Analyzing Salaries in the AI Job Market

Andrew Mitchell

Analyzing Salaries in the AI Job Market

In our previous exercises, we practiced taking samples and calculating means using some simple measurements. Now let's apply these sampling concepts to the actual dataset and automate the sampling, rather than manually inputting each sample. We'll look at salary data from the AI job market and use sampling to understand the variation in salaries across different job roles.

The previous pages were about introducing you to how Quarto works - here we'll show what an actual analysis document might look like.

You can look at the actual `.qmd` file that generated this page here: `sampling-exercise-extended.qmd`

Setting Up Our Analysis

Load packages:

```
library(tidyverse)
```

Load Data

Now let's load our AI jobs dataset. This dataset contains information about various jobs in the AI industry, including salaries, job titles, and other interesting information:

```
ai_jobs <- read_csv("data/ai_jobs.csv")

# Let's take a look at what job titles we have
ai_jobs |>
  count(job_title) |>
  arrange(desc(n))
```

```
# A tibble: 10 × 2
   job_title                 n
   <chr>                 <int>
 1 Data Scientist           62
 2 HR Manager               57
 3 Cybersecurity Analyst    55
 4 UX Designer              54
 5 AI Researcher            51
 6 Sales Manager            49
```

```
 7 Marketing Specialist     48
 8 Operations Manager       44
 9 Software Engineer        41
10 Product Manager          39
```

Looking at the output above, we can see we have several different job titles in our dataset. For this analysis, let's focus on comparing salaries between different levels of automation risk. Our goal is to see whether there is a relationship between salary levels and the risk of a job being automated.

Hint: When working with real data, it's good practice to look at your data first before diving into analysis. This helps you understand what you're working with and spot any potential issues.

Taking Samples from Our Population

Instead of manually recording samples like we did in class, we can use R to take random samples from our dataset. Let's take 1000 samples of size 50 from the low and high automation risk groups.

When looking at this code, try to break it down into its component parts. We are using a `for` loop, which allows us to repeat a process multiple times. Start by understanding what is happening within the for loop - what are we doing each time?

- We're using the pipe operate to filter the dataset, then sample it, calculate the mean of the sample, and output the result into a list.

By repeating this in a `for` loop, we repeat this process many times, each time adding the new calculated sample mean on to the end of the list.

Next, understand how we tell the `for` loop how many times to perform the sampling.

- We define an `n_samples` variable at the beginning. Then, when starting up the `for` loop, we tell it to loop from 1 to `n_samples`: `for(1:n_samples) {}`.

```
sample_size <- 50
n_samples <- 1000

# Set up an empty list to add to
high_sample_means <- c()

for (i in 1:n_samples) {
  # Sample the High automation risk observations
  mean <- ai_jobs |>
    filter(automation_risk == "High") |>
    sample_n(sample_size) |>
    summarise(mean(salary_usd)) |>
    pull()

  # Add to the sample_means list
  high_sample_means <- append(high_sample_means, mean)
}
```

```
# Repeat for the Low risk group
low_sample_means <- c()
for (i in 1:n_samples) {
  # Sample the High automation risk observations
  mean <- ai_jobs |>
    filter(automation_risk == "Low") |>
    sample_n(sample_size) |>
    summarise(mean(salary_usd)) |>
    pull()

  # Add to the sample_means list
  low_sample_means <- append(low_sample_means, mean)
}
```

Now we have our sample means! Let's combine these into a single table:

```
means_table <- tibble(
  "High" = high_sample_means,
  "Low" = low_sample_means,
)
means_table
```

```
# A tibble: 1,000 × 2
     High     Low
    <dbl>   <dbl>
 1 79295.  95774.
 2 80163.  92348.
 3 76897.  99987.
 4 76381.  93937.
 5 81203.  98027.
 6 81979.  96259.
 7 82945. 102826.
 8 80932. 101283.
 9 72527.  93999.
10 79257. 102692.
# i 990 more rows
```

And reshape it into a tidy long table format. This just means we're stacking the two columns from the table above into one column with a label for which automation risk the sample is from:

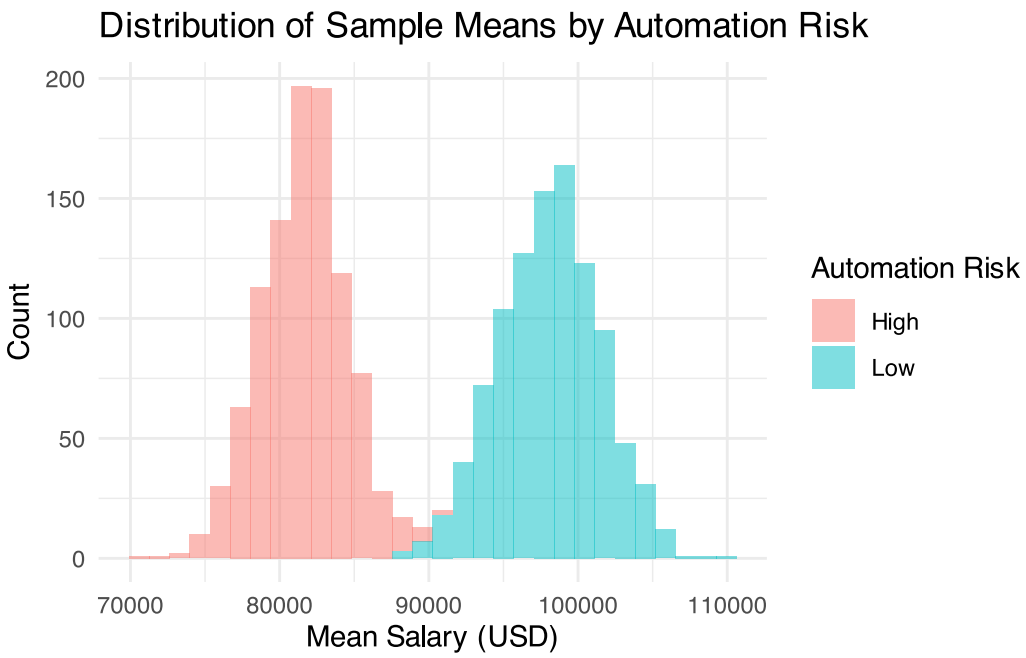
```
means_table <- means_table |>
  gather(key = "automation_risk", value = "sample_mean")

means_table
```

```
# A tibble: 2,000 × 2
   automation_risk sample_mean
   <chr>                 <dbl>
 1 High                 79295.
 2 High                 80163.
 3 High                 76897.
 4 High                 76381.
 5 High                 81203.
 6 High                 81979.
 7 High                 82945.
 8 High                 80932.
 9 High                 72527.
10 High                 79257.
# ℹ 1,990 more rows
```

Visualizing Our Sample Means

Now we can create a visualization to compare the distribution of sample means between these two job titles.

```
# Create a plot comparing the distributions
ggplot(means_table, aes(x = sample_mean, fill = automation_risk)) +
  geom_histogram(alpha = 0.5) +
  theme_minimal() +
  labs(
    title = "Distribution of Sample Means by Automation Risk",
    x = "Mean Salary (USD)",
    y = "Count",
    fill = "Automation Risk"
  )
```



Connecting to the t-test

This visualization shows us how the sample means are distributed for each job title. The overlapping histograms make it easy to compare the two distributions. Another way we could visualise this is with side by side points with error bars:

```
# Create a plot comparing the distributions

# We need to start by calculating the mean and
# standard deviation of the sampling distributions
# to pass to the errorbar layer:
summary_stats <- means_table |>
  group_by(automation_risk) |>
  summarise(
    mean = mean(sample_mean),
    sd = sd(sample_mean)
  ) |>
  ggplot(aes(y = mean, x = automation_risk, color = automation_risk)) +
  geom_point() +
  geom_errorbar(aes(ymin = mean - (2 * sd), ymax = mean + (2 * sd))) +
  theme_minimal() +
  labs(
    title = "Distribution of Sample Means by Automation Risk",
    x = "Automation Risk",
  ) +
  scale_y_continuous(name = "Mean Salary (USD)", labels = scales::comma)
```

This plot shows us quite clearly that the sampling distributions are quite different between the different automation risk groups.

Think back to our Hypothesis Testing using the t-test. The goal there was to tell whether the difference between the means in the two groups was significantly different. How we define statistically significant is based on the estimated sampling distribution. We estimated this based on just a single sample. Here, we're directly looking at the the actually sampling distributions (for sample size = 25). Another way to interpret the t-test statistical significance is to look at whether these sampling distributions are far enough apart *and have a low enough variance* that their standard error bars don't overlap.

In the boxplot, we can see that the external lines of the boxplots (representing 2 standard error) don't overlap with each other. This would indicate that, with at least 95% confidence (remember, 2 SE encompasses 95% of the distribution), the mean of a given sample of 50 salaries from the High risk group would not overlap with the mean from the Low risk group.

Let's translate this into a t-test:

```
# Start by drawing a new sample of 50 from each group
high_sample <- ai_jobs |>
  filter(automation_risk == "High") |>
  sample_n(sample_size) |>
  pull(salary_usd)

low_sample <- ai_jobs |>
  filter(automation_risk == "Low") |>
  sample_n(sample_size) |>
  pull(salary_usd)

# Run the t-test
t.test(high_sample, low_sample)
```

```
    Welch Two Sample t-test

data:  high_sample and low_sample
t = -2.5432, df = 94.137, p-value = 0.01262
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -23915.989  -2945.089
sample estimates:
mean of x mean of y
 79436.77  92867.31
```

Yes! Our t-test results match up to what our plots showed! This demonstrates how the Hypothesis Testing framework allows us to estimate patterns in the sampling distribution even from a single sample.

What Have We Learned?

This exercise shows how sampling helps us understand patterns in real-world data. Instead of looking at every single salary in our dataset, we can take samples and use their means to get a good idea of typical salaries in different groups.

Some key points to notice:

1. We used the same basic sampling concepts as in our class exercise, used the power of R to take many simulated samples.
2. We can easily compare different groups (job titles) by taking samples from each group.
3. Visualizing our sample means helps us see patterns that might not be obvious just looking at numbers.
4. We looked at the relationship between simulating the sampling distribution to estimating the properties of the sampling distribution to apply in hypothesis testing.

Bibliography