MGT4018/MGT4090 Lab 2

Bernd Wurth

Table of contents

1 Introduction

This lab is the same as the SPSS Lab 2 for MGT4018 and MGT4090. We use base R functions as the default. While there are many R packages available, understanding base R operations provides a strong foundation for data analysis.

> Alternatives using R packages
>
> Alternatives for achieving the same outcomes using different R packages are provided in green boxes. If you want to explore these alternatives, each box will introduce the respective package, how to install and use it, and the outputs it can produce.

2 Exercise D: Correlation and Linear Regression

This lab will guide you through creating a dataset, assigning labels, and conducting basic analyses using R. You will learn how to create variables, enter data, and generate summary tables similar to those you would in SPSS.

2.1 Step D1: Setup

You can continue working in the same project (Step A1, Option 1) or working directory (Step A1, Option 2) that you created in the previous lab. You should, however, do the following:

1. Create a new **R script**: Go to **File > New File > R Script**.
2. Save the script in your `scripts` folder with an appropriate name, e.g., `Lab2_Exercise_D_E_F_G.R`.

> Note
>
> You can either work through the following steps and copy/paste the respective code into the `Lab_Exercise_D_E_F_G.R` file that you will create in Step D1 or download the R script for Exercises D, E, F, and G and follow the instructions below and save the downloaded file in the `scripts` folder that you will create.

Now you are ready to begin your work in R and continue with Step D2!

2.2 Step D2: Loading the Dataset

Please download the dataset and save it in the `data` folder within your project folder or working directory.

```r
# Load the CSV file stored in the "data" folder
survey_data_full <- read.csv("data/lab2-survey.csv")
```

You can easily explore and check the basic structure of your data and get a summary:

```r
# View the first few rows using head()
head(survey_data_full)

# Examine the structure
str(survey_data_full)

# Get basic summary statistics
summary(survey_data_full)
```

Similar to Exercises B and C in Lab 1, we want to assign labels to the demographic variables in the `survey_data_full` data frame using **factors** with labeled levels. This method ensures the data remains categorical but with human-readable labels for easier interpretation and analysis.

```r
# Convert demographic variables to factors with labeled levels

# Assign labels for "sex"
survey_data_full$sex <- factor(
  survey_data_full$sex,
  levels = c(1, 2),
  labels = c("Male", "Female")
)

# Assign labels for "marital"
```

```r
survey_data_full$marital <- factor(
  survey_data_full$marital,
  levels = c(1, 2, 3, 4, 5, 6, 7, 8),
  labels = c(
    "Single", "Steady relationship", "Living with partner",
    "Married first time", "Remarried", "Separated",
    "Divorced", "Widowed"
  )
)

# Assign labels for "child"
survey_data_full$child <- factor(
  survey_data_full$child,
  levels = c(1, 2),
  labels = c("Yes", "No")
)

# Assign labels for "educ"
survey_data_full$educ <- factor(
  survey_data_full$educ,
  levels = c(1, 2, 3, 4, 5, 6),
  labels = c(
    "Primary", "Some secondary", "Completed high school",
    "Some additional training", "Completed undergraduate",
    "Postgraduate completed"
  )
)

# Assign labels for "source"
survey_data_full$source <- factor(
  survey_data_full$source,
  levels = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  labels = c(
    "Work", "Spouse or partner", "Relationships", "Children",
    "Family", "Health/illness", "Life in general",
    "Money/finances", "Lack of time, too much to do"
  )
)

# Assign labels for "smoke"
survey_data_full$smoke <- factor(
  survey_data_full$smoke,
  levels = c(1, 2),
  labels = c("Yes", "No")
)
```

The `summary()` function can be used to confirm that the labels have been applied correctly to the variables.

```r
# Verify changes by printing a summary
summary(survey_data_full)
```

2.3 Step D3: Correlation

Correlation analysis helps us understand the relationship between two variables. Before we dive into the practical implementation, let's understand some key concepts.

2.3.1 What is Correlation?

Correlation measures the strength and direction of the relationship between two variables. The **Pearson correlation coefficient (PCC)**, often denoted as `r`, is a statistical measure that quantifies the strength and direction of a linear relationship between two continuous variables. It is one of the most widely used correlation metrics.

The correlation coefficient (r) ranges from -1 to +1:

- r = +1: Perfect positive correlation
- r = 0: No linear correlation
- r = -1: Perfect negative correlation

The magnitude of r (how close it is to ±1) indicates the strength of the relationship, while the sign indicates the direction.

> **Note**
>
> The correlation coefficient is standardized, meaning it's independent of the units of measurement of the original variables. This makes it useful for comparing relationships between different pairs of variables.

2.3.2 Assumptions for Pearson's Correlation

Before calculating correlations, we should check these assumptions:

1. Variables are measured at the interval or ratio level
2. Linear relationship between variables
3. No significant outliers
4. Approximate normal distribution of variables
5. Homoscedasticity (equal variances)

2.3.3 Visual Inspection: Scatterplots

Before calculating correlations, it's important to visualize the relationships:

```r
# Install ggplot2 (NOTE: not needed if you completed the first lab)
install.packages("ggplot2")

# Load ggplot2
library(ggplot2)


# Simple scatterplot
scatter_stress_control <- ggplot(data = survey_data_full, aes(x = tpcoiss, y = tpstress)) +
  geom_point(color = "blue", size = 2) +
  labs(
    title = "Scatterplot of Perceived Stress vs Coping Strategies",
    x = "Coping Strategies (tpcoiss)",
    y = "Perceived Stress (tpstress)"
  ) +
  theme_minimal()

# Show plot
scatter_stress_control
```

2.3.4 Computing Correlations

We can calculate correlation. In the first instance, we focus on the two variables `tpcoiss` and `tpstress`. Afterwards, we will explore how to calculate correlations for all variables at once.

Single Correlation

The `cor()` function calculates the correlation coefficient:

```r
# Calculate correlation between tpcoiss and tpstress
cor(survey_data_full$tpcoiss, survey_data_full$tpstress)

# Specify the method (default is Pearson)
cor(survey_data_full$tpcoiss, survey_data_full$tpstress, method = "pearson")
```

ℹ Note

Best Practices in Base R

1. Always visualize your data first
2. Check assumptions (e.g., normality):

   ```
   # Test for normality
   shapiro.test(x)
   shapiro.test(y1)
   ```

3. Consider different correlation methods when appropriate:

   ```
   # Pearson correlation (default)
   cor(x, y1, method = "pearson")

   # Spearman correlation (for non-normal data)
   cor(x, y1, method = "spearman")

   # Kendall correlation (for ordinal data)
   cor(x, y1, method = "kendall")
   ```

Correlation Matrix

We can calculate correlations for multiple variables variables at once.

First, let's create a smaller data frame with only continuous variables. We can do this in base R using the following code:

```
survey_data_small <- survey_data_full[, c("tpcoiss", "tpstress", "toptim",
                                     "tposaff", "tnegaff", "tlifesat",
                                     "tslfest", "tmarlow")]
```

You can verify the structure of your new data frame using:

```
# Check the structure of the new data frame
str(survey_data_small)

# Or see the first few rows
head(survey_data_small)
```

Tidyverse alternative

You can also use the `tidyverse` package to accomplish the same result: a new data frame containing only these eight variables:

```
library(tidyverse)

survey_data <- survey_data_full %>%
  select(tpcoiss, tpstress, toptim, tposaff,
         tnegaff, tlifesat, tslfest, tmarlow)
```

You can verify the structure of your new data frame using:

```
# Check the structure of the new data frame
str(survey_data_small)

# Or see the first few rows
head(survey_data_small)
```

```
# Calculate correlation matrix
correlation_matrix <- cor(survey_data_small)

# Round to 3 decimal places for clarity
round(correlation_matrix, 3)
```

2.3.5 Statistical Testing with `cor.test()`

The `cor.test()` function provides a full statistical test:

```r
# Perform correlation test
correlation_test <- cor.test(x, y1)

# View complete results
print(correlation_test)
```

This output includes:

- The correlation coefficient
- The test statistic
- The p-value
- The confidence interval

2.3.6 Interpreting Correlation Results

When interpreting correlation results, consider:

1. **Strength**: Common guidelines for absolute values:
    - 0.00 to 0.19: “very weak”
    - 0.20 to 0.39: “weak”
    - 0.40 to 0.59: “moderate”
    - 0.60 to 0.79: “strong”
    - 0.80 to 1.00: “very strong”
2. **Direction**: Positive or negative relationship
3. **Statistical Significance**: Check the p-value
    - $p < 0.05$ typically indicates statistical significance
    - Consider effect size, not just significance
4. **Context**: What’s meaningful in your field?

2.3.7 Common Pitfalls and Considerations

1. **Correlation Causation**: Correlation only indicates association, not causation
2. **Outliers**: Can strongly influence correlation coefficients
3. **Non-linear Relationships**: Pearson’s correlation only measures linear relationships
4. **Missing Data**: Handle missing values appropriately

2.4 Step D4: Linear Regression

Regression analysis helps us understand how one variable (the dependent variable) changes when another variable (the independent variable) changes. Before we dive into the practical implementation, let’s understand some key concepts.

2.4.1 What is Linear Regression?

Linear regression is one of the foundational techniques in statistics. Linear regression gives us a mathematical way to describe and predict the relationship between two variables.

Think of linear regression like drawing a “best-fit” line through a cloud of data points. This line helps us:

1. Understand the relationship between variables.
2. Predict values for new observations.
3. Quantify how strong the relationship is.

2.4.2 Creating Our Linear Regression Model

Let's analyze the relationship between stress (`tpstress`) and sense of control (`tpcoiss`). We'll first visualize the data, then create our regression model.

```
# First, let's create a scatterplot to visualize the relationship
plot(survey_data_full$tpcoiss, survey_data_full$tpstress,
     main = "Relationship between Stress and Control",
     xlab = "Sense of Control (PCOISS)",
     ylab = "Perceived Stress",
     pch = 16,
     col = "darkblue")

# Add our regression line
abline(lm(tpstress ~ tpcoiss, data = survey_data_full),
       col = "red",
       lwd = 2)
```

Now, let's create our regression model using base R:

```
# Create the linear regression model
stress_model <- lm(tpstress ~ tpcoiss, data = survey_data_full)

# View the complete summary
summary(stress_model)
```

2.4.3 Understanding the Results

Let's break down each part of the output to understand what it tells us:

1. The Correlation Coefficient (R)

In our case, we can find the correlation coefficient (r) by taking the square root of R-squared. The negative or positive sign comes from the slope coefficient in our regression output.

```
# Calculate r from our model
r <- sign(coef(stress_model)[2]) * sqrt(summary(stress_model)$r.squared)
cat("Correlation coefficient (r):", round(r, 3))
```

This value tells us:

- The strength of the relationship (how close to -1 or 1)
- The direction (positive or negative)

2. The ANOVA Table

The ANOVA table helps us assess if our model is statistically significant:

```
# Display the ANOVA table
anova(stress_model)
```

Looking at the p-value (Pr(>F)), we can see if our relationship is statistically significant. A value less than 0.05 suggests strong evidence of a real relationship between our variables.

3. The Regression Equation

From our coefficients table, we can write our regression equation:

```
# Display coefficients
coef(stress_model)
```

Our regression equation is:

Stress = + (Control)

Where:

- is our intercept (constant)
- is our slope coefficient

Let's fill in the actual values:

```r
# Extract coefficients
intercept <- coef(stress_model)[1]
slope <- coef(stress_model)[2]

cat("Regression equation:\n")
cat("Stress =", round(intercept, 3), "+", round(slope, 3), "× Control")
```

This equation means:

- When Control = 0, predicted Stress = intercept
- For each one-unit increase in Control, Stress changes by the slope amount

4. R-squared (R^2)

R-squared tells us how much of the variation in stress can be explained by control:

```r
# Extract R-squared
r_squared <- summary(stress_model)$r.squared
cat("R-squared:", round(r_squared, 3))
```

This means that approximately {round(r_squared * 100, 1)}% of the variation in stress levels can be explained by a person's sense of control.

2.4.4 Making Predictions

We can use our model to predict stress levels for new values of control:

```r
# Create some example control values
new_control <- data.frame(tpcoiss = c(20, 30, 40))

# Make predictions
predictions <- predict(stress_model, newdata = new_control)

# Display predictions
cbind(Control = new_control, Predicted_Stress = round(predictions, 2))
```

2.4.5 Checking Model Assumptions

For our regression to be valid, we should check certain assumptions:

```r
# Create diagnostic plots
par(mfrow = c(2, 2))
plot(stress_model)
```

These plots help us check:

1. Linearity (Residuals vs Fitted)
2. Normality (Normal Q-Q)
3. Homoscedasticity (Scale-Location)
4. Influential points (Residuals vs Leverage)

2.4.6 Interpretation Guide

When interpreting your regression results, consider:

1. **Statistical Significance**
   - Look at the p-value in the ANOVA table
   - A p-value < 0.05 suggests a significant relationship
2. **Practical Significance**
   - Look at R-squared to understand how much variance is explained
   - Consider the slope coefficient for practical impact
3. **Direction of Relationship**
   - A negative slope means as one variable increases, the other decreases
   - A positive slope means both variables increase together
4. **Model Assumptions**

- Check diagnostic plots for violations
- Consider transformations if assumptions are violated

Linear Regression Analysis with R Packages

Introduction While base R provides solid foundations for linear regression, modern R packages offer enhanced capabilities for analysis, visualization, and interpretation. We'll use several powerful packages that make our analysis more intuitive and visually appealing:

```r
# Install packages if needed
install.packages(c("tidyverse", "broom", "performance", "see", "ggpubr", "sjPlot"))

# Load required packages
library(tidyverse)    # For data manipulation and visualization
library(broom)        # For tidying statistical objects
library(performance)  # For model performance metrics
library(see)          # For model visualization
library(ggpubr)      # For publication-ready plots
library(sjPlot)      # For model visualization and tables
```

Data Preparation

First, let's prepare our data using `tidyverse` functions:

```r
# Create a focused dataset for analysis
analysis_data <- survey_data_full %>%
  select(tpstress, tpcoiss) %>%
  drop_na()  # Remove any missing values

# Quick summary of our variables
summary(analysis_data)
```

Visual Exploration with `ggplot2`

The `ggplot2` package (part of `tidyverse`) creates beautiful visualizations:

```r
# Create an enhanced scatter plot
ggplot(analysis_data, aes(x = tpcoiss, y = tpstress)) +
  geom_point(alpha = 0.5, color = "steelblue") +
  geom_smooth(method = "lm", color = "red") +
  labs(
    title = "Relationship between Stress and Control",
    subtitle = "With linear regression line and 95% confidence interval",
    x = "Sense of Control (PCOISS)",
    y = "Perceived Stress Level"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```

Creating the Regression Model

We'll create our model and use modern packages to examine it:

```r
# Create the model
model <- lm(tpstress ~ tpcoiss, data = analysis_data)

# Get a tidy summary using broom
tidy_model <- tidy(model, conf.int = TRUE)
glance_model <- glance(model)

# Display tidy results
tidy_model
```

Model Diagnostics with `performance`

The `performance` package provides enhanced diagnostic tools:

```r
# Check model assumptions
check_model(model)

# Model performance metrics
model_performance(model)
```

Creating Publication-Ready Tables with `sjPlot`

The `sjPlot` package creates beautiful HTML and Word-compatible tables:

```r
# Create regression table
tab_model(model,
          title = "Linear Regression Results",
          dv.labels = "Perceived Stress",
          pred.labels = c("(Intercept)", "Sense of Control"))
```

Visualizing Effects with `ggeffects`

We can visualize the relationship more clearly:

```r
# Plot predicted values
plot_model(model, type = "pred") +
  labs(
    title = "Predicted Stress Levels by Control",
    x = "Sense of Control",
    y = "Predicted Stress Level"
  )
```

Enhanced Regression Diagnostics

Let's create more informative diagnostic plots using `ggplot2`:

```r
# Get augmented data (includes residuals, etc.)
model_data <- augment(model)

# Create diagnostic plots
p1 <- ggplot(model_data, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "Residuals vs Fitted",
       x = "Fitted values",
       y = "Residuals") +
  theme_minimal()

p2 <- ggplot(model_data, aes(sample = .std.resid)) +
  stat_qq() +
  stat_qq_line() +
  labs(title = "Normal Q-Q Plot") +
  theme_minimal()

# Arrange plots side by side
ggarrange(p1, p2, ncol = 2)
```

Making Predictions

We can use tidyverse functions for predictions:

```r
# Create new data for predictions
new_data %
  bind_cols(predict(model, newdata = ., interval = "confidence"))

# Plot predictions
ggplot(predictions, aes(x = tpcoiss)) +
  geom_ribbon(aes(ymin = lwr, ymax = upr),
              alpha = 0.2, fill = "blue") +
  geom_line(aes(y = fit), color = "blue") +
  geom_point(data = analysis_data,
             aes(y = tpstress),
             alpha = 0.5) +
  labs(
    title = "Regression Line with Confidence Interval",
    x = "Sense of Control",
    y = "Stress Level"
  ) +
  theme_minimal()
```

Interactive Model Summary

We can create an interactive summary of our model:

```r
# Create model summary
summary_stats <- tibble(
  Statistic = c("R-squared", "Adjusted R-squared", "F-statistic", "p-value"),
  Value = c(
    glance_model$r.squared,
    glance_model$adj.r.squared,
    glance_model$statistic,
    glance_model$p.value
  )
) %>%
  mutate(Value = round(Value, 3))

# Display as a formatted table
knitr::kable(summary_stats,
             caption = "Model Summary Statistics")
```

Advantages of Using Modern Packages

These modern R packages offer several advantages over base R:

1. **Better Visualization**: `ggplot2` creates publication-quality graphics
2. **Tidy Output**: `broom` makes statistical output easier to work with
3. **Enhanced Diagnostics**: `performance` provides comprehensive model checking
4. **Publication-Ready Tables**: `sjPlot` creates professional tables
5. **Consistent Interface**: `tidyverse` provides a coherent framework

3 Exercise E: T-Test

A t-test helps us determine whether there's a meaningful difference between groups in our data. Imagine you're wondering whether men and women tend to have different levels of self-esteem. A t-test can tell us if any difference we observe is likely to be real or just due to chance.

Think of a t-test like a referee in a debate: it helps us decide whether we have enough evidence to say two groups are truly different from each other. Just as a referee needs clear rules to make fair decisions, our t-test uses statistical principles to make this determination.

3.1 Step E1: Understanding the Independent Samples t-test

The independent samples t-test is used when we have:

- A continuous dependent variable (like self-esteem scores)
- Two independent groups (like males and females)

- Independent observations (one person's score doesn't influence another's)

Before we dive into our analysis, let's prepare our data and create some helpful visualizations:

```r
# First, let's see what our data looks like
head(survey_data_full[c("sex", "tslfest", "tpcoiss")])

# Create visualization of self-esteem by gender
boxplot(tslfest ~ sex,
        data = survey_data_full,
        main = "Self-esteem Levels by Gender",
        xlab = "Gender (1 = Male, 2 = Female)",
        ylab = "Total Self-esteem Score",
        col = c("lightblue", "lightpink"))

# Add individual points for better visualization
stripchart(tslfest ~ sex,
           data = survey_data_full,
           vertical = TRUE,
           method = "jitter",
           add = TRUE,
           pch = 20,
           col = "darkgray")
```

3.2 Step E2: Performing the T-Test (Self-esteem by Gender)

Let's conduct our first t-test examining self-esteem differences between genders:

```r
# First, let's check basic descriptive statistics
tapply(survey_data_full$tslfest,
       survey_data_full$sex,
       function(x) c(mean = mean(x, na.rm = TRUE),
                     sd = sd(x, na.rm = TRUE),
                     n = sum(!is.na(x))))

# Perform Levene's test for equality of variances
var_test <- var.test(tslfest ~ sex, data = survey_data_full)
print("Levene's test results:")
print(var_test)

# Perform t-test based on Levene's test result
t_test_result <- t.test(tslfest ~ sex,
                        data = survey_data_full,
                        var.equal = var_test$p.value > 0.05)  # true if p > 0.05
print("\nt-test results:")
print(t_test_result)
```

Let's break down what these results tell us:

1. **Levene's Test** The p-value from our variance test helps us decide whether to assume equal variances. If $p > 0.05$, we assume equal variances and use the standard t-test. If $p < 0.05$, we use the Welch's t-test which doesn't assume equal variances.

2. **t-test Results**

   - The t-statistic tells us how many standard errors the group means are apart
   - The degrees of freedom (df) help us determine critical values
   - The p-value tells us the probability of seeing such differences by chance
   - The confidence interval shows us the likely range of the true difference

3. **Effect Size** Let's calculate Cohen's d to understand the practical significance:

```r
# Calculate Cohen's d
group1 <- survey_data_full$tslfest[survey_data_full$sex == 1]
group2 <- survey_data_full$tslfest[survey_data_full$sex == 2]
```

```
cohens_d <- function(x, y) {
  nx <- length(x)
  ny <- length(y)
  pooled_sd <- sqrt(((nx-1)*var(x) + (ny-1)*var(y))/(nx+ny-2))
  abs(mean(x) - mean(y))/pooled_sd
}

effect_size <- cohens_d(group1, group2)
cat("Cohen's d effect size:", round(effect_size, 3))
```

T-Tests with R Packages

We can use several R packages that provide enhanced capabilities for conducting and visualizing t-tests and work together to create a comprehensive analysis:

```
# Install packages if needed
install.packages(c("tidyverse", "car", "rstatix", "ggpubr", "effectsize"))

# Load required packages
library(tidyverse)  # For data manipulation and visualization
library(car)        # For Levene's test and additional diagnostics
library(rstatix)    # For statistical analysis
library(ggpubr)    # For publication-ready plots
library(effectsize) # For effect size calculations
```

Data Preparation

First, let's prepare our data and ensure it's in the right format for analysis:

```
# Convert sex to a factor with meaningful labels

analysis_data <- survey_data_full %>%
  mutate(sex = factor(sex,
                      levels = c(1, 2),
                      labels = c("Male", "Female"))) %>%
  select(sex, tslfest, tpcoiss)

# Display the structure of our prepared data
str(analysis_data)
```

Visual Exploration

The `ggpubr` package creates beautiful and informative visualizations:

```
# Create boxplot with individual points
ggboxplot(analysis_data,
          x = "sex",
          y = "tslfest",
          color = "sex",
          add = "jitter",
          title = "Self-esteem Levels by Gender") +
  theme_minimal() +
  labs(y = "Total Self-esteem Score",
       x = "Gender")

# Add descriptive statistics
desc_stats %
  group_by(sex) %>%
  get_summary_stats(tslfest, type = "common")
print(desc_stats)
```

Checking Assumptions

Modern packages provide comprehensive tools for checking t-test assumptions:

1. Normality Check

We can check normality both visually and statistically:

```r
# Create Q-Q plots by group
ggqqplot(analysis_data, "tslfest", facet.by = "sex")

# Shapiro-Wilk test for each group
analysis_data %>%
  group_by(sex) %>%
  shapiro_test(tslfest)
```

2. Homogeneity of Variances

The `car` package provides a robust Levene's test:

```r
# Levene's test using car package
leveneTest(tslfest ~ sex, data = analysis_data)
```

Conducting the T-Test

The `rstatix` package provides a comprehensive t-test function:

```r
# Perform t-test
t_test_results <- analysis_data %>%
  t_test(tslfest ~ sex) %>%
  add_significance()

# Display results
t_test_results
```

Effect Size Calculation

The `effectsize` package helps us understand the practical significance:

```r
# Calculate Cohen's d
cohens_d <- cohens_d(tslfest ~ sex, data = analysis_data)
print(cohens_d)

# Interpret effect size
interpret_cohens_d(cohens_d$Cohens_d)
```

Complete Analysis for Control Levels

Let's apply the same comprehensive analysis to our second research question about control levels:

```r
# Visualization
ggboxplot(analysis_data,
          x = "sex",
          y = "tpcoiss",
          color = "sex",
          add = "jitter",
          title = "Control Levels by Gender") +
  theme_minimal() +
  labs(y = "Total Control Score",
       x = "Gender")

# Check assumptions
# 1. Normality
analysis_data %>%
  group_by(sex) %>%
  shapiro_test(tpcoiss)

# 2. Homogeneity of variance
leveneTest(tpcoiss ~ sex, data = analysis_data)

# Perform t-test
control_t_test <- analysis_data %>%
  t_test(tpcoiss ~ sex) %>%
  add_significance()

# Calculate effect size
control_effect <- cohens_d(tpcoiss ~ sex, data = analysis_data)

# Display comprehensive results
list(
  "T-test Results" = control_t_test,
  "Effect Size" = control_effect
)
```

Advantages of Using Modern Packages

These modern R packages offer several benefits over base R:

1. The `rstatix` package provides:
   - Clearer output formatting
   - Automatic significance indicators
   - Built-in assumption checking
2. The `ggpubr` package creates:
   - Publication-ready plots
   - Easy visualization of assumptions
   - Combined plots with statistical results
3. The `effectsize` package offers:
   - Standardized effect size calculations
   - Effect size interpretation
   - Confidence intervals for effect sizes

Creating a Complete Report

We can create a comprehensive analysis report:

```r
# Function to run complete analysis
analyze_group_difference <- function(data, dv, group_var) {
  # Descriptive statistics
  desc <- data %>%
    group_by(!!sym(group_var)) %>%
    get_summary_stats(!!sym(dv), type = "common")

  # Assumption checks
  normality <- data %>%
    group_by(!!sym(group_var)) %>%
    shapiro_test(!!sym(dv))

  # Levene's test
  homogeneity <- leveneTest(as.formula(paste(dv, "~", group_var)), data = data)

  # T-test
  t_test <- data %>%
    t_test(as.formula(paste(dv, "~", group_var))) %>%
    add_significance()

  # Effect size
  effect <- cohens_d(as.formula(paste(dv, "~", group_var)), data = data)

  list(
    descriptives = desc,
    normality_test = normality,
    variance_test = homogeneity,
    t_test = t_test,
    effect_size = effect
  )
}

# Example usage
self_esteem_analysis <- analyze_group_difference(
  analysis_data,
  "tslfest",
  "sex"
)

# Display results
print(self_esteem_analysis)
```

Writing Up Results

Here’s how to write up the results in APA format using our comprehensive analysis:

```r
# Create formatted output
report_results <- function(analysis) {
  with(analysis, {
    cat("Results:\n")
    cat("Descriptive Statistics:\n")
    print(descriptives)
    cat("\nAssumption Tests:\n")
    cat("Normality (Shapiro-Wilk):\n")
    print(normality_test)
    cat("\nHomogeneity of Variance (Levene's Test):\n")
    print(variance_test)
    cat("\nT-test Results:\n")
    print(t_test)
    cat("\nEffect Size:\n")
    print(effect_size)
  })
}

report_results(self_esteem_analysis)
```

3.3 Step E3: Performing the T-Test (Control Levels by Gender)

Now let's examine our second research question about control levels between genders:

```r
# Visualize control levels by gender
boxplot(tpcoiss ~ sex,
        data = survey_data_full,
        main = "Control Levels by Gender",
        xlab = "Gender (1 = Male, 2 = Female)",
        ylab = "Total Control Score",
        col = c("lightblue", "lightpink"))

stripchart(tpcoiss ~ sex,
           data = survey_data_full,
           vertical = TRUE,
           method = "jitter",
           add = TRUE,
           pch = 20,
           col = "darkgray")

# Perform the analysis
var_test_control  0.05)
print("\nt-test results for control:")
print(t_test_control)
```

3.4 Step E4: A Step-by-Step Guide to Interpreting Results

When interpreting t-test results, follow these steps:

1. **Check Assumptions**
   - Look at the boxplots for obvious outliers
   - Check Levene's test for equality of variances
     - If Levene's test is significant ($p < 0.05$), use the Welch's t-test output (R does this automatically with var.equal = FALSE).
   - Consider whether the groups are truly independent
   - Check if the data is normally distributed
     - If the data is not normally distributed, consider using non-parametric alternatives like the Wilcoxon rank-sum test:

```r
# Example of non-parametric test
wilcox.test(tslfest ~ sex, data = survey_data_full)
```

2. **Examine the t-test Results**
   - Look at the p-value (significance level)
   - Consider the confidence interval
   - Think about the effect size
3. **Draw Conclusions**
   - If $p < 0.05$, we have evidence of a significant difference
   - Consider the practical significance (effect size)
   - Think about the real-world implications

3.5 Step E5: Writing Up Results

Here is an example for how to write up t-test results in APA format:

> “An independent-samples t-test was conducted to compare [variable] between males and females. There was a [significant/non-significant] difference in scores for males (M = XX.XX, SD = XX.XX) and females (M = XX.XX, SD = XX.XX); t(df) = XX.XX, p = .XXX. The magnitude of the differences in the means (mean difference = XX.XX, 95% CI: XX.XX to XX.XX) was [small/medium/large] (d = X.XX).”

4 Exercise F: Analysis of Variance (ANOVA)

Analysis of Variance, commonly known as ANOVA, extends our analytical capabilities beyond comparing just two groups. While t-tests are perfect for comparing means between two groups (like comparing males and females), researchers often need to examine differences across multiple groups. For instance, we might want to know how optimism levels differ across various age groups, or how stress levels vary with education.

Think of ANOVA like a sophisticated referee in a multi-team tournament: instead of just comparing two teams, it helps us determine whether any teams in the entire tournament are significantly different from each other. Just as a tournament referee needs clear rules to make fair decisions across multiple teams, ANOVA uses statistical principles to examine differences across multiple groups simultaneously.

4.1 Step F1: Understanding One-way ANOVA

One-way ANOVA is used when we have:

- A continuous dependent variable (like optimism scores).
- One categorical independent variable with three or more groups (like age groups).
- Independent observations (one person’s score doesn’t influence another’s).

Let’s begin by preparing our data and creating some helpful visualizations:

```r
# First, let's prepare our age groups properly
survey_data_full$agegp3 <- factor(survey_data_full$agegp3,
                                 levels = c(1, 2, 3),
                                 labels = c("18-29", "30-44", "45+"))

# Create an informative boxplot
boxplot(toptim ~ agegp3,
        data = survey_data_full,
        main = "Optimism Levels by Age Group",
        xlab = "Age Group",
        ylab = "Total Optimism Score",
        col = c("lightblue", "lightgreen", "lightpink"))

# Add individual points for better visualization
stripchart(toptim ~ agegp3,
           data = survey_data_full,
           vertical = TRUE,
           method = "jitter",
           add = TRUE,
           pch = 20,
           col = "darkgray")
```

4.2 Step F2: Performing the One-Way ANOVA (Optimism Across Age Groups)

Before diving into the ANOVA, let's examine our data descriptively:

```r
# Create comprehensive descriptive statistics
get_descriptives <- function(dv, group) {
    tapply(dv, group, function(x) {
        c(n = sum(!is.na(x)),
          mean = mean(x, na.rm = TRUE),
          sd = sd(x, na.rm = TRUE),
          se = sd(x, na.rm = TRUE) / sqrt(sum(!is.na(x))))
    })
}

# Get descriptives for optimism by age group
descriptives <- get_descriptives(survey_data_full$toptim,
                               survey_data_full$agegp3)
print("Descriptive Statistics:")
print(descriptives)
```

Just as t-tests have assumptions, ANOVA requires certain conditions to be met:

```r
# Levene's test for homogeneity of variance
levene_test <- function(y, group) {
    group <- factor(group)
    meds <- tapply(y, group, median, na.rm = TRUE)
    abs_dev <- abs(y - meds[group])
    anova(lm(abs_dev ~ group))[1, "Pr(>F)"]
}

# Perform Levene's test
levene_p <- levene_test(survey_data_full$toptim, survey_data_full$agegp3)
cat("Levene's test p-value:", levene_p, "\n")

# Brown-Forsythe test for a more robust check
bf_test <- function(y, group) {
    group <- factor(group)
    meds <- tapply(y, group, median, na.rm = TRUE)
    abs_dev <- abs(y - meds[group])
    anova(lm(abs_dev ~ group))[1, "Pr(>F)"]
}

bf_p <- bf_test(survey_data_full$toptim, survey_data_full$agegp3)
cat("Brown-Forsythe test p-value:", bf_p, "\n")
```

Now we'll perform both standard ANOVA and its robust alternative:

```r
# Standard one-way ANOVA
anova_result <- aov(toptim ~ agegp3, data = survey_data_full)
print("Standard ANOVA results:")
print(summary(anova_result))

# Welch's ANOVA (robust to unequal variances)
welch_result <- oneway.test(toptim ~ agegp3,
                           data = survey_data_full,
                           var.equal = FALSE)
print("\nWelch's ANOVA results:")
print(welch_result)
```

If we find significant differences, we need to know which groups differ from each other:

```r
# Tukey's HSD test for pairwise comparisons
tukey_result <- TukeyHSD(anova_result)
print("Tukey's HSD test results:")
print(tukey_result)

# Visualize the Tukey results
plot(tukey_result)
```

Let's calculate eta-squared to understand the practical significance:

```
# Calculate eta-squared
aov_summary <- summary(anova_result)
eta_squared <- aov_summary[[1]]$"Sum Sq"[1] / sum(aov_summary[[1]]$"Sum Sq")
cat("Eta-squared:", round(eta_squared, 3))
```

ANOVA with R Packages

Modern R packages provide enhanced capabilities for conducting and visualizing Analysis of Variance (ANOVA). We'll use several powerful packages that work together to create a comprehensive analysis:

```
# Install packages if needed
install.packages(c("tidyverse", "car", "rstatix", "ggpubr", "effectsize", "emmeans"))

# Load required packages
library(tidyverse)  # For data manipulation and visualization
library(car)        # For Levene's test and additional diagnostics
library(rstatix)    # For statistical analysis
library(ggpubr)    # For publication-ready plots
library(effectsize) # For effect size calculations
library(emmeans)    # For estimated marginal means and post-hoc tests
```

Data Preparation

First, let's prepare our data and ensure it's in the right format for analysis:

```
# Convert age groups to a factor with meaningful labels
analysis_data <- survey_data_full %>%
  mutate(age_group = factor(agegp3,
                           levels = c(1, 2, 3),
                           labels = c("18-29", "30-44", "45+")),
         education = factor(educrec)) %>%
  select(age_group, education, toptim, tpstress)

# Display the structure of our prepared data
glimpse(analysis_data)
```

Visual Exploration

The `ggpubr` package creates beautiful and informative visualizations:

```
# Create violin plot with boxplot and individual points
ggviolin(analysis_data,
         x = "age_group",
         y = "toptim",
         fill = "age_group",
         add = "boxplot",
         add.params = list(fill = "white")) +
  geom_jitter(width = 0.2, alpha = 0.5) +
  theme_minimal() +
  labs(title = "Optimism Levels by Age Group",
       y = "Total Optimism Score",
       x = "Age Group") +
  theme(legend.position = "none")

# Add descriptive statistics
desc_stats <- analysis_data %>%
  group_by(age_group) %>%
  get_summary_stats(toptim, type = "common")
print(desc_stats)
```

Checking Assumptions

Modern packages provide comprehensive tools for checking ANOVA assumptions:

1. Normality Check

We can check normality both visually and statistically:

```
# Create Q-Q plots by group
ggqqplot(analysis_data, "toptim", facet.by = "age_group")

# Shapiro-Wilk test for each group
analysis_data %>%
  group_by(age_group) %>%
  shapiro_test(toptim)
```

2. Homogeneity of Variances

The car package provides both Levene's and Brown-Forsythe tests:

```
# Levene's test using car package
leveneTest(toptim ~ age_group, data = analysis_data)

# Brown-Forsythe test using car package
leveneTest(toptim ~ age_group, data = analysis_data, center = median)
```

Conducting the ANOVA

The `rstatix` package provides comprehensive ANOVA functions:

```
# Perform one-way ANOVA
anova_results <- analysis_data %>%
  anova_test(toptim ~ age_group) %>%
  add_significance()

# Display results
anova_results

# Welch's ANOVA (robust to heterogeneity of variance)
welch_results <- analysis_data %>%
  welch_anova_test(toptim ~ age_group)

print("Welch's ANOVA results:")
welch_results
```

Post-hoc Analysis

Modern packages offer various methods for post-hoc analysis:

```r
# Tukey's HSD using emmeans
emmeans_result <- emmeans(aov(toptim ~ age_group, data = analysis_data),
                         "age_group")
pairs(emmeans_result)

# Games-Howell post-hoc test (robust to heterogeneity of variance)
games_howell_result <- analysis_data %>%
  games_howell_test(toptim ~ age_group)
print("Games-Howell test results:")
games_howell_result
```

Effect Size Calculation

The `effectsize` package helps us understand the practical significance:

```r
# Calculate eta squared
eta_squared <- effectsize::eta_squared(aov(toptim ~ age_group, data = analysis_data))
print(eta_squared)

# Interpret effect size
interpret_eta_squared(eta_squared$Eta2)
```

Complete Analysis: Stress Levels Across Age Groups

Let's apply the same comprehensive analysis to perceived stress levels:

```
# Visualization
ggviolin(analysis_data,
         x = "age_group",
         y = "tpstress",
         fill = "age_group",
         add = "boxplot",
         add.params = list(fill = "white")) +
  geom_jitter(width = 0.2, alpha = 0.5) +
  theme_minimal() +
  labs(title = "Stress Levels by Age Group",
       y = "Total Perceived Stress Score",
       x = "Age Group") +
  theme(legend.position = "none")

# Function for comprehensive ANOVA analysis
analyze_group_differences <- function(data, dv, group_var) {
  # Descriptive statistics
  desc <- data %>%
    group_by(!!sym(group_var)) %>%
    get_summary_stats(!!sym(dv), type = "common")

  # Assumption checks
  normality <- data %>%
    group_by(!!sym(group_var)) %>%
    shapiro_test(!!sym(dv))

  # Levene's test
  homogeneity <- leveneTest(as.formula(paste(dv, "~", group_var)), data = data)

  # ANOVA
  anova <- anova_test(data = data, as.formula(paste(dv, "~", group_var)))

  # Welch's ANOVA
  welch <- welch_anova_test(data = data, as.formula(paste(dv, "~", group_var)))

  # Post-hoc tests
  posthoc <- games_howell_test(data = data, as.formula(paste(dv, "~", group_var)))

  # Effect size
  model <- aov(as.formula(paste(dv, "~", group_var)), data = data)
  effect <- effectsize::eta_squared(model)

  # Return results
  list(
    descriptives = desc,
    normality_test = normality,
    variance_test = homogeneity,
    anova_test = anova,
    welch_test = welch,
    posthoc = posthoc,
    effect_size = effect
  )
}

# Run complete analysis
stress_analysis <- analyze_group_differences(
  analysis_data,
  "toptim",
  "age_group"
)

# Display results
print(stress_analysis)
```

Creating Powerful Visualizations

We can create publication-ready plots that combine statistical information:

```
# Create plot with statistical annotations
ggboxplot(analysis_data,
          x = "age_group",
          y = "toptim",
          color = "age_group") +
  stat_compare_means(method = "anova") +
  geom_jitter(width = 0.2, alpha = 0.5) +
  theme_minimal() +
  labs(title = "Optimism Levels by Age Group with ANOVA Results",
       y = "Total Optimism Score",
       x = "Age Group") +
  theme(legend.position = "none")
```

Advantages of Using Modern Packages

These modern R packages offer several benefits over base R:

1. The `rstatix` package provides:
   - Tidyverse-compatible statistical functions
   - Automatic assumption checking
   - Built-in effect size calculations
2. The `ggpubr` package creates:
   - Publication-ready plots
   - Easy statistical annotations
   - Combined plots with results
3. The `emmeans` package offers:
   - Sophisticated post-hoc analyses
   - Flexible contrast specifications
   - Multiple comparison adjustments

Writing Up Results

Here's how to write up the results in APA format:

```
# Function to create formatted output
report_anova_results <- function(analysis) {
  with(analysis, {
    cat("Results:\n")
    cat("\nDescriptive Statistics:\n")
    print(descriptives)
    cat("\nANOVA Results:\n")
    print(anova_test)
    cat("\nEffect Size:\n")
    print(effect_size)
    cat("\nPost-hoc Analysis:\n")
    print(posthoc)
  })
}

report_anova_results(stress_analysis)
```

4.3 Step F3: Performing the One-Way ANOVA (Stress Levels Across Age Groups)

Let's apply what we've learned to analyze stress levels (`tpstress`) across age groups. First, we create some helpful visualizations:

```
# Create an informative boxplot
boxplot(tpstress ~ agegp3,
        data = survey_data_full,
        main = "Total Perceived Stress by Age Group",
        xlab = "Age Group",
        ylab = "Total Perceived Stress",
        col = c("lightblue", "lightgreen", "lightpink"))
```

```
# Add individual points for better visualization
stripchart(tpstress ~ agegp3,
           data = survey_data_full,
           vertical = TRUE,
           method = "jitter",
           add = TRUE,
           pch = 20,
           col = "darkgray")
```

Before diving into the ANOVA, let's examine our data descriptively:

```
# Create comprehensive descriptive statistics
get_descriptives <- function(dv, group) {
    tapply(dv, group, function(x) {
        c(n = sum(!is.na(x)),
          mean = mean(x, na.rm = TRUE),
          sd = sd(x, na.rm = TRUE),
          se = sd(x, na.rm = TRUE) / sqrt(sum(!is.na(x))))
    })
}

# Get descriptives for total perceived stress by age group
descriptives <- get_descriptives(survey_data_full$tpstress,
                                survey_data_full$agegp3)
print("Descriptive Statistics:")
print(descriptives)
```

Just as t-tests have assumptions, ANOVA requires certain conditions to be met:

```
# Levene's test for homogeneity of variance
levene_test <- function(y, group) {
    group <- factor(group)
    meds <- tapply(y, group, median, na.rm = TRUE)
    abs_dev <- abs(y - meds[group])
    anova(lm(abs_dev ~ group))[1, "Pr(>F)"]
}

# Perform Levene's test
levene_p <- levene_test(survey_data_full$tpstress, survey_data_full$agegp3)
cat("Levene's test p-value:", levene_p, "\n")

# Brown-Forsythe test for a more robust check
bf_test <- function(y, group) {
    group <- factor(group)
    meds <- tapply(y, group, median, na.rm = TRUE)
    abs_dev <- abs(y - meds[group])
    anova(lm(abs_dev ~ group))[1, "Pr(>F)"]
}

bf_p <- bf_test(survey_data_full$tpstress, survey_data_full$agegp3)
cat("Brown-Forsythe test p-value:", bf_p, "\n")
```

Now we'll perform both standard ANOVA and its robust alternative:

```
# Standard one-way ANOVA
anova_result <- aov(tpstress ~ agegp3, data = survey_data_full)
print("Standard ANOVA results:")
print(summary(anova_result))

# Welch's ANOVA (robust to unequal variances)
welch_result <- oneway.test(tpstress ~ agegp3,
                           data = survey_data_full,
                           var.equal = FALSE)
print("\nWelch's ANOVA results:")
print(welch_result)
```

If we find significant differences, we need to know which groups differ from each other:

```
# Tukey's HSD test for pairwise comparisons
tukey_result <- TukeyHSD(anova_result)
print("Tukey's HSD test results:")
print(tukey_result)

# Visualize the Tukey results
plot(tukey_result)
```

Let's calculate eta-squared to understand the practical significance:

```
# Calculate eta-squared
aov_summary <- summary(anova_result)
eta_squared <- aov_summary[[1]]$"Sum Sq"[1] / sum(aov_summary[[1]]$"Sum Sq")
cat("Eta-squared:", round(eta_squared, 3))
```

4.4 Step F4: A Step-by-Step Guide to Interpreting Results

When interpreting ANOVA results, follow these steps:

1. **Check Assumptions**
   - Review Levene's and Brown-Forsythe test results.
   - Consider whether observations are truly independent.
   - ANOVA can handle unequal sample sizes, but you should pay extra attention to the homogeneity of variance assumption.
   - Examine the boxplots for outliers and distribution shape. If the data isn't normally distributed, consider using the non-parametric Kruskal-Wallis test:

```
# Example of non-parametric alternative
kruskal.test(toptim ~ agegp3, data = survey_data_full)
```

2. **Examine the ANOVA Results**
   - Look at the F-statistic and p-value
   - Consider which ANOVA version to trust (standard or Welch's)
   - Examine effect sizes
3. **Interpret Post-hoc Tests**
   - Look for significant pairwise differences
   - Consider the practical meaning of the differences
   - Think about the confidence intervals

4.5 Step F5: Writing Up Results

Here's how to write up ANOVA results in APA format:

> "A one-way between-groups analysis of variance was conducted to explore the impact of age on optimism levels. Participants were divided into three groups according to their age (Group 1: 18-29 years; Group 2: 30-44 years; Group 3: 45+ years). There was a [significant/non-significant] difference in optimism scores for the three age groups: F(2, XXX) = XX.XX, p = .XXX. The effect size, calculated using eta squared, was .XX. Post-hoc comparisons using the Tukey HSD test indicated that the mean score for Group 1 (M = XX.XX, SD = XX.XX) was significantly different from Group 2 (M = XX.XX, SD = XX.XX)…"

5 Exercise G: Chi-Square Test

Chi-Square tests help us analyze categorical data - information that falls into distinct groups or categories. Think of these tests as ways to determine whether the patterns we see in our data are meaningful or just due to chance. We'll explore two main types of Chi-Square tests:

1. **Chi-Square Test for Goodness of Fit**: Compares our observed data to what we would expect based on some theory or known population values.
2. **Chi-Square Test for Independence**: Examines whether two categorical variables are related to each other.

5.1 Step G1: Chi-Square Test for Goodness of Fit (Smoking Rates)

Imagine you're wondering whether the proportion of smokers in your sample matches what you'd expect based on national statistics. This is exactly what a goodness of fit test helps us determine.

In this example, we check whether our sample's smoking rate matches the expected population rate of 20%:

```r
# First, let's look at our data
smoker_table <- table(survey_data_full$smoker)
print("Observed frequencies:")
print(smoker_table)

# Calculate percentages
smoker_props <- prop.table(smoker_table) * 100
print("\nObserved percentages:")
print(round(smoker_props, 1))

# Perform Chi-Square test
# Expected proportions (20% smokers, 80% non-smokers)
expected_props <- c(0.2, 0.8)

chisq_result <- chisq.test(smoker_table, p = expected_props)

# Display detailed results
print("\nChi-Square Test Results:")
print(chisq_result)

# Calculate and display expected frequencies
n_total <- sum(smoker_table)
expected_freq <- n_total * expected_props

print("\nComparison of observed vs expected frequencies:")
comparison_df <- data.frame(
  Category = c("Smokers", "Non-smokers"),
  Observed = as.numeric(smoker_table),
  Expected = expected_freq,
  Difference = as.numeric(smoker_table) - expected_freq
)
print(comparison_df)
```

5.2 Step G2: Interpreting Goodness of Fit Results

The Chi-Square test gives us several pieces of information:

1. The Chi-Square statistic (^{2}): Measures how different our observed values are from what we expected
2. Degrees of freedom (df): The number of categories minus 1
3. p-value: Tells us whether any difference is statistically significant

In our example:

- If $p < 0.05$: The smoking rate in our sample is significantly different from 20%
- If p 0.05: We don't have evidence that our sample differs from the expected 20%

Chi-Square Test for Goodness of Fit with R Packages

Modern R packages provide enhanced capabilities for conducting and visualizing Chi-Square analyses. We'll use several powerful packages that work together to create comprehensive and intuitive analyses:

```
# Install packages if needed
install.packages(c("tidyverse", "rstatix", "ggstatsplot", "ggsignif",
                   "corrplot", "effectsize", "DescTools"))

# Load required packages
library(tidyverse)   # For data manipulation and visualization
library(rstatix)     # For statistical analysis
library(ggstatsplot) # For statistical visualization
library(ggsignif)    # For significance annotations
library(corrplot)    # For visualization of relationships
library(effectsize)  # For effect size calculations
library(DescTools)   # For additional statistical tools
```

Data Preparation

Let’s prepare our data using tidyverse functions:

```
# Prepare data for analysis
analysis_data %
  mutate(
    sex = factor(sex, levels = c(1, 2), labels = c("Male", "Female")),
    smoker = factor(smoker, levels = c(1, 2), labels = c("Smoker", "Non-smoker"))
  )

# Display the structure of our prepared data
glimpse(analysis_data)
```

Enhanced Visualization of Observed vs Expected

The `ggstatsplot` package creates informative visualizations that include statistical results:

```
# Create enhanced bar plot with statistical results
ggbarstats(
  data = analysis_data,
  x = smoker,
  title = "Observed vs Expected Smoking Rates",
  xlab = "Smoking Status",
  ylab = "Count",
  results.subtitle = TRUE,
  subtitle = "Testing against expected 20% smoking rate",
  type = "parametric",
  paired = FALSE,
  null.value = 0.2
)
```

Comprehensive Statistical Analysis

The `rstatix` package provides clear statistical output:

```
# Calculate observed frequencies
obs_freq %
  count(smoker) %>%
  mutate(
    prop = n / sum(n),
    expected_prop = c(0.2, 0.8),
    expected_n = sum(n) * expected_prop,
    contrib = (n - expected_n)^2 / expected_n
  )

# Display comprehensive results
obs_freq %>%
  knitr::kable(digits = 3)

# Perform test with effect size
chisq_result %
  add_significance()

# Calculate effect size
effect <- cramers_v(table(analysis_data$smoker))

# Display results
list(
  "Chi-square test results" = chisq_result,
  "Effect size (Cramer's V)" = effect
)
```

Advantages of Modern Packages

1. **Enhanced Visualization**
    - One-click visualization of observed vs expected frequencies
    - Automated proportion plots with error bars
    - Clear representation of deviations from expected values
2. **Clearer Output**
    - Formatted tables comparing observed and expected frequencies
    - Clear presentation of test statistics
    - Automatic computation of standardized residuals
3. **Additional Tools**
    - Multiple effect size measures (Cohen’s w, Cramer’s V)
    - Power analysis capabilities
    - Easy comparison with multiple expected distributions
4. **Better Integration**
    - Simple workflow from data to visualization
    - Easy export of results for reporting
    - Consistent syntax across analyses

Best Practices for Reporting

1. Report:
    - Sample size and degrees of freedom
    - Expected proportions and their source/justification
    - Chi-square statistic and exact p-value
    - Effect size (typically Cohen’s w)
    - Visual comparison of observed vs expected frequencies
2. Include:
    - Clear statement of the null hypothesis
    - Table of observed and expected frequencies
    - Standardized residuals for each category
    - Power analysis if the result is non-significant
3. Consider:
    - Whether categories should be combined
    - If sample size is adequate
    - Alternative tests for small samples

5.3 Step G3: Chi-Square Test for Independence (Smoking and Gender)

Now let's examine whether there's a relationship between smoking and gender. This test helps us determine if two categorical variables are independent of each other.

First, let's create a cross-tabulation and examine our data:

```r
# Create cross-tabulation
cross_tab <- table(survey_data_full$sex, survey_data_full$smoker)

# Add row and column names for clarity
dimnames(cross_tab) <- list(
  Gender = c("Male", "Female"),
  Smoking = c("Smoker", "Non-smoker")
)

# Display the cross-tabulation
print("Cross-tabulation of Gender and Smoking:")
print(cross_tab)

# Calculate and display percentages by gender
prop_table <- prop.table(cross_tab, margin = 1) * 100
print("\nPercentages within each gender:")
print(round(prop_table, 1))
```

Before performing the Chi-Square test of independence, we should check if our data meets the assumptions:

```r
# Calculate expected frequencies
expected <- chisq.test(cross_tab)$expected
print("\nExpected frequencies:")
print(round(expected, 2))

# Check if any expected frequencies are less than 5
min_expected <- min(expected)
print("\nMinimum expected frequency:", min_expected)
if(min_expected < 5) {
  print("Warning: Some expected frequencies are less than 5!")
}
```

Now we can perform the Chi-Square test of independence:

```r
# Perform Chi-Square test with continuity correction (for 2x2 tables)
chi_result <- chisq.test(cross_tab, correct = TRUE)
print("\nChi-Square Test Results:")
print(chi_result)

# Calculate effect size (Cramer's V)
n <- sum(cross_tab)
cramer_v <- sqrt(chi_result$statistic / (n * (min(dim(cross_tab)) - 1)))
print("\nCramer's V:", round(cramer_v, 3))
```

To understand effect sizes in a 2x2 table, we can interpret Cramer's V as follows:

- 0.1: Small effect
- 0.3: Medium effect
- 0.5: Large effect

Standardized residuals help us understand which cells contribute most to any significant results:

```r
# Calculate adjusted standardized residuals
stdres <- chisq.test(cross_tab)$stdres
dimnames(stdres) <- dimnames(cross_tab)
print("\nAdjusted standardized residuals:")
print(round(stdres, 2))
```

Let's create some helpful visualizations:

```r
# Create barplot of smoking rates by gender
barplot(prop_table,
        beside = TRUE,
        main = "Smoking Status by Gender",
        xlab = "Smoking Status",
        ylab = "Percentage",
        col = c("lightblue", "pink"),
        legend.text = TRUE)
```

Chi-Square Test for Independence with R Packages

Modern R packages provide enhanced capabilities for conducting and visualizing Chi-Square analyses. We'll use several powerful packages that work together to create comprehensive and intuitive analyses:

```r
# Install packages if needed
install.packages(c("tidyverse", "rstatix", "ggstatsplot", "ggsignif",
                   "corrplot", "effectsize", "DescTools"))

# Load required packages
library(tidyverse)   # For data manipulation and visualization
library(rstatix)     # For statistical analysis
library(ggstatsplot) # For statistical visualization
library(ggsignif)    # For significance annotations
library(corrplot)    # For visualization of relationships
library(effectsize)  # For effect size calculations
library(DescTools)   # For additional statistical tools
```

Data Preparation

Let's prepare our data using tidyverse functions:

```r
# Prepare data for analysis
analysis_data %
  mutate(
    sex = factor(sex, levels = c(1, 2), labels = c("Male", "Female")),
    smoker = factor(smoker, levels = c(1, 2), labels = c("Smoker", "Non-smoker"))
  )

# Display the structure of our prepared data
glimpse(analysis_data)
```

Enhanced Cross-tabulation Visualization

```r
# Create an enhanced visualization of the relationship
ggbarstats(
  data = analysis_data,
  x = sex,
  y = smoker,
  title = "Relationship between Gender and Smoking Status",
  xlab = "Gender",
  ylab = "Proportion",
  label = "percentage",
  results.subtitle = TRUE
)
```

Comprehensive Analysis

```r
# Perform analysis with rstatix
independence_test %
  tabyl(sex, smoker) %>%
  chisq_test() %>%
  add_significance()

# Calculate effect size
independence_effect %
  tabyl(sex, smoker) %>%
  cramer_v()

# Display results
list(
  "Chi-square test results" = independence_test,
  "Effect size (Cramer's V)" = independence_effect
)
```

Post-hoc Analysis

If we find significant associations, we can examine standardized residuals:

```r
# Calculate and visualize standardized residuals
residuals_analysis %
  tabyl(sex, smoker) %>%
  chisq_test() %>%
  subset_residuals()

# Create heatmap of residuals
ggplot(residuals_analysis, aes(sex, smoker, fill = residual)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white") +
  geom_text(aes(label = round(residual, 2))) +
  theme_minimal() +
  labs(title = "Standardized Residuals",
       fill = "Residual")
```

Effect Size Visualization

The `effectsize` package helps understand practical significance:

```r
# Calculate and visualize effect sizes
effect_size <- effectsize::cramers_v(
  table(analysis_data$sex, analysis_data$smoker)
)

# Create effect size plot
plot(effect_size) +
  theme_minimal() +
  labs(title = "Effect Size (Cramer's V) with Confidence Interval")
```

Creating a Complete Report

We can create a comprehensive analysis function:

```
analyze_categorical %
        count(!!sym(var1)) %>%
        mutate(proportion = n/sum(n)),

      visualization = ggbarstats(
        data = data,
        x = !!sym(var1)
      ),

      test = data %>%
        pull(!!sym(var1)) %>%
        table() %>%
        chisq_test() %>%
        add_significance(),

      effect = data %>%
        pull(!!sym(var1)) %>%
        table() %>%
        cramers_v()
    )
  } else {
    # Independence test
    result %
        tabyl(!!sym(var1), !!sym(var2)),

      visualization = ggbarstats(
        data = data,
        x = !!sym(var1),
        y = !!sym(var2)
      ),

      test = data %>%
        tabyl(!!sym(var1), !!sym(var2)) %>%
        chisq_test() %>%
        add_significance(),

      effect = data %>%
        tabyl(!!sym(var1), !!sym(var2)) %>%
        cramer_v()
    )
  }

  return(result)
}

# Example usage
smoking_analysis <- analyze_categorical(
  data = analysis_data,
  var1 = "sex",
  var2 = "smoker"
)

# Display results
smoking_analysis
```

Advantages of Modern Packages

1. **Enhanced Visualization**
    - Mosaic plots with statistical annotations
    - Heat maps of associations
    - Interactive contingency tables

2. **Clearer Output**
   - Well-formatted contingency tables
   - Clear presentation of cell percentages
   - Highlighted significant associations
3. **Additional Tools**
   - Multiple association measures (Phi, Cramer's V, Lambda)
   - Post-hoc cell-wise tests
   - Assumption checking for expected frequencies
4. **Better Integration**
   - Streamlined workflow for multiple variables
   - Easy integration with other categorical analyses
   - Consistent reporting formats

Best Practices for Reporting

1. Report:
   - Complete contingency table with row and column totals
   - Chi-square statistic and exact p-value
   - Effect size (Cramer's V or Phi for 2x2 tables)
   - Degrees of freedom
   - Smallest expected frequency
2. Include:
   - Cross-tabulation with percentages
   - Standardized residuals for significant results
   - Visual representation of the relationship
   - Confidence intervals for proportions
3. Consider:
   - Cell size assumptions
   - Need for Fisher's exact test
   - Multiple comparison corrections
   - Direction and strength of associations

5.4 Step G4: Writing Up Results

Here's how to write up the results of a **goodness of fit test** in a formal report:

> "A chi-square goodness of fit test was performed to determine whether the sample's smoking rate differed from the expected population rate of 20%. The test revealed [insert significance statement based on results]. The observed rate was [X]% compared to the expected 20%."

And how to write up the results of an **independence test** in a formal report:

> "A chi-square test of independence was conducted to examine the relationship between gender and smoking status. All expected cell frequencies were greater than 5. The test revealed [insert significance statement based on results]. Among males, [X]% were smokers compared to [Y]% of females."

6 Exercise H: Exploring the Staff Survey Data

6.1 Step H1: Setup

You can continue working in the same project or working directory that you created in the previous lab and have been working in for the past exercises. You should, however, do the following:

1. Create a new **R script**: Go to **File > New File > R Script**.
2. Save the script in your `scripts` folder with an appropriate name, e.g., `Lab2_Exercise_H.R`.

ℹ Note

Try to start with the empty script that you created above (`Lab2_Exercise_H.R`) and work through this exerise before looking at the solutions. But you can also download the R script for Exercise H and save the downloaded file in the `scripts` folder.

Now you are ready to begin your work in R and continue with Step H2!

6.2 Step H2: Loading the Dataset

Please download the dataset and download the code book and save both in the `data` folder within your project folder or working directory.

```r
# Load the CSV file stored in the "data" folder
staff_survey_data <- read.csv("data/lab2-staff-survey.csv")
```

You can easily explore and check the basic structure of your data and get a summary:

```r
# View the first few rows using head()
head(staff_survey_data)

# Examine the structure
str(staff_survey_data)

# Get basic summary statistics
summary(staff_survey_data)
```

In line with the code book, we want to assign labels to the variables in the `staff_survey_data` data frame using **factors** with labeled levels. This method ensures the data remains categorical but with human-readable labels for easier interpretation and analysis.

```r
# First, let's create vectors for our common level labels since they're reused often
extent_levels <- c("not at all", "to a slight extent", "to a moderate extent",
                  "to a great extent", "to a very great extent")

importance_levels <- c("not important", "slightly important", "moderately important",
                      "very important", "extremely important")

# Create age groups labels
staff_survey_data$age <- factor(staff_survey_data$age,
                               levels = 1:5,
                               labels = c("under 20", "21 to 30", "31 to 40",
                                        "41 to 50", "over 50"))

# Employment status
staff_survey_data$employstatus <- factor(staff_survey_data$employstatus,
                                        levels = 1:2,
                                        labels = c("permanent", "casual"))

# Now let's handle the repeated patterns for Q1a through Q10a
# We'll use a loop to avoid repetitive code
for(i in 1:10) {
  # Convert "extent" questions (Q1a through Q10a)
  staff_survey_data[[paste0("Q", i, "a")]] <- factor(
    staff_survey_data[[paste0("Q", i, "a")]],
    levels = 1:5,
    labels = extent_levels
  )

  # Convert "importance" questions (Q1imp through Q10imp)
  staff_survey_data[[paste0("Q", i, "imp")]] <- factor(
    staff_survey_data[[paste0("Q", i, "imp")]],
    levels = 1:5,
    labels = importance_levels
  )
}

# Finally, convert the recommend variable
staff_survey_data$recommend <- factor(staff_survey_data$recommend,
                                    levels = c(0, 1),
                                    labels = c("no", "yes"))
```

```r
# Let's add a verification step to check our work
# This will print the levels of each variable to confirm they're correctly labeled
verify_factors <- function(data) {
  for(col in names(data)) {
    if(is.factor(data[[col]])) {
      cat("\nLevels for", col, ":\n")
      print(levels(data[[col]]))
    }
  }
}

# Run the verification
verify_factors(staff_survey_data)
```

The `summary()` function can also be used to confirm that the labels have been applied correctly to the variables.

```r
# Verify changes by printing a summary
summary(staff_survey_data)
```

Try working through the folloing steps and answer the questions before looking at the solutions.

6.3 Step H3: Frequency Tables

Create frequency tables of the demographic variables `city`, `service`, and `employstatus`.

Note: The solution is provided in the callout notes below. We'll explore the `staff_survey_data` using only base R functions (orange callout note) and also provide solutions using R packages (in the usual green callout notes). While modern packages offer many conveniences, understanding base R approaches provides a solid foundation and helps us appreciate the underlying statistical processes.

Solution to Step H3

Let's begin by examining the distribution of our key demographic variables. Base R provides several functions for creating and formatting frequency tables:

```r
# Create frequency table for city
city_table <- table(staff_survey_data$city)
city_props <- prop.table(city_table) * 100

# Combine counts and percentages
city_summary <- cbind(
  Frequency = as.vector(city_table),
  Percentage = as.vector(city_props)
)
rownames(city_summary) <- names(city_table)

# Display the results with formatting
print("Distribution of Staff by City:")
print(round(city_summary, 1))

# Create frequency table for employment status
status_table <- table(staff_survey_data$employstatus)
status_props <- prop.table(status_table) * 100

status_summary <- cbind(
  Frequency = as.vector(status_table),
  Percentage = as.vector(status_props)
)
rownames(status_summary) <- names(status_table)

print("\nDistribution of Employment Status:")
print(round(status_summary, 1))

# Create frequency table for service
# First, let's create reasonable bins for years of service
service_breaks <- seq(0, max(staff_survey_data$service, na.rm = TRUE) + 5, by = 5)
service_cats <- cut(staff_survey_data$service,
                   breaks = service_breaks,
                   include.lowest = TRUE)

service_table <- table(service_cats)
service_props <- prop.table(service_table) * 100

service_summary <- cbind(
  Frequency = as.vector(service_table),
  Percentage = as.vector(service_props)
)
rownames(service_summary) <- names(service_table)

print("\nDistribution of Years of Service:")
print(round(service_summary, 1))
```

When interpreting these results, consider:

- Look for notable imbalances in group sizes
- Consider whether distributions match expectations

Alternative Solution to H3 with R Packages

We can also use R packages that work together to create comprehensive and intuitive analyses:

```r
# Install packages if needed
install.packages(c("tidyverse", "knitr", "kableExtra"))

# Load required packages
library(tidyverse)  # For data manipulation and visualization
library(knitr)      # For nice tables
library(kableExtra) # For enhanced table formatting
```

First, we'll create clear, informative frequency tables for our key demographic variables:

```r
# Function to create a nicely formatted frequency table
create_freq_table <- function(data, variable) {
  data %>%
    count({{variable}}) %>%
    mutate(Percentage = n/sum(n) * 100) %>%
    kable(digits = 1,
          col.names = c("Category", "Count", "Percentage"),
          caption = paste("Distribution of", deparse(substitute(variable)))) %>%
    kable_styling(bootstrap_options = c("striped", "hover"))
}

# Create tables for each demographic variable
create_freq_table(staff_survey_data, city)
create_freq_table(staff_survey_data, service)
create_freq_table(staff_survey_data, employstatus)
```

6.4 Step H4: Hisogram

Create a histogram of the demographic variable `service` and look for outliers.

🔥 Solution to Step H4

We'll create a histogram to understand the distribution of service years and identify potential outliers:

```r
# Create histogram
hist(staff_survey_data$service,
     main = "Distribution of Years of Service",
     xlab = "Years of Service",
     ylab = "Frequency",
     breaks = 20,
     col = "lightblue",
     border = "white")

# Add a box plot for outlier detection
boxplot(staff_survey_data$service,
        horizontal = TRUE,
        main = "Box Plot of Years of Service",
        xlab = "Years of Service",
        col = "lightblue")

# Reset plotting parameters
par(mfrow = c(1, 1))

# Calculate summary statistics
service_summary_stats <- summary(staff_survey_data$service)
service_sd <- sd(staff_survey_data$service, na.rm = TRUE)

# Print summary statistics
print("\nSummary Statistics for Years of Service:")
print(service_summary_stats)
print(paste("Standard Deviation:", round(service_sd, 2)))

# Identify potential outliers using the 1.5 * IQR rule
Q1 <- quantile(staff_survey_data$service, 0.25, na.rm = TRUE)
Q3 <- quantile(staff_survey_data$service, 0.75, na.rm = TRUE)
IQR <- Q3 - Q1
outlier_threshold_upper <- Q3 + 1.5 * IQR
outlier_threshold_lower <- Q1 - 1.5 * IQR

outliers <- staff_survey_data$service[staff_survey_data$service > outlier_threshold_upper |
                                    staff_survey_data$service < outlier_threshold_lower]

if(length(outliers) > 0) {
  print("\nPotential outliers identified:")
  print(sort(outliers))
}
```

When interpreting these results, consider:

- Note the shape of the distribution
- Identify any unusual patterns or gaps
- Consider whether outliers represent errors or true values

Alternative Solution to H4 with R Packages

We can also use R packages that work together to create comprehensive and intuitive analyses:

```r
# Install packages if needed
install.packages(c("tidyverse", "knitr", "kableExtra"))

# Load required packages
library(tidyverse)  # For data manipulation and visualization
library(knitr)      # For nice tables
library(kableExtra) # For enhanced table formatting
```

Let’s create a detailed histogram to understand the distribution of years of service and identify any potential outliers:

```r
# Create histogram with density curve
ggplot(staff_survey_data, aes(x = service)) +
  geom_histogram(aes(y = ..density..),
                 bins = 30,
                 fill = "lightblue",
                 color = "black") +
  geom_density(color = "red") +
  theme_minimal() +
  labs(x = "Years of Service",
       y = "Density",
       title = "Distribution of Years of Service",
       subtitle = "With density curve overlay") +
  # Add boxplot for outlier visualization
  geom_boxplot(aes(y = -0.02), width = 0.1)

# Calculate summary statistics for service
service_summary <- staff_survey_data %>%
  summarize(
    Mean = mean(service, na.rm = TRUE),
    Median = median(service, na.rm = TRUE),
    SD = sd(service, na.rm = TRUE),
    Q1 = quantile(service, 0.25, na.rm = TRUE),
    Q3 = quantile(service, 0.75, na.rm = TRUE),
    IQR = IQR(service, na.rm = TRUE),
    Min = min(service, na.rm = TRUE),
    Max = max(service, na.rm = TRUE)
  )

# Print summary statistics
kable(service_summary, digits = 1,
      caption = "Summary Statistics for Years of Service") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))

# Identify potential outliers
outliers <- staff_survey_data %>%
  filter(service > (service_summary$Q3 + 1.5 * service_summary$IQR) |
         service < (service_summary$Q1 - 1.5 * service_summary$IQR))
```

Looking at the distribution of years of service, we can observe:

- The shape of the distribution (whether it’s symmetric or skewed)
- Any unusual patterns or gaps
- Potential outliers, which we’ve identified using the 1.5 × IQR rule

6.5 Step H5: Cross-Tabulation

Cross-tabulate and study the age groups (`agerecode`) against `employstatus`.

Solution to Step H5

Let’s examine the relationship between age groups and employment status using a cross-tabulation:

```r
# Create cross-tabulation
age_employ_table <- table(staff_survey_data$agerecode,
                         staff_survey_data$employstatus)

# Calculate row percentages
age_employ_props <- prop.table(age_employ_table, margin = 1) * 100

# Print the results
print("Counts by Age Group and Employment Status:")
print(age_employ_table)
print("\nPercentages within Age Groups:")
print(round(age_employ_props, 1))

# Create a visual representation using base R
# Set up the plotting area
barplot(t(age_employ_props),
        beside = TRUE,
        col = c("lightblue", "lightgreen"),
        main = "Employment Status by Age Group",
        xlab = "Age Group",
        ylab = "Percentage",
        legend.text = colnames(age_employ_props))

# Perform chi-square test of independence
chi_sq_test <- chisq.test(age_employ_table)
print("\nChi-square test of independence:")
print(chi_sq_test)
```

When interpreting these results, consider:

- Look for patterns in employment type across age groups
- Consider the chi-square test results
- Think about implications for workforce planning

Alternative Solution to H5 with R Packages

We can also use R packages that work together to create comprehensive and intuitive analyses:

```r
# Install packages if needed
install.packages(c("tidyverse", "knitr", "kableExtra"))

# Load required packages
library(tidyverse)  # For data manipulation and visualization
library(knitr)      # For nice tables
library(kableExtra) # For enhanced table formatting
```

Let's examine the relationship between age groups and employment status:

```r
# Create cross-tabulation
age_employ_table <- table(staff_survey_data$agerecode,
                         staff_survey_data$employstatus)

# Convert to percentages and format nicely
age_employ_props <- prop.table(age_employ_table, margin = 1) * 100

# Combine counts and percentages in a nice table
age_employ_combined <- cbind(
  as.data.frame.matrix(age_employ_table),
  as.data.frame.matrix(age_employ_props)
) %>%
  setNames(c("Permanent (n)", "Casual (n)",
             "Permanent (%)", "Casual (%)"))

kable(age_employ_combined,
      digits = 1,
      caption = "Age Groups by Employment Status") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))

# Create a visualization
ggplot(staff_survey_data,
       aes(x = agerecode, fill = employstatus)) +
  geom_bar(position = "fill") +
  theme_minimal() +
  labs(x = "Age Group",
       y = "Proportion",
       fill = "Employment Status",
       title = "Employment Status Distribution by Age Group") +
  coord_flip()
```

6.6 Step H6: Total Staff Satisfaction

How do you think total staff satisfaction (`totsatis`) is calculated? Is there a difference in Total staff satisfaction among the two staff statuses? (Hint: t-test)

Solution to Step H6

To understand how total staff satisfaction (`totsatis`) might be calculated, let's examine its relationship with individual satisfaction items:

```r
# Identify satisfaction-related variables (Q*a)
satisfaction_vars <- grep("Q.*a$", names(staff_survey_data), value = TRUE)

# Calculate correlations with total satisfaction
correlations <- sapply(staff_survey_data[satisfaction_vars],
                      function(x) cor(x, staff_survey_data$totsatis,
                                      use = "complete.obs"))

# Print correlations
print("Correlations with Total Satisfaction:")
print(round(correlations, 3))

# Create scatterplot matrix of selected variables
pairs(staff_survey_data[c(satisfaction_vars[1:5], "totsatis")],
      main = "Relationships between Satisfaction Measures")
```

Let's examine whether there are differences in total satisfaction between permanent and casual staff:

```
# Calculate descriptive statistics by group
tapply(staff_survey_data$totsatis,
       staff_survey_data$employstatus,
       function(x) c(n = length(x),
                    mean = mean(x, na.rm = TRUE),
                    sd = sd(x, na.rm = TRUE)))

# Create box plot for visual comparison
boxplot(totsatis ~ employstatus,
        data = staff_survey_data,
        main = "Total Satisfaction by Employment Status",
        xlab = "Employment Status",
        ylab = "Total Satisfaction Score",
        col = "lightblue")

# Add individual points for better visualization
stripchart(totsatis ~ employstatus,
           data = staff_survey_data,
           vertical = TRUE,
           method = "jitter",
           add = TRUE,
           pch = 20,
           col = "darkgray")

# Perform t-test
satisfaction_ttest <- t.test(totsatis ~ employstatus,
                             data = staff_survey_data)

# Print t-test results
print("\nt-test Results:")
print(satisfaction_ttest)

# Calculate effect size (Cohen's d)
group1 <- staff_survey_data$totsatis[staff_survey_data$employstatus == "permanent"]
group2 <- staff_survey_data$totsatis[staff_survey_data$employstatus == "casual"]

cohens_d <- function(x, y) {
  nx <- length(x)
  ny <- length(y)
  pooled_sd <- sqrt(((nx-1)*var(x) + (ny-1)*var(y))/(nx+ny-2))
  abs(mean(x) - mean(y))/pooled_sd
}

effect_size <- cohens_d(group1, group2)
print(paste("\nEffect size (Cohen's d):", round(effect_size, 3)))
```

When interpreting these results, consider:

- Consider both statistical significance (p-value) and practical significance (effect size)
- Look at the distribution of scores within each group
- Think about potential confounding variables

Alternative Solution to H6 with R Packages

We can also use R packages that work together to create comprehensive and intuitive analyses:

```
# Install packages if needed
install.packages(c("tidyverse", "knitr", "kableExtra"))

# Load required packages
library(tidyverse)  # For data manipulation and visualization
library(knitr)      # For nice tables
library(kableExtra) # For enhanced table formatting
```

The total staff satisfaction score (`totsatis`) is likely calculated as a composite measure of various satisfaction-related questions in the survey. Let's examine its components and distribution:

```
# Examine the structure of satisfaction-related variables
satisfaction_vars <- names(staff_survey_data)[grep("Q.*a$", names(staff_survey_data))]

# Look at correlations between these variables and totsatis
satisfaction_correlations <- staff_survey_data %>%
  select(all_of(c(satisfaction_vars, "totsatis"))) %>%
  cor(use = "complete.obs")

# Print correlations with total satisfaction
kable(satisfaction_correlations["totsatis", ],
      digits = 3,
      caption = "Correlations with Total Satisfaction") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

To examine whether there are differences in total satisfaction between permanent and casual staff, we'll conduct a t-test:

```r
# First, let's look at descriptive statistics
satisfaction_by_status <- staff_survey_data %>%
  group_by(employstatus) %>%
  summarise(
    n = n(),
    mean = mean(totsatis, na.rm = TRUE),
    sd = sd(totsatis, na.rm = TRUE),
    se = sd/sqrt(n)
  )

# Print descriptive statistics
kable(satisfaction_by_status,
      digits = 2,
      caption = "Satisfaction Scores by Employment Status") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))

# Create visualization
ggplot(staff_survey_data, aes(x = employstatus, y = totsatis)) +
  geom_boxplot(fill = "lightblue") +
  geom_jitter(width = 0.2, alpha = 0.2) +
  theme_minimal() +
  labs(x = "Employment Status",
       y = "Total Satisfaction Score",
       title = "Distribution of Satisfaction Scores by Employment Status")

# Perform t-test
satisfaction_ttest <- t.test(totsatis ~ employstatus,
                           data = staff_survey_data)

# Calculate effect size (Cohen's d)
cohens_d <- function(x, y) {
  nx <- length(x)
  ny <- length(y)
  pooled_sd <- sqrt(((nx-1)*var(x) + (ny-1)*var(y))/(nx+ny-2))
  abs(mean(x) - mean(y))/pooled_sd
}

effect_size <- with(staff_survey_data,
                   cohens_d(totsatis[employstatus == "permanent"],
                           totsatis[employstatus == "casual"]))

# Print t-test results
print(satisfaction_ttest)
cat("\nEffect size (Cohen's d):", round(effect_size, 3))
```

7 Exercise I: Logistic Regression

7.1 Step I1: Setup

You can continue working in the same project or working directory that you created in the previous lab and have been working in for the past exercises. You should, however, do the following:

1. Create a new **R script**: Go to **File** > **New File** > **R Script**.
2. Save the script in your `scripts` folder with an appropriate name, e.g., `Lab2_Exercise_I.R`.

Note

You can either work through the following steps and copy/paste the respective code into the Lab_Exercise_I.R file that you will create in Step I1 or download the R script for Exercise H and follow the instructions below and save the downloaded file in the scripts folder that you will create.

Now you are ready to begin your work in R and continue with Step I2!

7.2 Step I2: Loading the Dataset

Please download the dataset and save the file in the data folder within your project folder or working directory.

```
# Load the CSV file stored in the "data" folder
sleep_survey_data <- read.csv("data/lab2-sleep-survey.csv")
```

You can easily explore and check the basic structure of your data and get a summary:

```
# View the first few rows using head()
head(sleep_survey_data)

# Examine the structure
str(sleep_survey_data)

# Get basic summary statistics
summary(sleep_survey_data)
```

We want to assign labels to some of the categorical variables in the sleep_survey_data data frame using **factors** with labeled levels. This method ensures the data remains categorical but with human-readable labels for easier interpretation and analysis.

```
# Let's start by creating our factor labels in a clear, organized way.
# When working with binary variables (0/1), it's especially important to be
# consistent with our labeling approach.

# First, let's handle the binary (0/1) variables
# We'll create these first since they share the same structure
sleep_survey_data$sex <- factor(sleep_survey_data$sex,
                               levels = c(0, 1),
                               labels = c("female", "male"))

sleep_survey_data$probsleep <- factor(sleep_survey_data$probsleep,
                                     levels = c(0, 1),
                                     labels = c("no", "yes"))

sleep_survey_data$fallsleep <- factor(sleep_survey_data$fallsleep,
                                     levels = c(0, 1),
                                     labels = c("no", "yes"))

sleep_survey_data$staysleep <- factor(sleep_survey_data$staysleep,
                                     levels = c(0, 1),
                                     labels = c("no", "yes"))

# Now let's handle the categorical variables with multiple levels
# For these, we'll use more descriptive labels
sleep_survey_data$marital <- factor(sleep_survey_data$marital,
                                   levels = 1:4,
                                   labels = c("single",
                                            "married/defacto",
                                            "divorced",
                                            "widowed"))

sleep_survey_data$edlevel <- factor(sleep_survey_data$edlevel,
                                    levels = 1:5,
                                    labels = c("primary school",
                                             "secondary school",
                                             "trade training",
                                             "undergraduate degree",
                                             "postgraduate degree"))

# Let's add a verification step to make sure our conversions worked correctly
# This function will print the levels of each factor variable we created
verify_factors <- function(data) {
    cat("Checking factor levels for all converted variables:\n\n")

    # List of variables we converted
    factor_vars <- c("sex", "probsleep", "fallsleep", "staysleep",
                     "marital", "edlevel")

    for(var in factor_vars) {
        cat(paste0("Levels for ", var, ":\n"))
        print(levels(data[[var]]))
        cat("\n")
    }
}

# Run the verification
verify_factors(sleep_survey_data)
```

7.3 Step I3: Logistic Regression

7.3.1 What is Logistic Regression?

Logistic regression is a statistical method used when we want to predict a categorical outcome (like yes/no, pass/fail) based on one or more predictor variables. Think of it as answering questions like "What factors influence whether someone has sleep problems?" or "What characteristics predict if a student will pass an exam?"

We use logistic regression when:

- Our outcome variable is categorical (usually binary).
- We have multiple predictor variables (can be continuous or categorical).
- We want to understand both prediction and relationships.

In our sleep study, we're trying to predict whether someone has sleep problems (yes/no) based on various characteristics and behaviors.

Understanding the Math (In Simple Terms)

While regular regression predicts actual values (like height or weight), logistic regression predicts the probability of belonging to a category (like having sleep problems). It uses a special S-shaped curve called a logistic function that keeps predictions between 0 and 1 (or 0% to 100% probability).

7.3.2 Initial Data Preparation

Let's prepare our data and examine our sample:

```
# First, let's check our sample size and missing data
initial_sample <- nrow(sleep_survey_data)
complete_cases <- sum(complete.cases(sleep_survey_data[
  c("probsleep", "sex", "age", "fallsleep", "staysleep", "hrswknight")
]))

# Print sample information
cat("Total cases:", initial_sample, "\n")
cat("Complete cases:", complete_cases, "\n")
cat("Missing cases:", initial_sample - complete_cases, "\n")
```

```r
# Check coding of categorical variables
cat("\nCoding of categorical variables:\n")
cat("\nSleep Problems (probsleep):\n")
print(table(sleep_survey_data$probsleep))

cat("\nSex:\n")
print(table(sleep_survey_data$sex))

cat("\nTrouble Falling Asleep (fallsleep):\n")
print(table(sleep_survey_data$fallsleep))

cat("\nTrouble Staying Asleep (staysleep):\n")
print(table(sleep_survey_data$staysleep))
```

7.3.3 Building the Logistic Regression Model

Now let's create our model. We'll use base R's `glm` (Generalized Linear Model) function with the logit link:

```r
# Create the logistic regression model
sleep_model <- glm(probsleep ~ sex + age + fallsleep + staysleep + hrswknight,
                   family = binomial(link = "logit"),
                   data = sleep_survey_data)

# Display the summary of our model
summary_results <- summary(sleep_model)
print(summary_results)

# Calculate odds ratios and confidence intervals
odds_ratios <- exp(coef(sleep_model))
conf_int <- exp(confint(sleep_model))

# Combine results in a readable format
results_table <- cbind(
  Estimate = round(coef(sleep_model), 3),
  "Odds Ratio" = round(odds_ratios, 3),
  "CI Lower" = round(conf_int[,1], 3),
  "CI Upper" = round(conf_int[,2], 3),
  "p-value" = round(summary_results$coefficients[,4], 3)
)

# Print results
cat("\nDetailed Results:\n")
print(results_table)

# Calculate model fit statistics
null_deviance <- sleep_model$null.deviance
model_deviance <- sleep_model$deviance
pseudo_r2 <- 1 - (model_deviance / null_deviance)

cat("\nModel Fit Statistics:\n")
cat("Null deviance:", round(null_deviance, 2), "\n")
cat("Model deviance:", round(model_deviance, 2), "\n")
cat("McFadden's Pseudo R²:", round(pseudo_r2, 3), "\n")
```

7.4 Step I4: Interpreting the Results

Let's break down what these results tell us.

7.4.1 Model Convergence and Overall Fit

Our model successfully converged, which means it found a stable solution. The reduction in deviance from the null model (model with no predictors) to our final model indicates how much our predictors help explain sleep problems.

7.4.2 Individual Predictors

Let's examine each predictor's contribution:

```r
# Create a function to interpret odds ratios
interpret_or <- function(or, p_value) {
  direction <- if(or > 1) "increase" else "decrease"
  magnitude <- abs(1 - or) * 100
  significance <- if(p_value < 0.05) "significant" else "not significant"

  sprintf("%.1f%% %s in odds (%s)", magnitude, direction, significance)
}

# Print interpretations for each predictor
cat("Interpretation of Effects:\n\n")
for(i in 2:nrow(results_table)) {
  var_name <- rownames(results_table)[i]
  or <- results_table[i, "Odds Ratio"]
  p_val <- results_table[i, "p-value"]

  cat(var_name, ":", interpret_or(or, p_val), "\n")
}
```

7.4.3 Key Findings

Our analysis reveals that:

- **Staying Asleep** ($p < 0.001$): This is our strongest predictor. People who have trouble staying asleep are significantly more likely to report general sleep problems.
- **Falling Asleep** ($p = 0.035$): Difficulty falling asleep is also a significant predictor, though not as strong as - staying asleep.
- **Hours of Sleep** ($p = 0.007$): Each additional hour of sleep is associated with a decrease in the odds of reporting sleep problems.
- **Age** and **Sex**: Neither demographic variable significantly predicts sleep problems ($p > 0.05$).

7.4.4 Model Accuracy

We can examine how well our model predicts sleep problems:

```r
# Calculate predicted probabilities
predicted_probs <- predict(sleep_model, type = "response")
predicted_class <- ifelse(predicted_probs > 0.5, 1, 0)
actual_class <- as.numeric(sleep_survey_data$probsleep) - 1

# Create confusion matrix
conf_matrix <- table(Predicted = predicted_class, Actual = actual_class)
print("\nConfusion Matrix:")
print(conf_matrix)

# Calculate accuracy metrics
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
sensitivity <- conf_matrix[2,2] / sum(conf_matrix[,2])
specificity <- conf_matrix[1,1] / sum(conf_matrix[,1])

cat("\nModel Performance Metrics:\n")
cat("Accuracy:", round(accuracy, 3), "\n")
cat("Sensitivity:", round(sensitivity, 3), "\n")
cat("Specificity:", round(specificity, 3), "\n")
```

7.4.5 Practical Implications

Our analysis suggests several important practical implications:

- **Sleep Maintenance**: The strong effect of staying asleep suggests that interventions focused on sleep maintenance might be most beneficial.
- **Sleep Duration**: The significant effect of hours of sleep indicates that strategies to increase sleep duration could help reduce sleep problems.
- **Universal Impact**: The lack of significant age and sex effects suggests that sleep problems affect people regardless of these demographic factors.

7.4.6 Limitations and Considerations

When interpreting these results, consider:

- Self-reported data may be subject to recall bias.
- The binary nature of our outcome variable might oversimplify complex sleep issues.
- There might be other important predictors we haven’t included in the model.

Logistic Regression with R Packages

Modern R packages provide enhanced capabilities for conducting and interpreting logistic regression analyses. We’ll use several packages that work together to create a comprehensive analysis while making the process more intuitive and the results more interpretable.

```r
# Install packages if needed
install.packages(c("tidyverse", "broom", "performance", "sjPlot", "marginaleffects"))

# Load required packages
library(tidyverse)    # For data manipulation and visualization
library(broom)        # For tidying model output
library(performance)  # For model diagnostics and performance metrics
library(sjPlot)      # For plotting odds ratios and model summaries
library(marginaleffects)  # For calculating and plotting marginal effects
```

Understanding our Data

The `tidyverse` package provides powerful tools for initial data exploration:

```r
# Create our analysis dataset
sleep_data <- sleep_survey_data %>%
  select(probsleep, sex, age, fallsleep, staysleep, hrswknight) %>%
  # Convert to factors with meaningful labels if not already done
  mutate(across(c(probsleep, sex, fallsleep, staysleep),
                ~factor(., levels = c(0, 1),
                        labels = c("No", "Yes"))))

# Examine our data structure
glimpse(sleep_data)

# Check missing data patterns
missing_pattern <- sleep_data %>%
  summarise(across(everything(), ~sum(is.na(.)))) %>%
  pivot_longer(everything(),
               names_to = "Variable",
               values_to = "Missing_Count")

ggplot(missing_pattern,
       aes(x = reorder(Variable, Missing_Count), y = Missing_Count)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Missing Data Pattern",
       x = "Variable",
       y = "Number of Missing Values")
```

Building the Logistic Regression Model

The `tidyverse` and associated packages make model building and interpretation more straightforward:

```r
# Fit the model
sleep_model <- glm(probsleep ~ sex + age + fallsleep + staysleep + hrswknight,
                   family = binomial(link = "logit"),
                   data = sleep_data)

# Get tidy model summary
model_summary <- tidy(sleep_model,
                     conf.int = TRUE,
                     exponentiate = TRUE)

# Create a nice summary table
model_summary %>%
  mutate(across(where(is.numeric), ~round(., 3))) %>%
  knitr::kable(col.names = c("Predictor", "Odds Ratio", "Std. Error",
                            "z value", "p-value", "CI Lower", "CI Upper"),
               caption = "Logistic Regression Results") %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

Model Diagnostics and Performance

The `performance` package provides comprehensive model diagnostics:

```
# Check model performance
model_performance <- check_model(sleep_model)

# Get various performance metrics
performance_metrics <- model_performance(sleep_model)

# Plot ROC curve
pred_data <- sleep_data %>%
  mutate(predicted = predict(sleep_model, type = "response"))

ggplot(pred_data, aes(d = as.numeric(probsleep) - 1, m = predicted)) +
  geom_roc(n.cuts = 0) +
  style_roc() +
  theme_minimal() +
  labs(title = "ROC Curve for Sleep Problems Model")
```

Understanding Variable Effects

The `marginaleffects` package helps us understand how our predictors influence sleep problems:

```
# Calculate average marginal effects
marg_effects <- avg_slopes(sleep_model)

# Plot marginal effects
plot_slopes(sleep_model,
           variables = "hrswknight",
           condition = "staysleep") +
  theme_minimal() +
  labs(title = "Effect of Sleep Hours by Staying Asleep Status",
       x = "Hours of Sleep per Weeknight",
       y = "Predicted Probability of Sleep Problems")

# Create effects plot for categorical variables
plot_model(sleep_model,
          type = "pred",
          terms = c("staysleep", "fallsleep")) +
  theme_minimal() +
  labs(title = "Interaction between Falling and Staying Asleep",
       y = "Predicted Probability of Sleep Problems")
```

Visualizing Model Results

`sjPlot` provides excellent visualization tools for model results:

```
# Plot odds ratios
plot_model(sleep_model,
          show.values = TRUE,
          title = "Odds Ratios for Sleep Problems") +
  theme_minimal()

# Create predicted probabilities plot
plot_model(sleep_model,
          type = "pred",
          terms = "hrswknight",
          title = "Effect of Sleep Hours on Problem Probability") +
  theme_minimal()
```

Advantages of Using Modern Packages

These modern R packages offer several benefits over base R:

1. The `tidyverse` provides:

- More intuitive data manipulation
- Enhanced visualization capabilities
- Consistent syntax across operations

2. `broom` offers:

- Standardized model output formats
- Easy extraction of model statistics
- Simple integration with other tidyverse tools

3. `performance` gives us:

- Comprehensive model diagnostics
- Easy-to-interpret visualizations
- Multiple performance metrics

4. `marginaleffects` enables:

- Intuitive interpretation of variable effects
- Beautiful visualization of interactions
- Straightforward prediction scenarios

Creating a Complete Report

We can create a comprehensive analysis report using these tools:

```r
# Function to create a complete model summary
create_model_report <- function(model) {
  # Model summary
  summary_stats <- glance(model)

  # Coefficients and odds ratios
  coef_table <- tidy(model, conf.int = TRUE, exponentiate = TRUE)

  # Predictions
  pred_data <- augment(model, type.predict = "response")

  # Return as list
  list(
    model_stats = summary_stats,
    coefficients = coef_table,
    predictions = pred_data
  )
}

# Generate report
model_report <- create_model_report(sleep_model)

# Print key findings
cat("Model Performance Metrics:\n")
print(model_report$model_stats)

cat("\nKey Predictors:\n")
print(model_report$coefficients)
```

Practical Implementation

Here's how to use these results in practice:

```r
# Create a prediction function for new cases
predict_sleep_problems <- function(new_data) {
  predictions <- predict(sleep_model,
                        newdata = new_data,
                        type = "response")

  tibble(
    probability = predictions,
    predicted_outcome = if_else(predictions > 0.5, "Yes", "No")
  )
}

# Example usage
example_cases <- tibble(
  sex = factor(c("Male", "Female"), levels = c("Male", "Female")),
  age = c(30, 45),
  fallsleep = factor(c("Yes", "No"), levels = c("No", "Yes")),
  staysleep = factor(c("No", "Yes"), levels = c("No", "Yes")),
  hrswknight = c(7, 6)
)

predictions <- predict_sleep_problems(example_cases)
print(predictions)
```

8 Summary

This lab introduced you to performing a variety of statistical tests and analyses in R (similar to the SPSS Lab 2). You can download the respective R scripts for Exercises D+E+F+G, H, and I below for comparison, in case you created your own:

- R script for Exercises D, E, F, G
- R script for Exercise H
- R script for Exercise I