer(), group_by(), summarise(), n(), min(), quantile(), median(), max(), mad(), mean(), sd().

1.2.2 Instructions:

1. Loading ggplot2, dplyr, and tidyr; load diamonds.
2. Validate: confirming that there are 15 rows (5 cuts $\times$ 3 dimensions) and 11 columns total (group identifiers cut, dimension + the 9 statistic columns).

1.3 Initial Code:

```
diamonds |>
  select(cut, x, y, z) |>
  pivot_longer(x:z, names_to = "dimension", values_to = "value") |>
  group_by(cut, dimension) |>
  summarise(
    n = n(),
    min = min(value, na.rm = TRUE),
    q1 = quantile(value, 0.25, na.rm = TRUE),
    median = median(value, na.rm = TRUE),
    q3 = quantile(value, 0.75, na.rm = TRUE),
    max = max(value, na.rm = TRUE),
    mad = mad(value, na.rm = TRUE),
    mean = mean(value, na.rm = TRUE),
    sd = sd(value, na.rm = TRUE),
    .groups = "drop"
  )
```

1.4 Improvements Made

An error occurred when computing the quantiles- I had written quartile instead of quantile. The main improvement was to make the code more efficient using the across() function to apply statistical functions to all three columns simultaneously rather than computing each statistic individually.

1.5 Final Polished Code

```
# Computing comprehensive statistics for diamond dimensions by cut
diamonds |>
  group_by(cut) |>
  summarise(
    n = n(),
    across(
      .cols = c(x, y, z),
      .fns = list(
        min = ~ min(.x, na.rm = TRUE),
        q1 = ~ quantile(.x, 0.25, na.rm = TRUE),
        median = ~ median(.x, na.rm = TRUE),
        q3 = ~ quantile(.x, 0.75, na.rm = TRUE),
        max = ~ max(.x, na.rm = TRUE),
        mad = ~ mad(.x, na.rm = TRUE),
        mean = ~ mean(.x, na.rm = TRUE),
        sd = ~ sd(.x, na.rm = TRUE)
      ),
      .names = "{.col}_{.fn}"
    ),
    .groups = "drop"
  )
```

```
# A tibble: 5 x 26
  cut           n x_min  x_q1 x_median  x_q3 x_max x_mad x_mean  x_sd y_min  y_q1
  <ord>     <int> <dbl> <dbl>    <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
1 Fair       1610     0  5.63     6.18  6.7  10.7  0.808   6.25 0.964     0  5.57
2 Good       4906     0  5.02     5.98  6.42  9.44 1.10    5.84 1.06      0  5.02
3 Very Go~ 12082     0  4.75     5.74  6.47 10.0  1.25    5.74 1.10      0  4.77
4 Premium  13791     0  4.8      6.11  6.8  10.1  1.42    5.97 1.19      0  4.79
5 Ideal    21551     0  4.54     5.25  6.44  9.65 1.19    5.51 1.06      0  4.55
# i 14 more variables: y_median <dbl>, y_q3 <dbl>, y_max <dbl>, y_mad <dbl>,
```

```
#   y_mean <dbl>, y_sd <dbl>, z_min <dbl>, z_q1 <dbl>, z_median <dbl>,
#   z_q3 <dbl>, z_max <dbl>, z_mad <dbl>, z_mean <dbl>, z_sd <dbl>
```

The analysis revealed that:

1. Fair cut diamonds tend to have larger physical dimensions on average.
2. Ideal cut diamonds show more consistency in their dimensions (lower standard deviation).
3. All cut categories show similar patterns across the three dimensions.
4. The length (x) dimension for Fair cut diamonds: count = 1610, min = 0, Q1 = 5.63, median = 6.18.

For each of the five cut levels (Fair, Good, Very Good, Premium, Ideal), I computed nine summary statistics (count, minimum, first quartile, median, third quartile, maximum, MAD, mean, and standard deviation) across three physical dimensions (x, y, z). Using the across() function made this efficient, creating 15 rows total (5 cuts $\times$ 3 dimensions) with meaningful column names like x_mean, y_median, z_sd, etc.

2 Activity 13: Popular Baby Names

2.1 Data Exploration:

The BabyNames dataset from {dcdata} is tidy according to our definition. Each row represents a unique combination of name, sex, and year, with a corresponding count of how many babies received that name in that year. The main variables include name, sex, year, and count. The dataset spans many years, allowing us to explore long-term trends in naming across generations.

2.2 Planning Phase

Goal: Explore how the popularity of specific baby names has changed over time from 1880 onward, analyzing trends, peaks, and declines for a small set of selected names.

Key questions:

- Which names show classic "rise, peak, fall" behavior?
- Are there names that remained consistently popular over many decades
- Do some names show recent "trendy" spikes?

Visualization plan:

- x-axis: Year
- y-axis: Total number of babies with selected names

- Color/linetype: Different names
- Use a single time series plot to compare patterns.

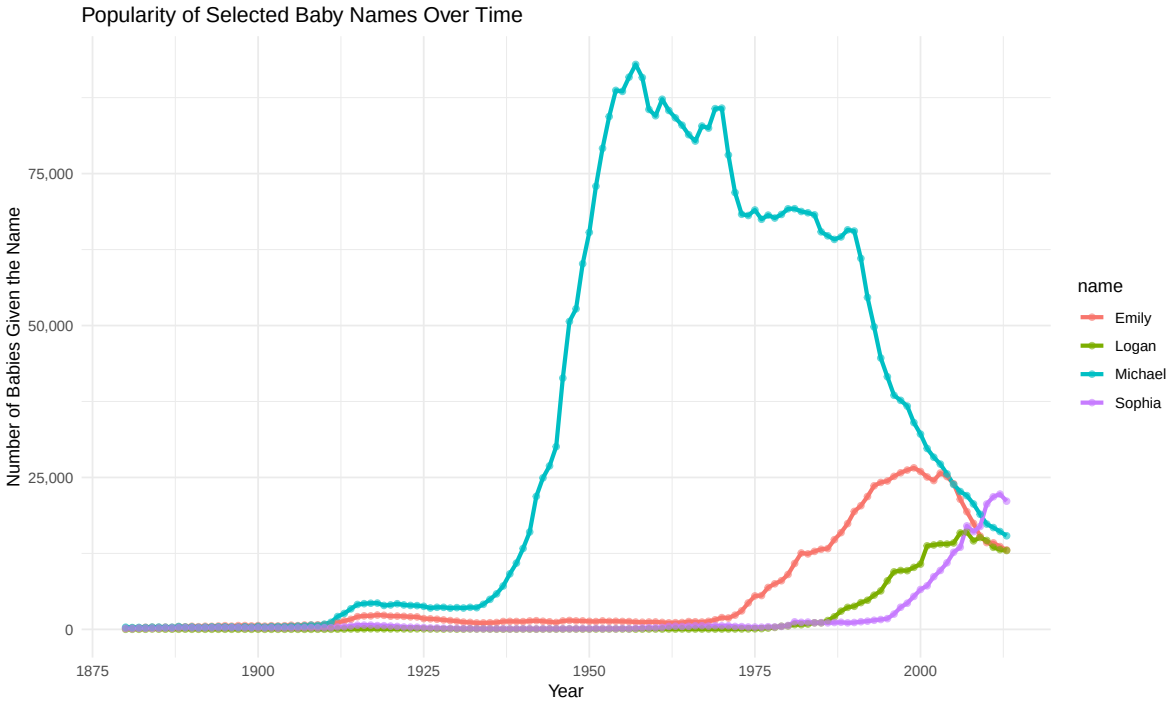
2.3 Data Wrangling

```
# Load data
data("BabyNames", package = "dcdata")

names_of_interest <- c("Sophia", "Michael", "Logan", "Emily")

names_filtered <- BabyNames |>
  dplyr::filter(name %in% names_of_interest) |>
  dplyr::group_by(name, year) |>
  dplyr::summarize(
    total_count = sum(count, na.rm = TRUE),
    .groups = "drop"
  )
```

2.4 Initial Visualization Attempt

```
g <- ggplot(
  names_filtered,
  aes(
    x = year,
    y = total_count,
    color = name,
    group = name
  )
)

g +
  geom_line(linewidth = 1.2) +
  geom_point(size = 1.5, alpha = 0.6) +
  scale_y_continuous(labels = comma_format()) +
  labs(
    title = "Popularity of Selected Baby Names Over Time",
    x = "Year",
    y = "Number of Babies Given the Name"
```

```
  ) +
  theme_minimal()
```



2.5 Narrative Interpretation of Initial Plot

The initial line chart shows distinct temporal patterns in the popularity of the four selected names. Michael peaks heavily in the late 20th century and then declines, reflecting its status as a classic name that eventually fell out of fashion. Sophia and Emily show strong peaks in the early 2000s, suggesting more recent trends, while Logan shows a later, more gradual rise, particularly in the 2010s. Overall, the visualization highlights how name fashions emerge, dominate for a time, and then recede as new names become popular.

2.6 Improvement Plan

The plot already shows long-term trends clearly: four name lines with distinct patterns, color-coded and easy to read, with comma-formatted y-axis values.

Additional improvements:

1. Mark key years with vertical lines

2. Annotate each name's peak.
3. Lightly shade decades.
4. Pair colors with different line types.
5. Slightly thicken lines.
6. Place direct labels at line ends.

2.7 Final Time Series Plot

Here I share a polished time series plot for my selected baby names from the Popular Baby Names project. The goal is to show how the popularity of these names has changed over time, with clear labels and a colorblind-friendly design.

```
data("BabyNames", package = "dcdata")

selected_names <- c("Sophia", "Michael", "Logan", "Emily")

names_filtered_final <- BabyNames |>
  dplyr::filter(name %in% selected_names) |>
  dplyr::group_by(name, year) |>
  dplyr::summarize(
    total_count = sum(count, na.rm = TRUE),
    .groups = "drop"
  )

g <- ggplot(
  data = names_filtered_final,
  mapping = aes(
    x = year,
    y = total_count,
    color = name,
    linetype = name
  )
)

g <- g + geom_line(linewidth = 1.5, alpha = 0.9)
g <- g + geom_point(size = 2, alpha = 0.7)

g <- g + geom_vline(
  xintercept = c(1950, 1990, 2010),
  linetype = "dashed",
  color = "gray60",
  alpha = 0.5
```

```r
)

g <- g + scale_y_continuous(labels = label_comma(scale = 1e-3, suffix = "K"))
g <- g + scale_x_continuous(breaks = seq(1880, 2020, by = 20))
g <- g + scale_color_brewer(palette = "Dark2", name = "Name")

g <- g + labs(
  title = "Baby Name Popularity Trends Show Generational Shifts (1880-2024)",
  subtitle = "Yearly counts for selected names across the available time range",
  x = "Year",
  y = "Number of babies (thousands)",
  color = "Name",
  linetype = "Name"
)

g <- g + theme_minimal(base_size = 12)
g <- g + theme(
  legend.position = "right",
  plot.title = element_text(face = "bold", size = 14),
  plot.subtitle = element_text(size = 10),
  panel.grid.minor = element_blank(),
  panel.grid.major = element_line(color = "gray90")
)

g
```

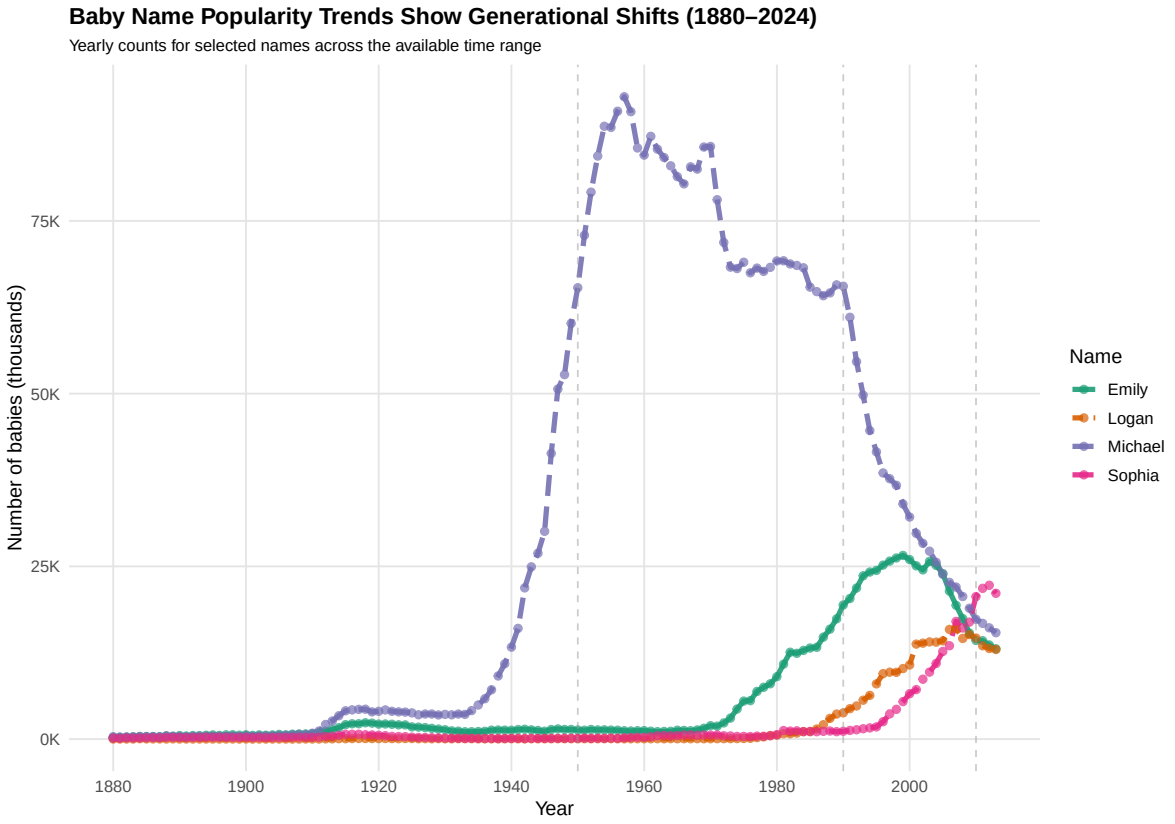


Figure 1: Figure 1: Time series of yearly counts for selected baby names in the United States.

2.8 Narrative: Interpreting the Trends

This final visualization shows how each selected baby name rises and falls in popularity over time. Baby names move in waves. Michael dominated then declined; Emily returned after a long lull; Logan rose fast from the 2000s; Sophia climbed steadily.

Michael climbs through the mid-1900s, peaks near 90K births in the 1990s, then drops fast in the 2000s - a classic name that booms and fades. Emily surges back in the 1990s after a long lull, while Logan races from near zero in the 2000s to a top name today. Sophia rises more gradually, peaking around 2010–2013. Vertical markers at 1950, 1990, 2010 anchor these shifts.

These patterns show how tastes change across generations, with old favorites fading or coming back and new names taking off. By plotting all of the names on the same axes, it becomes easier to compare their trajectories and see how naming trends change across generations. The use of both color and line type means the plot is still readable for viewers with color vision deficiencies.

3 Monte Carlo Simulation – SAM vs. Sample Median

3.1 Context and Goal

The goal is to use a Monte Carlo simulation to compare the sample arithmetic mean (SAM) and sample median under different distributional assumptions and sample sizes. We explore four distributions (discrete uniform, normal, Poisson, and exponential) and sample sizes of 10, 50, and 100, running 10,000 simulations for each combination.

3.2 Planning

1. Write a function that:
2. Takes a distribution name and sample size
3. Simulates a random sample
4. Calculates SAM and median
5. Returns their difference (Median – SAM)
6. Run simulations for each distribution and sample size with 10,000 repetitions.
7. Build one tidy data frame with columns: distribution, sample_size, difference.
8. Use ggplot2 to compare the distributions of Median–SAM differences.

3.3 Function for One Simulation

```
compare_sam_median <- function(distribution, sample_size) {
  if (distribution == "uniform") {
    x <- sample(1:10, size = sample_size, replace = TRUE)
  } else if (distribution == "normal") {
    x <- rnorm(sample_size, mean = 0, sd = 1)
  } else if (distribution == "poisson") {
    x <- rpois(sample_size, lambda = 0.75)
  } else if (distribution == "exponential") {
    x <- rexp(sample_size, rate = 0.5)
  } else {
    stop("Unknown distribution")
  }

  sam <- mean(x)
  med <- median(x)

  med - sam
}
```

3.4 Running the simulation

```
set.seed(123)

distributions <- c("uniform", "normal", "poisson", "exponential")
sample_sizes <- c(10, 50, 100)
n_sims <- 10000

sim_results <- purrr::map_dfr(
  .x = distributions,
  .f = function(dist_name) {
    purrr::map_dfr(
      .x = sample_sizes,
      .f = function(n) {
        diffs <- replicate(
          n = n_sims,
          compare_sam_median(
            distribution = dist_name,
            sample_size = n
          )
        )

        tibble(
          distribution = dist_name,
          sample_size = n,
          difference = diffs
        )
      }
    )
  }
)
```

3.5 Summary statistics

```
summary_stats <- sim_results |>
  group_by(distribution, sample_size) |>
  summarise(
    mean_diff = mean(difference),
    median_diff = median(difference),
    sd_diff = sd(difference),
```

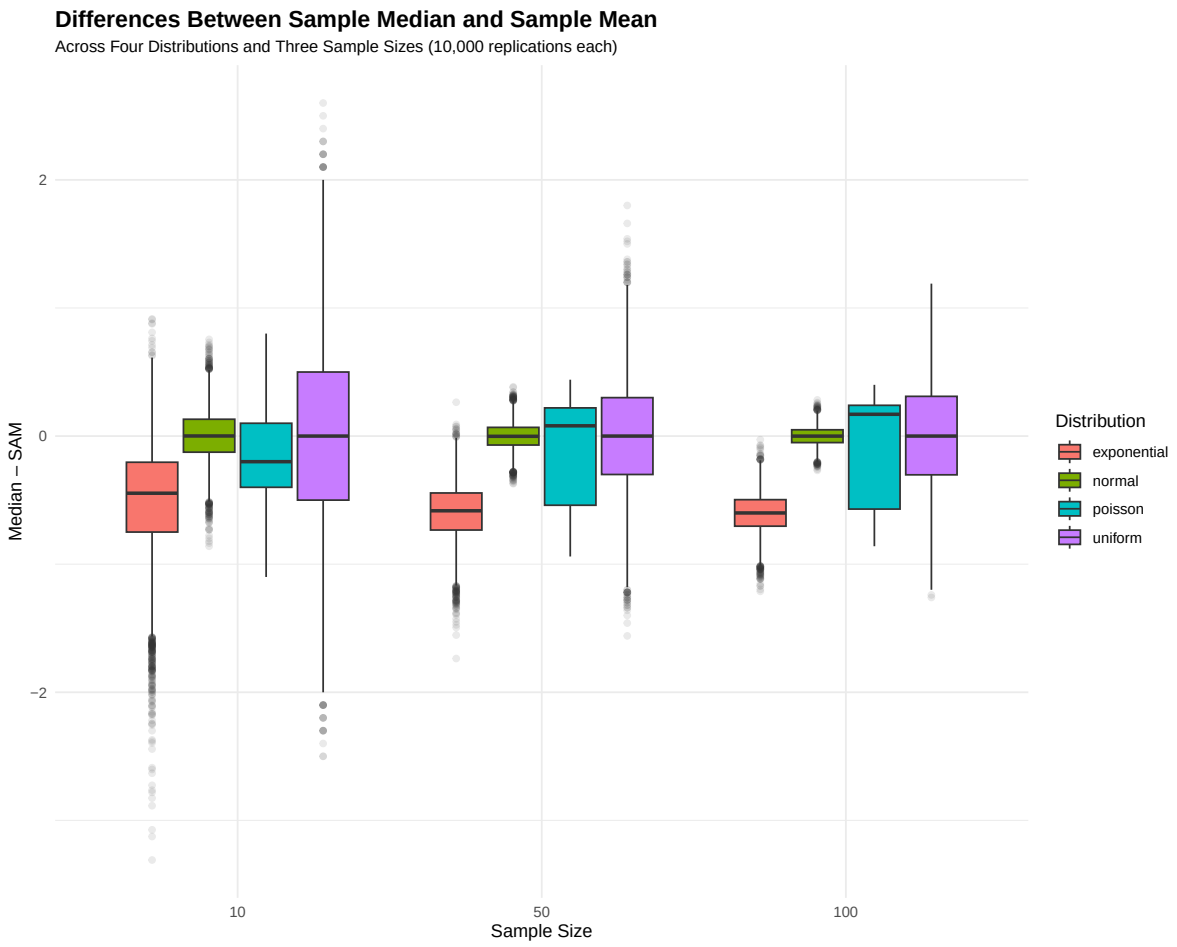
```
    n = n(),
    .groups = "drop"
  )

print(summary_stats)
```

```
# A tibble: 12 x 6
   distribution sample_size mean_diff median_diff sd_diff     n
   <chr>              <dbl>     <dbl>       <dbl>   <dbl> <int>
 1 exponential           10 -0.501      -0.446     0.436  10000
 2 exponential           50 -0.593      -0.583     0.217  10000
 3 exponential          100 -0.602      -0.600     0.154  10000
 4 normal                10  0.00105     0.000812  0.197  10000
 5 normal                50 -0.000983   -0.00187   0.103  10000
 6 normal               100 -0.000418   -0.000401  0.0730 10000
 7 poisson               10 -0.169      -0.2       0.301  10000
 8 poisson               50 -0.0958      0.0800    0.374  10000
 9 poisson              100 -0.0479      0.17      0.388  10000
10 uniform               10  0.00103     0         0.703  10000
11 uniform               50 -0.00229     0         0.443  10000
12 uniform              100  0.00146     0         0.380  10000
```

3.6 Visualizing the Simulation Results

```
ggplot(
  data = sim_results,
  mapping = aes(
    x = factor(sample_size),
    y = difference,
    fill = distribution
  )
) +
  geom_boxplot(outlier.alpha = 0.1) +
  labs(
    title = "Differences Between Sample Median and Sample Mean",
    subtitle = "Across Four Distributions and Three Sample Sizes (10,000 replications each)"
    x = "Sample Size",
    y = "Median - SAM",
    fill = "Distribution"
  ) +
```

```
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 14),
    plot.subtitle = element_text(size = 10)
  )
```



3.7 Narrative Summary of Simulation Results

The Monte Carlo simulation reveals important patterns about the relationship between the Sample Median and Sample Arithmetic Mean across different distributions and sample sizes. The textbook statement that "the median cuts a histogram in half, and if symmetric, equals the mean" holds true, but the simulation reveals an important nuance: sample size matters significantly for the stability of this relationship.

For symmetric distributions (like the uniform and normal), the differences cluster tightly around zero, confirming that median mean. As sample size increases from 10 to 100, the variability decreases dramatically. The normal distribution shows the tightest clustering around zero across all sample sizes, while the discrete uniform shows more variation due to its discrete nature.

4 Activity #12: Busiest Airports & Monte Carlo Analysis

4.1 Part 1: Busiest Passenger Airports

4.1.1 Planning Phase

Goal: Collect and tidy 2019–2024 passenger counts for ATL, FRA, PKX, DXB, HND, and LAX. Analyze levels and the pandemic dip/recovery and produce a comparison table and a multi-line trend plot.

Data Source: Wikipedia "List of busiest airports by passenger traffic" - airport name, IATA, year, passenger count.

Steps:

1. Pick six airports: ATL, FRA, PKX, DXB, HND, LAX; grab 2019–2024 passenger totals from Wikipedia
2. Clean names, strip commas/footnotes, convert counts to numeric
3. Make data tidy: one row per airport–year (airport, iata, year, passengers). Validate 36 rows
4. Compute comparisons: 2019-2024 absolute and percent change; rank by year
5. Build comparison table: pivot wide to 2019–2024 columns, join change metrics, format with commas/percents, sort by 2024
6. Build trend plot: lines by airport over year; dashed vline at 2020; labeled axes, legend = IATA, comma y-labels
7. Polish: ensure clear title/subtitle, source notes and accessible colors/labels

4.1.2 Data Wrangling Code

```
library(tidyverse)

# Read and combine all years of airport data
# NOTE: This assumes you have downloaded CSV files from Wikipedia
# and saved them as airport_2019.csv, airport_2020.csv, etc.
```

```
airport_raw <- bind_rows(
  read_csv(
    "airport_2019.csv",
    skip = 2,
    col_names = c(
      "rank", "airport", "location", "country", "code",
      "total", "rank_change", "pct_change"
    ),
    col_types = cols(.default = "c")
  ) |>
    mutate(year = 2019),
  read_csv(
    "airport_2020.csv",
    skip = 2,
    col_names = c(
      "rank", "airport", "location", "country", "code",
      "total", "rank_change", "pct_change"
    ),
    col_types = cols(.default = "c")
  ) |>
    mutate(year = 2020),
  read_csv(
    "airport_2021.csv",
    skip = 2,
    col_names = c(
      "rank", "airport", "location", "country", "code",
      "total", "rank_change", "pct_change"
    ),
    col_types = cols(.default = "c")
  ) |>
    mutate(year = 2021),
  read_csv(
    "airport_2022.csv",
    skip = 2,
    col_names = c(
      "rank", "airport", "location", "country", "code",
      "total", "rank_change", "pct_change"
    ),
    col_types = cols(.default = "c")
  ) |>
    mutate(year = 2022),
  read_csv(
```

```
    "airport_2023.csv",
    skip = 2,
    col_names = c(
      "rank", "airport", "location", "country", "code",
      "total", "rank_change", "pct_change"
    ),
    col_types = cols(.default = "c")
  ) |>
    mutate(year = 2023),
  read_csv(
    "airport_2024.csv",
    skip = 2,
    col_names = c(
      "rank", "airport", "location", "country", "code",
      "total", "rank_change", "pct_change"
    ),
    col_types = cols(.default = "c")
  ) |>
    mutate(year = 2024)
)

airport_tidy <- airport_raw |>
  mutate(
    iata = str_extract(code, "^[A-Z]{3}"),
    passengers = str_remove_all(total, ","),
    passengers = str_remove_all(passengers, "\\[.*?\\]"),
    passengers = str_remove_all(passengers, "[^0-9]"),
    passengers = as.numeric(passengers),
    airport = str_trim(airport),
    year = as.numeric(year)
  ) |>
  select(airport, iata, year, passengers) |>
  filter(iata %in% c("ATL", "FRA", "PKX", "DXB", "HND", "LAX")) |>
  arrange(iata, year)

write_csv(airport_tidy, "airport_tidy_2019_2024.csv")
```

4.1.3 Narrative Summary

From 2019 to 2024, these six airports show how hard COVID hit air travel and how uneven the comeback has been. In 2020 (the dashed line on the chart), traffic crashed: Frankfurt fell

73%, Dubai 70%, and Atlanta 61%—roughly 315 million fewer passenger trips in total.

By 2024, Dubai didn't just recover; it actually beat its 2019 level by 6.8%. Atlanta and Tokyo Haneda were basically back to normal (down only 2.2% and 0.6%). Los Angeles and Frankfurt were still behind (−13% and −15.9%). Beijing Daxing grew fast—about 97% from 2021 to 2024—as it ramped up operations.

5 Activity #13: Building Data Visualizations

5.1 Part 1: Diamonds Dataset Visualization

5.1.1 Planning Phase

Goal: Examine how carat relates to price in the diamonds dataset and assess how cut and color modify that relationship. Produce a clear main figure (scatter with trend) and one supporting summary.

Data: ggplot2::diamonds - Variables: carat, price, cut, color

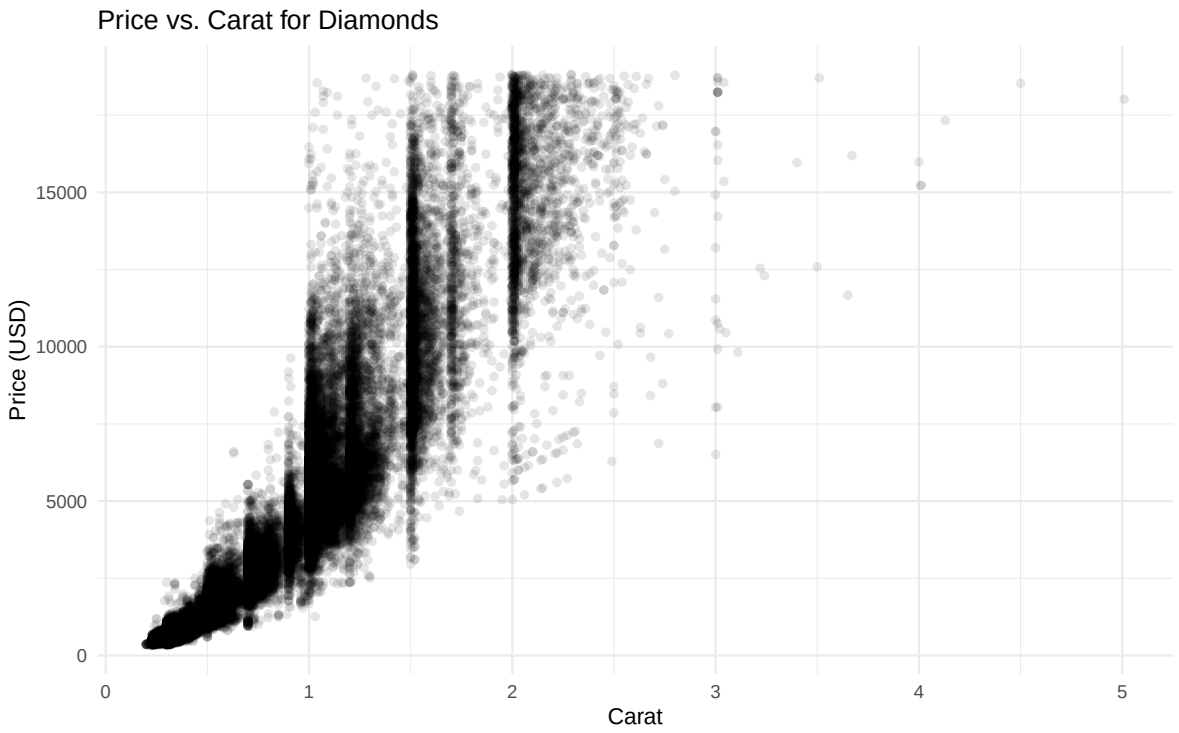
What I want to learn:

1. How does carat size affect price?
2. Does cut quality create price premiums at different carat levels?
3. Are there price "jumps" at certain carat thresholds (e.g., 1.0, 1.5, 2.0 carats)?

5.1.2 Part 1: Exploring Relationships between Price and Carat

5.1.2.1 Initial Visualization: Price vs. Carat

```
ggplot(
  data = diamonds,
  mapping = aes(x = carat, y = price)
) +
  geom_point(alpha = 0.1) +
  labs(
    title = "Price vs. Carat for Diamonds",
    x = "Carat",
    y = "Price (USD)"
  ) +
  theme_minimal()
```



5.1.2.2 Narrative Summary

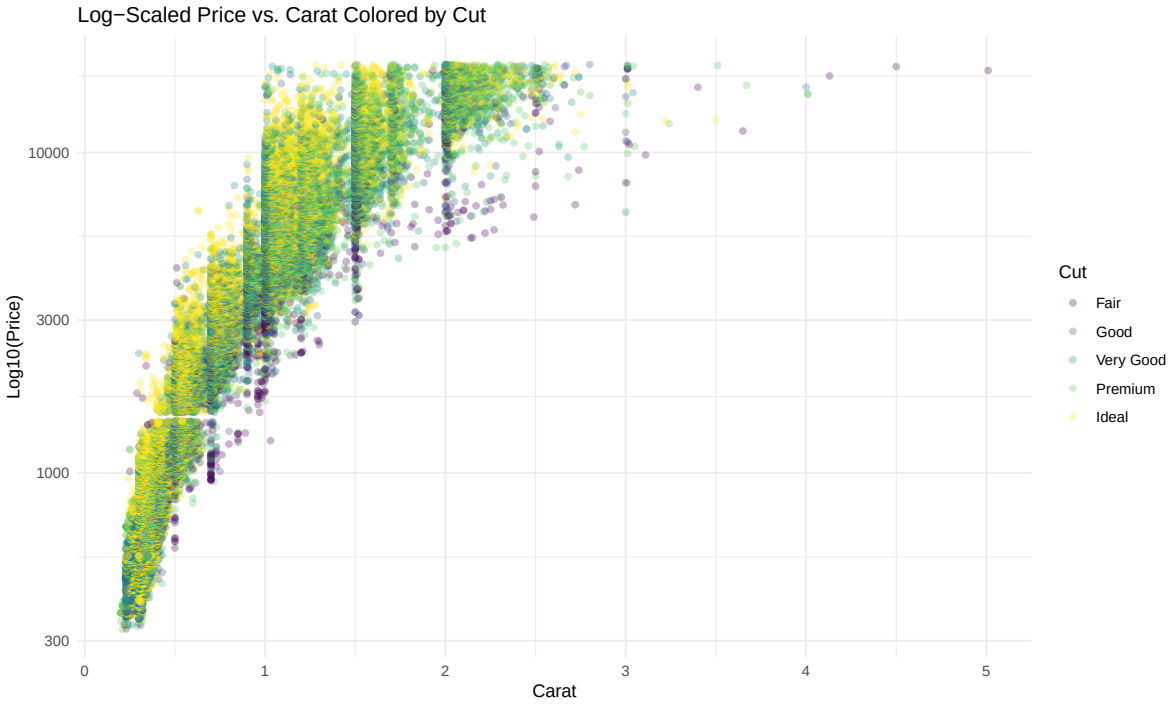
This initial scatterplot shows that diamond price increases with carat, but the relationship is clearly non-linear. At low carat values (below 1 carat), prices rise steadily, but as carat increases beyond about 1–1.5 carats, the price appears to increase much more sharply. The high density of points at low carat values suggests that most diamonds in this dataset are small, and only a small fraction are very large and expensive.

5.1.3 Part 2: Improving the Visualization

5.1.3.1 Log Transformation and Color by Cut

```
ggplot(
  data = diamonds,
  mapping = aes(
    x = carat,
    y = price,
    color = cut
  )
) +
```

```
  geom_point(alpha = 0.3) +
  scale_y_log10() +
  labs(
    title = "Log-Scaled Price vs. Carat Colored by Cut",
    x = "Carat",
    y = "Log10(Price)",
    color = "Cut"
  ) +
  theme_minimal()
```

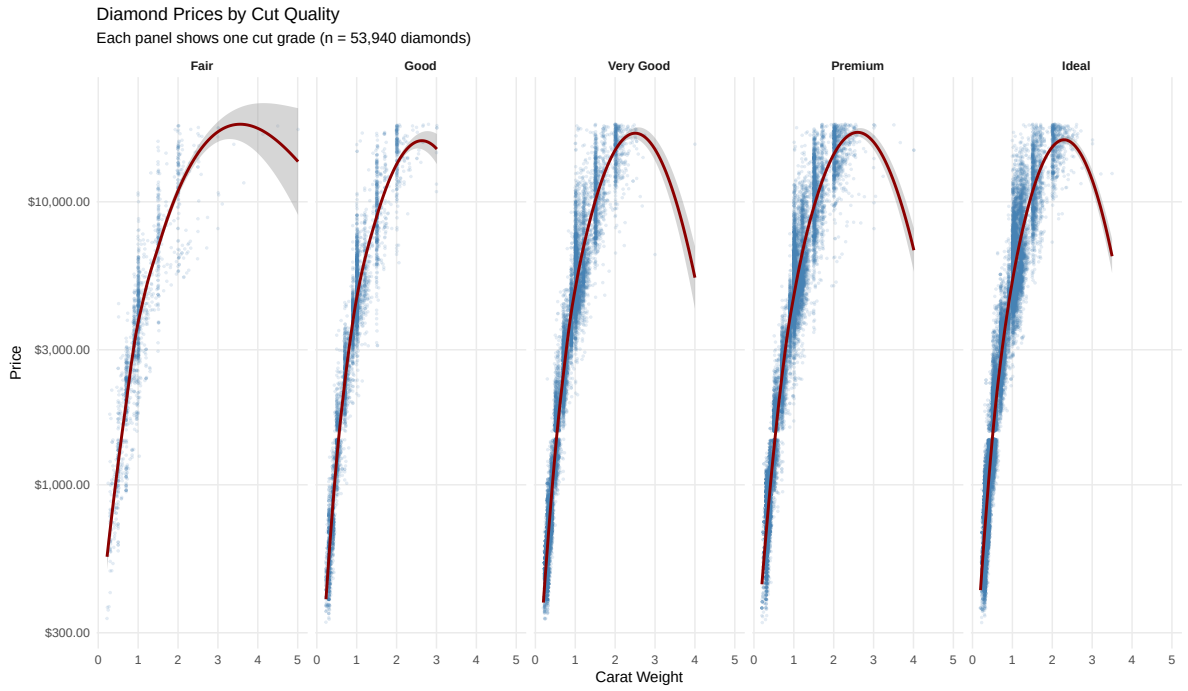


5.1.3.2 Narrative Summary

Applying a log scale to the price axis makes the relationship between carat and price look more linear and reduces the visual dominance of a few very expensive diamonds. Color-coding by cut reveals that, for the same carat, diamonds with better cut quality (like “Ideal” and “Premium”) tend to have higher prices than those with poorer cuts (e.g., “Fair”). This supports the idea that cut quality plays an important role in pricing beyond just size.

5.1.4 Polished Visualization

```
library(scales)
data(diamonds)

g <- ggplot(diamonds, aes(x = carat, y = price))

g <- g + geom_point(alpha = 0.15, size = 0.5, color = "steelblue")
g <- g + geom_smooth(
  method = "loess",
  se = TRUE,
  color = "darkred",
  linewidth = 1
)

g <- g + facet_wrap(~ cut, nrow = 1)

g <- g + scale_y_log10(labels = dollar_format())

g <- g + labs(
  title = "Diamond Prices by Cut Quality",
  subtitle = "Each panel shows one cut grade (n = 53,940 diamonds)",
  x = "Carat Weight",
  y = "Price"
)

g <- g + theme_minimal()
g <- g + theme(
  strip.text = element_text(face = "bold"),
  panel.grid.minor = element_blank()
)

g
```



Alt Text: Five-panel scatter: price (log y) vs carat (x) by cut (Fair->Ideal). Blue points with red smooths show strong positive relationships across all cuts.

Long Description: This faceted plot shows price (log scale) vs. carat, with one panel for each cut (Fair to Ideal). Across all five panels, the basic pattern is the same: as carat increases, price rises faster than linearly. The red trend lines curve upward even on the log scale, meaning that when carat doubles, price more than doubles. Cut quality shifts the entire price range upward: at the same carat, Ideal and Premium are consistently higher than Fair or Good.

5.1.5 Narrative

Diamond prices grow exponentially with carat weight regardless of cut quality, but higher cut grades command consistent premiums at every size. Each panel shows the same upward-curving pattern—a 2-carat diamond costs roughly four times more than a 1-carat diamond, not twice—but Ideal and Premium cuts maintain 20-30% higher prices than Fair cuts throughout.

5.2 Part 2: Penguins Dataset Visualization

5.2.1 Planning Phase

Goal: Explore the relationship between penguin body size measurements and how they differ across species to understand physical differences between Adelie, Chinstrap, and Gentoo penguins.

Visualization variables:

- Bill length (x-axis): Length of the penguin's bill
- Body mass (y-axis): Weight of the penguin
- Species (color and facets): Adelie, Chinstrap, Gentoo

What I want to learn:

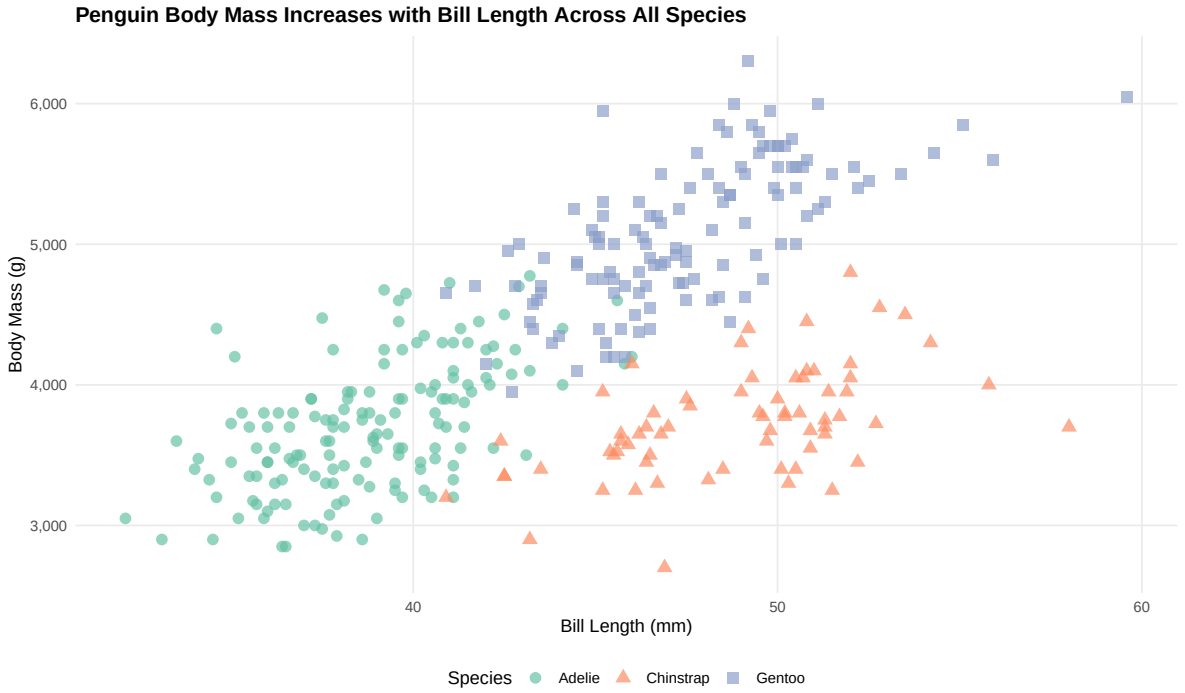
1. Do larger bills correlate with heavier body mass?
2. Are there distinct size differences between species?
3. Can we predict species based on body measurements?

5.2.2 Initial Visualization

```
library(palmerpenguins)
library(ggplot2)

data(penguins)

ggplot(
  penguins,
  aes(
    x = bill_length_mm,
    y = body_mass_g,
    color = species
  )
) +
  geom_point(alpha = 0.7) +
  labs(
    title = "Penguin Body Mass vs Bill Length",
    x = "Bill Length (mm)",
    y = "Body Mass (g)",
    color = "Species"
  )
```

5.2.3 Polished Visualization

```
library(palmerpenguins)
library(ggplot2)
library(scales)

data(penguins)

g <- ggplot(
  penguins,
  aes(
    x = bill_length_mm,
    y = body_mass_g,
    color = species,
    shape = species
  )
)

g <- g + geom_point(alpha = 0.7, size = 3)

g <- g + scale_y_continuous(labels = comma_format())
g <- g + scale_color_brewer(palette = "Set2", name = "Species")
g <- g + scale_shape_manual(
  values = c(16, 17, 15),
  name = "Species"
)

g <- g + labs(
  title = "Penguin Body Mass Increases with Bill Length Across All Species",
  x = "Bill Length (mm)",
  y = "Body Mass (g)"
)

g <- g + theme_minimal()
g <- g + theme(
  legend.position = "bottom",
  plot.title = element_text(face = "bold", size = 13),
  panel.grid.minor = element_blank()
)

g
```



Alt Text: Scatter of body mass vs bill length. Species by color/shape: Adelie = teal circles, Chinstrap = orange triangles, Gentoo = green squares.

Long Description: This scatter plot shows clear size differences among Palmer penguins by plotting bill length against body mass. Species are distinguished with both color and shape, so the plot works for colorblind readers and in grayscale. Gentoo penguins (green squares) form a distinct upper-right cluster: about 4,000–6,500 g and 40–50 mm—indicating they are substantially larger than the other species. Adelie and Chinstrap overlap more, but Chinstrap tends to have slightly longer bills for a given body mass.

5.2.4 Narrative

The plot shows three species separated by size: Gentoo are clearly larger, clustering around the top right, while Adelie and Chinstrap overlap more on the left. Dual encoding (color and shape) keeps species easy to tell apart. Bill length and body mass rise together in all species, suggesting bill length is a simple indicator of overall size and potential differences in foraging niches.

6 What I've Learned So Far

Throughout this course, I've learned how to move beyond just running R code to writing reproducible, documented analyses. Working with the diamonds data and the Popular Baby Names project helped me see how careful wrangling, explicit argument naming, and clear function design make it easier to revisit work later and share it with others.

From the Monte Carlo simulations and visualization activities, I've also gotten more comfortable using ggplot2 and the principles of effective graphics to tell a focused story with data instead of just plotting everything by default. The iterative process of creating an initial visualization, identifying its weaknesses, and then improving it with better colors, scales, and annotations has become second nature.

Finally, building this Quarto portfolio showed me how to integrate narrative, code, and outputs in a single document, which feels much closer to how real data science reports are written and shared. The ability to explain not just what I did, but why I made certain choices and what the results mean, has been one of the most valuable skills I've developed.

```
sessionInfo()
```

```
R version 4.5.1 (2025-06-13 ucrt)
Platform: x86_64-w64-mingw32/x64
Running under: Windows 11 x64 (build 26200)

Matrix products: default
  LAPACK version 3.12.1

locale:
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] palmerpenguins_0.1.1 dcdata_0.5           scales_1.4.0
[4] tibble_3.3.0         purrr_1.1.0          tidyr_1.3.1
```

```
[7] dplyr_1.1.4          ggplot2_3.5.2

loaded via a namespace (and not attached):
 [1] Matrix_1.7-3       gtable_0.3.6       jsonlite_2.0.0     compiler_4.5.1
 [5] tidyselect_1.2.1   splines_4.5.1      yaml_2.3.10        fastmap_1.2.0
 [9] lattice_0.22-7     R6_2.6.1           labeling_0.4.3     generics_0.1.4
[13] knitr_1.50         pillar_1.11.0      RColorBrewer_1.1-3 rlang_1.1.6
[17] utf8_1.2.6         xfun_0.52          viridisLite_0.4.2  cli_3.6.5
[21] withr_3.0.2        magrittr_2.0.3     mgcv_1.9-3         digest_0.6.37
[25] grid_4.5.1         rstudioapi_0.17.1  lifecycle_1.0.4    nlme_3.1-168
[29] vctrs_0.6.5        evaluate_1.0.5     glue_1.8.0         farver_2.1.2
[33] rmarkdown_2.29     tools_4.5.1        pkgconfig_2.0.3    htmltools_0.5.8.1
```